

Points of View

Points of View

a tribute to Alan Kay

Edited by
Ian Piumarta &
Kimberly Rose

Copyright © 2010 by the editors and contributors
All rights reserved.

This work is licensed under the Creative Commons
Attribution–Noncommercial–Share Alike 3.0 License.
<http://creativecommons.org/licenses/by-nc-sa/3.0>

Designed and assembled by the editors in Los Angeles, California.

Published by Viewpoints Research Institute, Inc., Glendale, California.
<http://www.vpri.org>

Printed and bound by Typecraft Wood & Jones, Inc., Pasadena, California.
<http://www.typecraft.com>

ISBN 978-0-9743131-1-5

11 10 9 8 7 6 5 4 3 2 1

Opinions expressed herein are those of the individual contributors and do not
necessarily reflect those of the editors or publisher.

Contents

Preface	v
Bob Sproull	
Alan Kay: visionary designer	i
Ivan Sutherland	
Old Salt Lake stories that you may not have heard	5
Adele Goldberg	
Alan Kay and the Search for the Holy Grail	9
Bert Sutherland	
Manager as Pupil	29
Bob Stein	
Do it	35
Leonard Kleinrock	
About an Ageless Alan Kay	45
John Sculley	
Genius is Seeing the Obvious Twenty Years Ahead of Everyone Else	49
Bobby Blatt	
The Vivarium—a place to learn about learning, and to think about thinking	55

Chunka Mui	
Notes on a twenty-five-year collaboration	63
Mel Bergstein	
Context, Inspiration and Aspiration: Alan Kay's Influence on Business	73
Larry Smarr	
The Emergence of a Planetary-Scale Collaboratory for Data-Intensive Research	79
Andy van Dam	
Reflections on what Alan Kay has meant to me, on the occasion of his 70th birthday	97
Raj Reddy	
Alan Kay and the Creation of the Centre Mondial Informatique et Ressources Humaines in Paris	103
Nicholas Negroponte	
The Book in Dynabook?	107
David P. Reed	
Get the verbs right	113
Chuck Thacker	
A Tiny Computer	123
Douglas B. Lenat	
The K Factor	141
Butler Lampson	
Declarative Programming: The Light at the End of the Tunnel	151

Vishal Sikka	
Some timeless lessons on software design	165
Vint Cerf	
Thoughts on Alan's 70th Birthday	173
Mitchel Resnick	
Life as a Learning Lab	177
Bran Ferren	
AK—A Graphic Exposé	183
Betty Edwards	
A Tribute to Alan Kay	193
Bob Lucky	
Portraits of Alan Kay	197
Greg Harrold	
Greg and Alan conspire to create a wonderful new pipe organ	205
Quincy Jones	
A three-sixty human being	217
Gordon Bell	
Dear Alan, Re: What about your digital afterlife?	225
Danny Hillis	
The Power of Conviction	233
Afterword	241
Bibliography	245

Preface

What do you give a man who has everything?

I started thinking about Alan's 70th birthday only days after his 69th. (Alan won't find this surprising. I am the "early-binder" in our group.) I was fortunate to be a part of Alan's 50th and 60th birthdays; his 50th a grand and spectacular celebration, the other small, intimate and low key. Alan would be the first to remind us that the number 70 has no greater or lesser significance than 68 or 73 or 42. (Well, maybe 42 *is* somewhat special.) He would also point out that May 2010 is really the end of his 70th year, not the beginning. We place significance on multiples of ten, but again Alan would point out that this is no coincidence to the fact that humans have ten fingers and ten toes. If we had only eight, we would have celebrated this particular birthday 16 years ago.

In any case, although Alan downplays these milestones for himself, he has been quick and ready to remember the occasions for others and has helped organize events for friends and colleagues. I've had the joy, pleasure and occasional frustration—see Bert Sutherland on expense reports—of working beside Alan for 24 years. For this occasion I wanted to do something unique for him. I didn't want to be a consumer and purchase some ready-made item. There's not much he doesn't have, or really wants for that matter. Instead, I wanted to create something that would be entirely unique. I wanted to be a "literate" gift-giver. One who is fully literate is not only a passive consumer of other people's

goods, but an active generator of their own. This is something about literacy Alan taught me long ago. I've worked on my consumption-to-production ratio over the years, and while it has improved it still disappoints me as it remains far heavier on the consumption side. Since I am not a programmer I knew I could not create an artistic expression on a computer, something I know Alan would appreciate and enjoy. As I continued to ponder I decided to draw upon my trait as a producer in an organizational sense and to use this talent and the many connections I've made over the years to "produce" something unique for Alan.

As I started to imagine what I might create I recalled a book produced for SAP's founder, Hasso Plattner, upon the occasion of his 60th birthday to which Alan contributed a piece. *Realtime* [1] became my inspiration and I decided *that* was it: I wanted to create a book.

I discussed the idea with Ian Piumarta who thought it a good one and immediately agreed to work beside me on the project. Ian's knowledge in the areas of book design, editing and typesetting were the perfect companion talents to my "hunting and gathering" of contributions from Alan's friends and colleagues. I felt confident that we could produce an artifact of beauty and value not only to Alan but to any reader looking for historical background and insights into Alan, his achievements and philosophies.

It may seem ironic to be creating a physical, hard-bound book for the man who imagined the Dynabook. A few contributors even suggested that we should produce something more dynamic, more "twenty-first century," more accessible to more people, and then deliver it over the Internet, but we held fast. Alan and I share a common love—the book, the Basic Organization Of Knowledge—and I believe the printed book remains Alan's favorite medium. We also suspect that this physical book may survive on Earth longer than any software program that could read it, or storage device that could house it. As Gordon Bell will share with us, Alan's personal library now holds over 11,000 books. Traditionally Alan, Ian and I exchange the newest publications, or our

favorites from the year, for birthdays and for Christmas. For this birthday we planned to present Alan with one book he couldn't possibly have bought for himself.

We began this book in the summer of 2009. I knew the people I would contact were extremely busy with too many commitments already, but that was not going to stop me. I wanted to contact a variety of colleagues from Alan's past and present, not only from the fields of computer science and information technology but from other areas so important to Alan such as art, education and music. Moments after hitting *send* on my initial e-mail query I received a flurry of positive overwhelming response: "Yes!" "Count me in!" "Great idea!" "Would be delighted!" It was so exciting. The responses from his long-time colleagues fueled my mission.

As you read the contributions I think the many personas of Alan will come to light—visionary, scientist, mentor, life-long learner and polymath. Although I have worked alongside Alan as manager of our research group, within a variety of corporate organizations and now in our own non-profit organization, many of these essays taught me even more about the man I feel I know so well. Some recollections made me chuckle, others brought tears to my eyes as I was reminded, once again, how Alan has affected so many in deep and profound ways and as I recalled some of our own experiences together.

Alan, with this book, Ian and I present to you, on behalf of several of your dear friends and colleagues, a tribute wherein we hope you will also learn something more about what you have given to us, taught us, and been to us—all of us who participated in this project. We believe this book also contains valuable lessons and historic information that will be of interest and value outside our circle and hope we can bring some of the remarkable influence you have had and lessons you have taught to a much larger group of people.

We extend our deep and heartfelt thanks to each contributor to this volume. You were all wonderful to work with and it was obvious that each of your contributions was a labor of love.

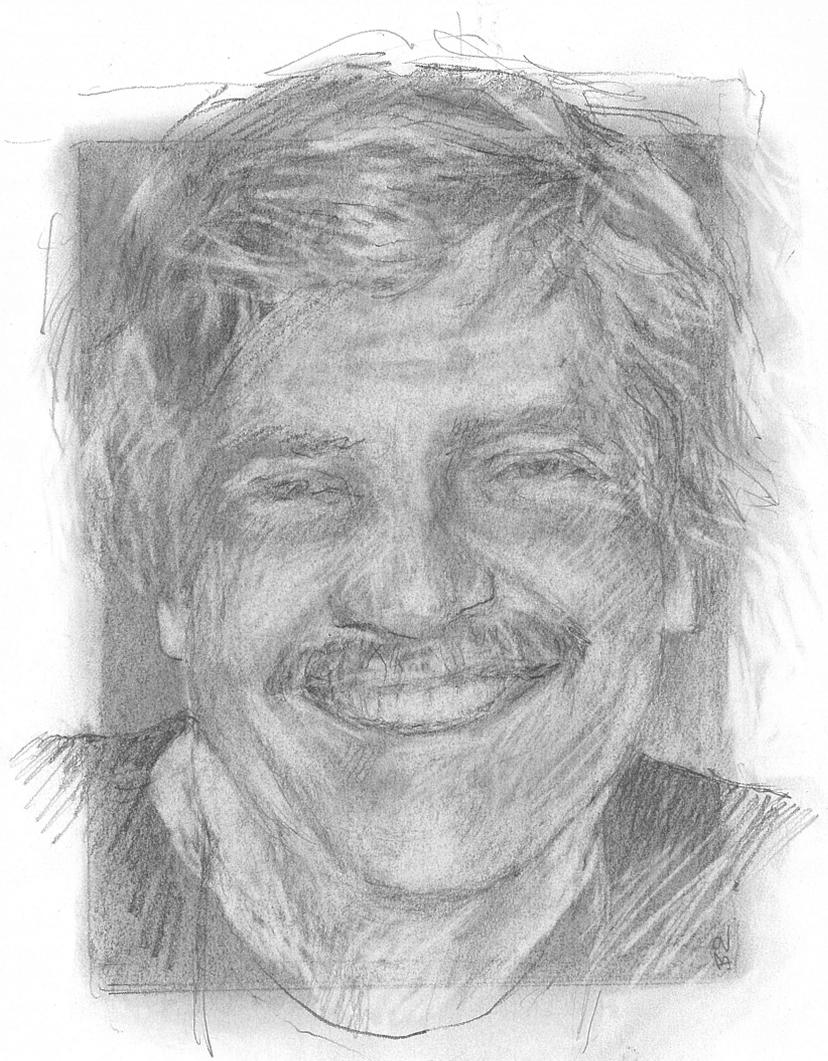
Happy 70th, Alan. We didn't worry about what anyone else was giving you: the best way to offer a unique birthday gift was to invent our own!

With love, thanks and gratitude for all you've done and been to so many.

Kim Rose
Los Angeles
February 2010

As I was doing this drawing of Alan,
I kept thinking,
“What does this smile remind me of?”
Then it came to me:
the Cheshire Cat.

Betty Edwards
January 2010



Bob Sproull

Alan Kay: visionary designer

Many people write computer programs. Many people make tools for others by creating computer applications. More than a few devise new computer languages. Many have “visions” of the future of applications and technology. Why do Alan Kay’s contributions eclipse all this? Because he’s a successful designer in the best sense of the word: he’s not a geek foisting his world on you; nor is he a bubblehead dreaming of worlds that can never be. Instead Alan works hard to sculpt a computer tool to be useful for you. So his achievements—and there are many—resonate like a beautiful product or fine woodworking tool.

Throughout his career, Alan has swept us along by his vision of the best personal computing could be. He has shown compelling visions, and—importantly—has brought them to reality. Enough of a visionary to imagine a different, better world; enough of an engineer to trace a path to the goal. An important feature of Alan’s visions, one that differentiates them from the huge number of “visions” in our industry, is that they are very fine designs. We don’t ask whether his visions meet a set of requirements, or solve a specific problem, or have the right cost/benefit tradeoff, or expect a committee or a community to make them real. Instead, we lust after them because we want to have them.

Our industry has remarkably few good designers. Some recent product designs are superb and may herald hope that good design will be valued by

users. But Alan’s visions are different from product designs—they require an imagination about what cannot yet be done. And the unwavering courage to ignore appeals to be more practical.

Alan was never ashamed of Smalltalk’s speed; he called it “stately.” But of course its speed alone exempted it from consideration as a “serious” programming language until, predictably, implementation techniques and micro-processor speeds became adequate. But neither Alan nor the Smalltalk team compromised the design while waiting for it to become fast.

Overlapping windows are another example. Unlike the alternative tiled windows, any implementation will have to redraw images in portions of windows that are newly exposed when an obscuring window moves. This will either be slow or use a lot of memory to store the obscured image. Again, with time, implementation techniques and computer hardware let the better design succeed.

Alan doesn’t confuse good design with perfection. Smalltalk’s early raster graphics displays, drawing editors, and animations appear crude because of low resolution, lack of color, and slow updates. You could print raster images only by hammering at the period on the daisy wheel of an early Xerox printer—and wearing out the wheel in a few hours. But these demonstrations were not to replace professional animators, rather to offer a means to express creative ideas—often by kids. The great appeal of computer graphics (and what hooked me on the field early on) is that they best reveal the computer’s simulation—to see what’s going on, to find the bugs where the simulation goes awry. To Alan, a computer’s unique capability is as a versatile simulator, and graphics are simply essential.

Alan’s visions are not only stakes in the ground like the Dynabook, known by the sketch of a computer-as-display resting in the lap of a child sitting under a tree. His visions also come in smaller units, refreshed every time he speaks. He has a way of expressing tantalizing ideas that are just beyond what you can see how to do or see how the ordinary course of technology development will achieve.

It's actually taken us a while to understand what Alan says. I remember in the early days of Smalltalk he described operations on objects such as, "the message plus four is sent to three" and I knew enough about how the implementation worked to detect no such thing. (The phrase stemmed in part from the popularity at the time of *actors* as a form of object-oriented computing.) Today, the locution is unremarkable. In effect, Alan has had to wait for all of us to catch up on object-oriented computing to "get it."

Alan keeps teaching us and showing what we must strive for. He tirelessly draws our attention to the designs that taught and inspired him: Sketchpad, for its graphics, data structures, object orientation (though not called that), and simulation by constraint satisfaction; NLS (Doug Engelbart's system) for its interactive collaboration, structured documents, and genuine exploration of ways to augment human capabilities; and Simula, the progenitor of object-oriented programming in the context favored by Alan—the computer as simulator. Not surprisingly, these are designs of visionaries not unlike Alan, ahead of their time. Alan says—and he's right—that these designs demonstrated important capabilities that we still don't enjoy routinely today.

Alan has not declared success. Despite wide use of personal computers, object-oriented programming, window systems, and other capabilities he developed or promoted, he's still painting just-outside-our-reach visions. Now he talks of building simulations collaboratively, with dispersed teams and virtual environments. And his Turing Award speech is a reminder that, "we haven't even started yet:" we have to get kids interested in the romance, the art form of our field. "It's our duty to help the children, as young as possible, to do a better job than we have." We need to give kids tools that are both simple and deep. Simple superficial behaviors, like "turtle forward 20," and deep capabilities to change the surface behaviors and tackle harder problems. Shouldn't all computers work that way?

Robert F. Sproull met Alan Kay at Harvard in 1968, when Alan visited Ivan Sutherland's research group. Alan was then a graduate student at the University of Utah—eager, full of questions, and unforgettable. Sproull was an undergraduate, wire-wrapping and debugging a rack full of hardware to implement the clipping divider.

Bob continued to build hardware and software for computer graphics, and to design asynchronous VLSI systems. He was a principal with Sutherland, Sproull & Associates, an associate professor at Carnegie Mellon University, and a member of the Xerox Palo Alto Research Center. He is a member of the National Academy of Engineering and a Fellow of the American Academy of Arts & Sciences.

He is currently a Vice President and Fellow at Sun Microsystems, Inc., serving as director of Sun Microsystems Laboratories.

Ivan Sutherland

Old Salt Lake stories that you may not have heard

Ed Cheadle was, according to one pundit we may both know, “the best engineer in the Salt Lake valley.” He worked for the Evans and Sutherland Computer Corporation, but I think you worked with him before that and knew him well. He’s dead now, but his memory lives on. Perhaps you were even the source of the “best engineer” description. Anyhow, here is a true story about Ed Cheadle that you may not have heard.

Cheadle played volleyball with us many afternoons at Evans and Sutherland. There was another game at the University, but I don’t recall your participation. Cheadle made a point of remembering only those games in which he played on the winning side. A grand plan for life—remember only the positive and forget the negative.

One of Cheadle’s engineering masterpieces was a large line-drawing cathode ray tube display of extraordinarily high quality. It used magnetic deflection. That meant that the deflection system had to cope with the inductance of the deflection coils, no easy task. To manage that, Cheadle arranged that the deflection system would start a short distance ahead of where the line was to begin. The deflection system then got “up to speed” so as to trace the line exactly straight and uniform. When the deflection system got to the start point of the

actual line, the stream of electrons forming the CRT beam turned on, making a perfect start to each visible line.

When reaching the end of each line, the stream of electrons cut off to make a perfect end to the visible line, but the deflection system overshot the end point so as to leave no trace of error. Only the proper parts of the line showed on the screen because the CRT beam was off during the “up to speed” and “overshoot” extensions.

The result was flawless, by far the most beautiful line-drawing display ever built. Indeed it was a masterpiece. Every line was perfect. End points matched exactly. It was a thing of rare and exquisite beauty.

One day Marcia and I went for a tour of Evans and Sutherland. We had, by then, moved to Santa Monica, but were in Salt Lake on a visit. In the laboratory we found Ed Cheadle showing off his beautiful display and surrounded by an admiring group of younger engineers. On the face of the display was a line drawing of an F-4 airplane rotating in real time. It was impressive indeed, with every line as perfect as it could be and the motion perfectly smooth.

Cheadle and his team were beaming with pride. One could feel the admiration of the younger folks for Cheadle and the pride of all in a magnificent accomplishment. There was spirit in the room.

Marcia, knowing full well what it took to make such a beautiful picture, said, “Ed, that would be a truly beautiful picture if you could only get it to stand still!”

Alan, some years later I was reading the in-flight magazine for TWA. (Remember them? They used to be an airline.) In it I found an article by you, or interviewing you, that talked about Sketchpad. Strange place to find the article, but not strange that you should mention Sketchpad—you are one of the few people who actually read my thesis, and maybe the only one who understood it.

Cambridge University scanned one of the hundred real original Sketchpad thesis books; not the Lincoln Labs report, but the original black-cover version

that I published. Accept no substitutes; you can recognize the original because it has two pages numbered 106. You can now find that original version in the archives of Cambridge University, and a transcription of it on their website. The transcription is carefully marked to show where the original pages begin and end. At least I know that in addition to you having read it, some computer in England has also read it and at least recognized all the characters in it. My daughter, Juliet, tells me that an OCR error is called a *scano* rather than a *typo*.

You and I both worked for Dave Evans. Early in my Salt Lake experience the Evans and Sutherland Company put our first line drawing system on the market. I named it LDS-1, but not without first checking with Dave that he had no objections. He took some time to think about it and returned the next day saying, "it's not offensive and it might be memorable, so why not."

Some years later an informant told me that he'd quizzed Dave about that incident, wanting to know how the LDS-1 name had come about. Dave admitted that the pun was my idea but that he'd approved it. The informant reported to me that when seeking out Dave's feelings about the topic, Dave had said, "I think they were testing me."

David's father, also David Evans, founded and ran the Evans Advertising firm in Salt Lake City. They were the largest U.S. advertising agency outside of New York City. One of the things David Senior was proud of was the built-in thermostat that comes in many turkeys. It's a little plastic thing that pops out when the turkey is fully baked. Evans Advertising had them developed to help sell turkeys.

One day I took Dave Senior to lunch. "Where shall we go?" he asked. I suggested Lou Dornbush's Deli. You may recall Lou; he was five feet in any dimension you might care to measure, and strong of character. He had a numbered tattoo acquired in wartime Germany. Did you know, by the way, that our Dave Evans was one of the first U.S. soldiers to enter Auschwitz? I have a copy of a thank-you certificate presented to him by a group of grateful detainees.

Anyway, we went for lunch. David Senior asked what he should order. When I suggested the lox and bagels, he had to ask what that was. I suspect he was the only advertising executive who ever had to ask that particular question.

I wanted to make an advertisement for LDS-1. I asked Dan Cohen to make a program that would display some text, but failed to specify the language of the text. It came back in Hebrew. What the heck, that's memorable, so I sent it off to Evans Advertising to make the advertisement.

When the proof came back the picture on the screen was left justified. I knew just enough about Hebrew to suspect an error, so I checked with Evans Advertising to see if they knew what was up. They didn't, but John Dwan from the agency and I both knew Lou Dornbush and were sure he could give us the answer we needed. So we went to seek advice at the Dornbush Deli. Lou looked at the ad, looked at me, looked at John, looked at me, and said to John, "It's backwards! That will be \$150." True story, I was there.

Alan, did you ever wonder how you got admitted to graduate school at the University of Utah? You have to admit that your credentials at the time were a bit irregular. Would a sane University have admitted such a student?

One time Dave and I talked about your admission. He expressed his belief that every graduate student class should have at least one outlier, someone who doesn't fit the mold but seems bright enough to succeed. Dave believed, correctly I think, that such outliers have a better than even chance of leaving a mark on the world.

I like the mark you've left.

Ivan Sutherland received his Ph.D. from MIT in 1963 and has taught at Harvard, The University of Utah, and Caltech.

He spent twenty-five years as a Fellow at Sun Microsystems, and is a member of the National Academy of Engineering and the National Academy of Sciences.

Ivan is currently a Visiting Scientist at Portland State University where he recently co-founded the "Asynchronous Research Center" (ARC). The ARC seeks to free designers from the tyranny of the clock by developing better tools and teaching methods for the design of self-timed systems.

Adele Goldberg

Alan Kay and the Search for the Holy Grail

Alan Kay challenged the computer industry's widely held presumption that computers were expensive, controlled-access, enterprise-owned devices. He took a very different, heretical-at-the-time, leap of faith when he proposed that anyone of any age would benefit from direct computer access. A computer should be a consumer device that is affordable, portable, and usable by anyone. Alan suggested that the real benefit in using a computer comes when anyone can tell a computer what to do, so as to have a personal thinking partner.

“Telling a computer what to do” translates into some form of programming, whether via interactive commands or persistently executing processes or computations. Letting anyone program increases expectations and requirements for real-time interaction, clear visual presentation of both construction tools and execution state, and reliably safe exploration of the effects of change. Users quickly learn to trust the computer to do the right thing. Fulfilling the expectation that everyone be able to use and program computers requires experimentation about the nature of computer-to-computer communications and human-to-human collaboration in a virtual world of real and imaginary agents. It also opens the question of whether everyone *should* be expected to learn to program. Alan wrote about this in his paper *The Early History of*

Smalltalk [7], in which he thinks out loud about the problems of teaching programming, the design aspects of telling a computer what to do, and the special quality that understanding computing brings to critically observing the world around us.

What Alan Kay liked to call the Holy Grail is a software paradigm by which anyone could be a programmer if he or she so chooses, much like anyone could be a novelist if taught reading and writing skills as a part of basic schooling. Alan's PARC research team—the Learning Research Group (LRG)—worked with children to see what they thought possible, couching educational interactions in the learning principles of Bruner, Piaget, and Bloom. I joined LRG in the summer of 1973 to explore the use of new approaches to programming with children and adults. We worked with the non-scientific PARC staff to understand what was easy and what was hard for adults to learn about programming, bringing to light concepts or actions that were often not hard for children. We worked inside the public school system to be directly informed of the possible political and economic barriers to widespread computer access. And we took the Xerox Alto workstations, running an early version of *Smalltalk*, to a junior high school in Palo Alto, brought them back to PARC the same day, just to return them the next day to the resource center we set up at the school. Alan was training me to do the right thing, to break down corporate barriers, to break rules! It turned out, of course, that we did not have permission to take Altos outside the PARC building. I tried hard to apply that early lesson to many future decisions.

LRG built a series of applications from the same software—programming languages, tools, and libraries of objects—that were offered to these children and adults alike. Research benefited from the discovery that everyone was more likely to become empowered if the applications code was itself readable and changeable and used the same system software used by the professionals. The solution needed to be a context in which to explore change with a low risk of bad outcomes and a high risk of pleasant surprise.

Kits: Modeling By Everyone

It would be a mistake to think that this notion of programming for everyone could be satisfied by any single programming language, and certainly not by one of the structured, procedural, 1960s languages. With “programming” Alan really intended “modeling.” A person achieves Alan’s goal if he or she can construct a computer executable model of a real world phenomenon, perhaps without understanding the detailed syntax and execution model of a professional programming language. The challenge question was (and still is): What combination of hardware, operating system, language, library, and user-level presentation and interaction enables general purpose *modeling*? Computer-implemented models of how a person believes the world works—in the small or in the large—could be shared, discussed, explored, modified, and plugged together, all to help build an inquiry approach to learning. With proper tools, models could become components of larger more complex models, which could then be explored through experimentation.

Why is this distinction between modeling and programming critical? It lifts our thinking about teaching from how to program functions—such as sorting names or calculating a bank balance or directing a robot to walk in a circle—to how anyone might express his or her understanding of how complex persistent interactions can be represented, simulated, and observed. It also emphasizes the creation of shared representations of what we can think of as the primitive objects from which new and interesting objects (aka models) can be constructed, as well as the framework in which the construction can take place. Thinking in this way allows us to view software much like a dictionary of words (the interesting objects) from which particular words are selected and connected together to form new patterns—a programmer’s “novel.” No one has to know all the words in the dictionary to write a story, no one has to know all the objects in a programming language library to start making new connections that tell the computer what to do. Just as no one ever has to know all the words in the dictionary to be skilled at writing interesting,

even award winning novels, no one ever has to know all the objects in the library to be skilled at telling the computer what to do. There is also a subtle cognitive change as the computer is no longer intimately involved. Models are constructed of objects, and programming becomes the task of directing the object components that make up the model to do what they know how to do. This software paradigm allows anyone to create new objects and add them to the library to be shared by others, which means that the range of what can be done by making new connections—constructions or compositions—is not limited.

The beginner learns basic skills: find objects that seem interesting, then explore what they can do and how they can be manipulated, changed, and connected to other objects. The library is initially small and consists of objects appropriate to the learner's interests and communications skills. As the learner progresses, the library grows with the addition of objects that represent new aspects of the real world and new ways to manipulate information in the simulated world. If the skills needed to take advantage of the library of objects are the same for learners of all ages, then we can imagine teaching a five-year-old child in essence what we teach a fifteen-year-old or a fifty-year-old. We offer each learner a different selection of objects from the library, because each learner has a different vocabulary and is interested in constructing a different model.

Alan used the word “kit” to name such a computer-based world for modeling. A number of kits have been created at PARC and elsewhere, with each one specific to creating models about business, science, mathematics, music, or even programming! Each kit defines the useful primitive objects, and the tools and visualizations for finding objects, making their connections, and observing their behaviors. Kit creation faces interesting challenges, especially when the goal is a kit whose applicability is open ended, as was certainly the case with Alan's Squeak/Etoys, Finzer and Gould's Rehearsal World [4], David Canfield Smith's Pygmalion [10] and KidSim [11] (which was the basis for StageCast), ParcPlace Systems' VisualWorks and LearningWorks, and Apple's Fabrik. But

also closed worlds—like the Simulation Kit I built in 1977–78 (to support discrete event-driven simulations), Randy Smith’s Alternative Reality Kit [12], and Alan Borning’s ThingLab [3]—offered important research opportunities. Kits provide a context in which to explore the creation of programming environments consisting of object connections that are always “alive” and visual displays of object capabilities. Each kit offers a way to inspect object interaction dynamics, locate objects for reuse in new connections, avoid making invalid connections, adjudicate entry into the library of reusable (primitive) objects, and so on.

Alan often explained the idea of an object by equating an object to a (simulated) computer. Like a computer, each object manages its own data and defines the methods for manipulating that data and communicating with other objects. A model, then, is a set of interacting objects that affect one another’s behavior and properties through mutual awareness and communication. The world of programming with objects is rich with respect to the details of how object definitions relate to one another, how object lifecycles are managed, how objects reflect on their own roles in forming a running application and, in the spirit of “software is data,” how objects change programmatically. At some point, whether a part of the static definition or the dynamic execution, an object’s data is determined and its behavior is modified by that data.

The Model’s Data

The two important questions for anyone wishing to create a model are: What data (properties) are required for the objects selected for the model, and how should the data be supplied to the objects? At the heart of the effort to commercialize the object paradigm was the need to provide a persistent store for the objects’ data. In a perfect world, computers just keep running and objects just keep interacting. And in this perfect world, there is no need for a static form for objects. Given that the world is not so perfect, one could imagine letting anyone talk to a computer without the concern that static snapshots of objects are regularly taken and serve as a persistent representation, ready to be marshaled

should there be a hardware or an operating system failure. This notion of regular snapshots was the approach taken in the earlier Smalltalk systems in the form of purely object-oriented virtual memory management (initially OOZE [5], implemented by Ted Kaehler, and later LOOM [6] for Smalltalk-80, by Ted and Glenn Krasner). The purist advocates of the object model pursued this approach, but also argued the case for making an explicit interface available to the programmer for storing objects external to the running kit. There were, of course, voices that argued for access to more standard SQL databases as the source for populating the data of the kit's objects, and also for the use of SQL queries as a part of the repertoire available to information-management objects. Whether the programming language system is Smalltalk, Java, C++, Python, or some other object-based language, some solution to static data store is provided. In the early days of commercializing object-oriented programming systems, the argument was often heated, as the outcome affected many enterprises already tied to SQL standards and procedural programming systems, while opening an opportunity for new object databases.

Information storage-and-retrieval plays a central role in the quest for Alan's Holy Grail. In the early 1980s, the (simulated) desktop metaphor became *the* way to organize application access. The graphical user interface looked (and still looks) like rectangular windows whose frames provide scrolling and editing tools for the content. Imagine that you have some information to be shared in multiple places—a calendar, an e-mail, an alert, a document. The desktop metaphor demands that the user open a window for each application and redundantly provide the information, possibly with the assist of cut-and-paste editing. The inverse approach would be to “tell” the information itself how to be shared or how to be processed. For example, you could attach post-its to multiple documents or you could write on a single post-it what is to be done. The latter seems more efficient, but requires that there be some model of how to interact with applications in a programmatic way rather than with direct manipulation. To the extent that the many applications available to the user are aware of the kinds of data to watch for, one could envision the

user simply providing the information to a data store that then triggers the applications to update. Specifying such awareness is an example of the kind of modeling users need to be empowered to create if the Holy Grail is to be found.

Experimenting with a data-driven modeling approach was a part of the research agenda when Alan's LRG started the Notetaker project in the late 1970s, so named because the challenge was to create a portable device that could substitute for note cards that students used to capture facts and references while browsing library stacks. The idea of carrying a computer to the library was a physical ideal to focus the team on creating something small and closer to the Dynabook [8] cardboard mockup that Alan carried to conferences (and which appeared so real that a number of conference attendees wrote to purchase one, with one attendee so unfortunately adamant that he threatened suicide if we did not deliver). That this library could be a virtual online collection was certainly understood; one Ph.D. dissertation from the group specifically studied how online books might be represented and searched using traditional indexing versus hyperlinks (Weyer's FindIt [15]). The particular model of interest for the Notetaker was the organization of the royal households of England and France and the ways in which interactions among the servants affected interactions between the heads of state.

The Notetaker was built in 1978. Larry Tesler took it on an airplane in coach (it was too big to fit under a first class seat). It was more a "luggable" than a "portable," but it ran a full Smalltalk system on a graphical display screen—on batteries. However, it did not yet run a model of the interactions between the royal households of England and France, nor did it make it easy to capture information while browsing in a physical library.

The Xerox strategists' apparent inability to see value in enabling technology—object-oriented software with a graphical user interface running on portable computers with flat panel displays—was disappointing. Technical documentation for Xerox machines, at that time, was developed on a main-frame computer and based on a parts inventory database. It was then printed

and bound as books that filled the trunk of a standard car, and, therefore, could not easily be brought into a customer's office or plugged into a diagnostic network built into the customer's copier/duplicator. The Xerox decision makers did not see the advantages that a small computer, loaded with documents and a browsing interface, could bring to the copier-duplicator service technicians. An LRG on-demand publishing proposal astonished the Xerox publishing group. And the 1982 design for a 68000-based personal computer running Smalltalk-80, aptly named Twinkle (think 1984 Mac), that could serve as an introductory computer to pave the way for the Xerox Star Workstation, was met with disinterest. These rejections contributed to a diaspora of LRG team members. Fortunately, people departed after writing several seminal articles for the August 1981 issue of *Byte Magazine*, completing the Smalltalk-80 system, starting a university licensing program, building a cooperation among five corporations to specify the Smalltalk-80 implementation on standard processors, and writing three books on the language, the user interface, and the various implementation experiences. Some of us decided to go commercial and created a venture-capital-backed spin-off called ParcPlace Systems, which I led as founding CEO and President from its inception to 1991.

Modeling Modeling

Was there a missed idea that could have led to a better solution for the modeling goal? Perhaps. Consider how people learn: Look at some data, build a model of what the data means, present the model to others who look for additional data to confirm or challenge the model, iterate. Many very smart people, who gathered in New Hampshire at one of Alan's off-site Learning Labs, testified that this approach was what they recalled using as young learners. They particularly recalled that parents served as the sounding board for their models, suggesting new data that led them to rethink their models again and again. If this data-driven way of creating a model is really valid, then perhaps there exists a different way to define a software modeling kit for everyone. We could

build a kit in which the specification of a model combines the structure of the data (perhaps selected from a library of data design templates or existing data structures) with patterns for queries and actions associated with query results. Data challenges would come from re-using already populated data structures to test new query patterns and actions, or from data-networking (think social networking in which data challenges to proposed models substitute for text messages). Actions could range from simple screen presentations, including animations, to complex computations.

A spreadsheet fits this view. The row-and-column model relating static data to computations can be specified by linking cells to an actual database. So the idea is not entirely new. The success of the spreadsheet programming paradigm supports a data-focused approach to modeling. But a spreadsheet is also simplistic in that its underlying computation model is functional. The more general approach would link externally-acquired data with object representation, behavior, and connections.

A learning kit fits this view, especially one that connects to external sensors—sound, light, touch, and so on. Mitch Resnick’s Scratch (built on Alan’s Squeak) makes interesting use of external sensors as stimuli for the behavior of Scratch sprites. With an emphasis on community learning/collaboration, creating Scratch programs and sharing them become an important part of the experiences and expectations in the use of Scratch. The San Francisco Exploratorium experimented with giving visitors sensors to collect data from exhibits and create personal websites for home use to explore the implications of their collected data. Game design incorporates external sensors, for example, in creating game controllers. And of course Alan’s Squeak/Etoys on the XO has a rich choice of physical world sensors such as sound, camera, and input devices that can be incorporated into the programs written using Etoys. In the spirit of the Model-View-Controller paradigm discussed later, Etoys includes the vision that multiple user interfaces are important in order to accommodate non-readers as well as older children who can handle a greater number of objects in the library and more interaction complexity. Both Etoys and Scratch

assign an important role to a companion website as the focal point for social networking, including the ability to find community contributions for reuse.

These examples share strong similarities. The learner is building a model on the computer to be verified with real world behavior, grabbing data from that world, and further deriving the model from this evidence. The learner is also sharing a model with other learners or reusing the models proposed by other learners.

The browser as a user interface to the Smalltalk programming language touches on the idea of a model as a specification of query. The code browser is a specialized retrieval interface. The programmer categorizes objects and object methods while providing their definitions. The categories define a browsing index. The browser incorporates tools to search for implicit relationships among objects, for example, find all objects that can respond to a particular message, or find all objects that rely on sending a particular message. Smalltalk code browsers work with static state, whereas Smalltalk inspectors browse dynamic state. Ted Kaehler added an Explain feature which allows a user to select any variable or punctuation in Smalltalk code and receive an explanation of it in the context of its usage. Every kit has a form of retrieval built in, but most support only searches to find primitive or augmented objects in the library.

After Alan's 1979 departure from PARC, LRG morphed from a group contained within a research laboratory into the Systems Concepts Laboratory (SCL) with me as its manager. By the early 1980's, both Apple and IBM succeeded in introducing personal computers to the market that clearly benefited from LRG's research. Having made the point in the 1970s and early 1980s that personal computing was not only feasible but desirable, SCL turned its research attention to the "interpersonal"—investigating how creative team members can be supported in their effort to collaborate, especially when some of those team members work remotely. In 1984, a second physical location for SCL was set up in Portland, Oregon—in the same time zone as PARC in Palo Alto, California, but not a quick car ride away. A 24-hour video link with speaker phones

connected one site to the other and gave a sense of constant shared presence. Software applications augmented the desktop to allow virtual “door knocking” and synchronous conversation. This work was the beginning of what we know today to be online collaboration systems or “computer supported cooperative work” (CSCW), and was the seed of online project management support. Our research shifted from language systems development to understanding how to capture and share work processes and artifacts. SCL researchers included trained architects, such as Bob Stults and Steve Harrison, and computer scientists focused on new media and user interface techniques, such as Sara Bly. Their “ShopTalk” and “Media Space” [13, 14] were efforts to understand the integration of design models with design processes, starting with capture, storage, and indexing video and audio as a way to represent the processes that led to design representations.

Going Commercial

Alan used to tell LRG that it is not a good idea to have customers—customers expect to be supported. Worse, customers require backward compatibility, which is, of course, limiting when you are doing research and need to leapfrog yourself. But SCL had a surprise customer brought in by a Xerox government contracting division, XSIS. The XSIS customer was the United States Central Intelligence Agency (CIA) on a mission to invent a new way in which analysts would work. It was 1979, and graphical user interfaces were not as broadly known and used as they are today, nor was there today’s abundance of choices for programming environments whose libraries facilitate building graphical user interfaces. The CIA’s technology thinkers were willing to take a chance with a research result, although it is likely they assumed at the time that Xerox would productize the Smalltalk system software and package it with appropriate hardware. Indeed, at one point, they indicated an interest in placing an order for 6,000 units. On first discovering that XSIS was contracted to develop a new Smalltalk-based Analyst Workstation, I paid a visit to the

National Photographic Interpretation Center (NPIC) to see a demonstration of an early prototype. The NPIC programmer at the time was a professional photographer who clearly believed the LRG marketing pitch that you can just browse around and read the code, find things you want to use, and connect them. Our programming environment was a kit for creating applications whose user interface could be made out of the same components we created to construct programming tools. The new NPIC code written to connect existing components was not an example of good programming style, but, as the photographer/programmer said, it worked, it demonstrated what he wanted, and someone else (in this case, XSYS engineers) could easily rewrite and extend it. What mattered is that the code was *readable*, and both the browser with categories and the debugger for interrupting a successfully running process helped him find what to read. The examples found in the programming system itself gave him a starting point that he could modify.

We had a customer we did not choose to have, but a customer who nonetheless proved to us that we had created an empowering artifact whose creative potential we ourselves did not yet fully understand. We also had a development process good enough to interest commercial adoption, i.e., programming by iterating on revisions of a running prototype. Thus was born a desire to create a version (eventually named Smalltalk-80) that would be broadly released and to do so with the cooperation of implementation innovators from universities and companies. With the help of the then PARC Science Center manager, Bert Sutherland, we enlisted programming teams at Apple, DEC, Tektronix, and Hewlett-Packard to test a reference implementation for this new version of Smalltalk. We also collaborated with several universities including the University of California at Berkeley (Dave Patterson and David Ungar), the University of Massachusetts (Elliot Moss), and the University of Illinois (Ralph Johnson). The success of this coordinated effort culminated in the three Smalltalk books (more about an unwritten fourth book later) authored or edited by myself, David Robson, and Glenn Krasner. The reference implementation for the Smalltalk-80 virtual machine was included in full in the first

book, the second detailed the user interface, and experiences implementing on various hardware systems were documented in the third. Having a committed customer also motivated, although it did not define, the formation of a venture-backed spin-out from PARC appropriately named ParcPlace Systems. (Peter Deutsch, Chief Scientist for the new company, dubbed us “ParcPlace Holders” during the eighteen months we negotiated the spin-out with Xerox.)

We implemented the Smalltalk virtual machine on Xerox proprietary workstations (including the Star Workstation) and on 68000-based workstations (we were very early adopters of the new Sun Microsystems workstations). Applications written on one workstation (Unix-based, Mac, Windows) always ran immediately and identically on any other supported workstation, either using a default user interface we created or leveraging the graphical windowing system of the vendor. Smalltalk exemplified the goal of “write once, run anywhere” in an age where that was almost never the case. It was the best of times and the worst of times to be in the object business. Despite a “pull” market from large enterprises desperate to have more predictability in building and maintaining software systems, the object paradigm was new and required a “push” marketing effort and a willingness on the part of our customers to retrain their programming staff. Part of the uphill push was to counter the hype that often accompanies something new, and make sure that our customers understood the transition in both the software architecture and team-development processes that they would be undertaking. We thus capitalized on our historical interest in education to provide professional development for customers and ultimately to develop formal methodologies for “finding the objects” and designing for reuse (Object Behavior Analysis [9] with its set of tools was packaged as a product called Methodworks). Kenneth Rubin led the professional services team that created both the methodology and the tools.

The shift in focus from K-12 education to professional programming teams immersed us in the rich and complex requirements these teams expect of a robust and stable programming language system, a different world from that of the typical one- or two-person PARC research teams. ParcPlace Systems

benefited greatly from the vibrant community of entrepreneurs that provided products and solutions compatible with our own offerings, such as the ENVY version management system and an add-on kit for interfacing to SQL database servers. Among the important requirements met by the commercialization effort were:

- creating a library of generally useful abstract objects;
- creating a library of concrete objects useful in specific industries;
- including robust exception handling as a part of the programming language;
- providing an object framework for rapidly creating GUIs portable across hardware systems (a kit named VisualWorks);
- providing real-time debugging tools;
- providing tools for coordinating teams;
- providing an applications (object library) portability layer;
- providing a way to manage persistent data;
- educating both programmers and their managers on the importance of designing for reusability and developing rapid prototypes to get early target user feedback.

Our experience with commercial usage of our system taught us an interesting “lesson learned” that affected market positioning. Early sales efforts emphasized the value of an object-oriented approach on predictable change—reliable, safe maintenance as requirements evolve over the lifetime of a software system. Unfortunately, this benefit could not be appreciated in an enterprise whose programming staff was not completing its systems to the point at which maintenance and change matter. A marketing shift to rapid, agile development produced more immediate sales attention!

At an early point in setting the company strategy, we thought to build on the success of the CIA Analyst Workstation. The core concept behind the Analyst Workstation was an integrated approach to improving understandability of data collected by analysts, especially data representing events over time. Data

could be brought into a new form of spreadsheet with embedded multiple viewers and analyses. At the time, the software system developed by the CIA with XSYS was a breakthrough whose features now exist in modern spreadsheets. A spreadsheet cell could be any object. A cell was not constrained to just hold numbers or text; it could hold anything represented by an active object. The object could be dynamically monitoring new data. The rows-and-columns of the spreadsheet's graphical layout provided a way to view a collection of objects. This graphical view was itself an object looking at a collection, that is, a spreadsheet was a kind of collection object. A single cell of one spreadsheet could embed another spreadsheet or any aspect of another spreadsheet, thus creating a nested collection of collections with no limit. A CIA analyst could use the spreadsheet tools to literally drill down from a cell into the supporting objects much the way a Smalltalk programmer used inspectors and debuggers to explore the dynamics of a running system.

A commercial version of the CIA Analyst was an attractive product idea for ParcPlace Systems, but it was not a market with which we were familiar. Moreover, we still would have had to provide the engineering and technical support for the underlying Smalltalk system. Our first priority was to realize an early revenue return for our core engineering work. In retrospect, an analyst-like kit could have provided an interesting modeling language for our enterprise customers.

ParcPlace Systems customers built application-specific kits for their enterprises. These kits served the dual purposes of giving enterprises flexibility in configuring their internal processes as well as opportunities to learn about how graphical interfaces reduce user training expenses when deploying new applications. The various kits ranged from new systems to manage payroll at Chrysler to new ways to manage package tracking at FedEx to an entire language system for creating new financial instruments at J. P. Morgan. This focus on application-specific kits was enabled by the Smalltalk-80 Model-View-Controller (MVC) architectural pattern and implementation. Embedded within the Smalltalk-80 programming system is a library of objects designed

to act as Views of other objects (the Models). A View could be as simple as a bar chart or as complex as an animation of objects queued up in a discrete event-driven simulation. A Controller is an interface between the user who interacts with the View—to explore the model or to alter the view. Changing the view might signal a change to the model. The benefits of this division into three roles for an application's objects are efficient reuse and ease of change. Libraries of Views and Controllers can be reused with different Models whose interface meets the interface requirements of the Views and Controllers. New Views and Controllers can be devised and linked to existing Models and Views respectively. For example, Controllers more appropriate to users with disabilities can be deployed for the same Views and Models. A View can filter what is shown to users, perhaps because users at different skill levels should see different age-appropriate details. Any object is really a Model, so the idea of viewing and controlling a Model can become quite complex. When the Model is itself graphical (such as text), the separation into the three aspects seemed at first to be a potential performance issue. It certainly was a source of animated discussion around whether to separate the View from its control. The earliest article introducing the MVC idea can be found in a 1981 *Byte Magazine* article by Trygve Reenskaug.

MVC became a popular architectural pattern used in several language systems. It was also the source of some patenting activities that led us to regret our decision not to write the fourth book in the Smalltalk-80 series, in order to have had a clearer record of our inventions and so block some of the patenting that has since taken place! No definitive decision was made to skip the fourth Smalltalk-80 book, but somehow time and effort were diverted to other activities. Two early exemplar kits embodying the MVC concept were the LRG Simulation Kit that I wrote in 1977–78 in support of a two-day course delivered at PARC for Xerox's top executives, and LearningWorks, a special interface to the Smalltalk-80 system, initially implemented at ParcPlace Systems by Steve Abell, and then completed at Neometron by David Leibs, Tami Lee, Wladimir Kubalski, and myself. It was successfully deployed by the

Open University in the UK as part of a new approach to teaching computer science in a course designed by a team led by Mark Woodman.

The LRG Simulation Kit was a framework for defining discrete event-driven simulations, modeled on the Simula 67 Demos [2] simulation package created by Graham Birtwistle. The Simulation Kit was created for a seminar attended by Xerox's top management. The participants, including the Xerox President, defined the queuing algorithms, copier/duplicator work times, and business center copying demand, as well as modified the object images to portray work versus idle times. The animated view of copier-duplicators queuing up for service in different kinds of customer stations was a playful approach to educating Xerox executives on the ways software can be used to modify hardware behavior, a powerful idea from the 1970s that we now take for granted. What made this project personally significant was Alan's arranging for me to take an Alto home for work (I was the first to do so) after the birth of my second daughter. I never did learn whether he was breaking Xerox rules again!

LearningWorks explored new ways to organize and browse object libraries so that the so-called dictionary of words analogy we used earlier could be better controlled—essentially enabling us to dole out objects in “learning books” according to a curriculum, and to build up the learner's available library by extending a book with student-created objects or by adding new books. The Open University curriculum devised for LearningWorks encouraged students to explore how differences in a library affect how they build a software application. LearningWorks also provided a more context-sensitive debugging tool so that students could see how the objects they already learned about were interacting, while avoiding a deep dive into a system library of objects with which they were unfamiliar.

Learning from the ParcPlace Systems customers was always an important benefit of having started the company. Business modeling—actually modeling in general—begs for concrete examples to stimulate thinking about new system designs. There are flashes of genius, but most creative design is incremental or based upon an analogy. So we are back to the importance of the idea of a kit.

Business modeling is at once straightforward and complex. The straightforward part is mapping the static representations of business information into object hierarchies, with an eye to reuse, and mapping the dynamics of business processing into object interactions. Complexity comes when modeling an unknown or new way of doing business, often one that involves people who previously did not have direct access to computer-based capabilities. It is much easier to see something someone else did and be able to say, “I want something like that except...” That’s why real estate developers put up fully decorated and furnished home models! And that is why prototyping systems as a part of the applications engineering process became so popular.

A Brief Thank You

Alan continually drew his colleagues into learning about the creative process and identifying how looking at concrete examples triggers understanding and ideas. His passionate quest has always been the support of individual and group creativity. As we look for new ways to enable anyone to be creative by telling a computer what to do, we are building on the successes of the LRG research and its “life after PARC.” In 1973, when I joined LRG, Alan started me on my own remarkable journey, for which I am forever in his debt.

On my journey, I have had the good fortune to take the nascent ideas we worked on for K–12 education and explore their applicability for professional programmers, online K–12 math and science education, online collaboration portals for research scientists, and support for remote project teams, particularly teams developing pharmaceuticals. Of the many ideas that grew out of the LRG experience, the concepts of objects as components, programming as modeling, and kits of extensible components stand out. That these things see broad applicability today is a testimony to their enduring power.

Adele Goldberg holds a Ph.D. in Information Science from the University of Chicago, for work jointly conducted while a research associate at the Stanford University Institute for Mathematical Studies in the Social Sciences.

She began working with Alan at Xerox PARC in July 1973. In 1987 Adele, Alan and Dan Ingalls received the ACM Systems Software Award for the development of Smalltalk. Adele left Xerox in 1988 to become the founding Chairman and CEO of ParcPlace Systems, Inc., where her work with Smalltalk continued. From 2002 to 2006 she was CTO of AgileMind, Inc., working to enhance equity and high achievement in secondary school math and science.

Adele is currently a consultant, focusing on the use of virtual communities to support more effective teamwork. She is involved in projects to design new computer technologies supporting drug development and to enhance collaboration among researchers in neuroscience and mental health.

Bert Sutherland

Manager as Pupil

Most of my working career has involved participating in and managing computer research operations at the MIT Lincoln Laboratory, at Bolt, Beranek and Newman, Inc., at the Xerox Palo Alto Research Center, and at Sun Microsystems Laboratory. Alan Kay was a fascinating outlier in the wide variety of talents and personalities I had as colleagues.

I learned a lot from my five years as Alan Kay's nominal manager at the Xerox PARC Systems Science Laboratory. I had met Alan several years before in the early 1970s at an ARPA IPTO Principal Investigators (PI) meeting in Alta, Utah. Those stimulating meetings were called the ARPA Games by the attendees, and the PIs had many interesting discussions. I was employed at Bolt, Beranek and Newman, Inc., at that time. Alan, then a graduate student at the University of Utah, had come up to the Alta ski resort as one of the Utah graduate student attendees. A blizzard closed the road down to Salt Lake City and Alan was stuck overnight in Alta. He spent the night in the spare bed in my room. Little did I know then what was in store for Alan and me in later endeavors.

Several years later I had changed employment to Xerox PARC as manager of the Systems Science Laboratory (SSL). Alan was already there as leader of SSL's Learning Research Group with a very interesting group of diverse talents including Adele Goldberg, Dan Ingalls and Larry Tesler, to mention

only a few. Alan knew my brother Ivan Sutherland from his time at Utah, and I suspect that Alan had a hand in recruiting me to Xerox PARC SSL. Alan and his group were all busy developing, improving, and using Alan's and Dan's Smalltalk system for a variety of experimental applications. This group had placed several of PARC's Alto personal computers in the Jordan Junior High School in Palo Alto for the students there to use for programming in Smalltalk. Alan and Adele were exploring how to make it easier for ordinary people to use the power of computers, and the youngsters were relatively uncontaminated raw material for their observations about learning. These youngsters also came up to PARC where their energy and enthusiasm were very noticeable. Later at Sun Labs I always tried to have several high school interns with some of the same enthusiasm of youth to enliven the atmosphere.

Alan traveled a lot while at PARC with speaking engagements all over the country. He was very casual about completing his trip expense reports in a timely manner to get Xerox reimbursement of the expenses. This was much in keeping with the relaxed fiscal accounting attitudes at Xerox PARC. Dr. George Pake, the center director, had decided not to bother the researchers much with the grubby details of accounting that might impede their creativity. We did have annual budgets but did not track expenditures very closely. The capital budget was considered mainly a bureaucratic hurdle to be overcome, and as SSL manager I never saw the resulting depreciation charges. During some big budget squeezes imposed on PARC after I left, I understand that the large body of overhanging depreciation charges were very painful to absorb. Coping with Alan's delayed expense reports was a continuing hassle for me in keeping an accurate track of my money resources. I never really knew how much Alan had already spent on his trips until he turned in his expense reports. One December in frustration I was particularly draconian, telling Alan, "All your reports in by December 31! I will not approve any last-year's reports in January using the current year's money for last year's expenses. December 31!—or any unreported expenses turn into your personal obligations!" Alan did get his reports in on time that year. And the extended lesson for me was to insist at Sun

Labs that the technical managers be more aware of their budgets and manage expenses with more attention than was the practice at Xerox PARC. Modest fiscal awareness can be achieved without sacrificing technical creativity. Alan helped me understand better the difference between leading and managing a group and the benefits of free spirited research and how to trade that off against the impact of R&D expense on corporate profits. It emphasized to me the role of being a buffer between the sponsor's necessary fiscal requirements and the value of freedom from fiscal worries that can disrupt real creativity.

I was recently at the Computer History Museum in Mountain View, California, and saw again in the exhibits the Notetaker portable computer that Doug Fairbairn in SSL made for Alan. While the Alto was envisioned as the "interim Dynabook" of Alan's dreams, Notetaker was Alan's subsequent push on portability in contrast to the Alto's advances in functionality. Notetaker was one of the early "luggable" computers, if not the first, copied shortly afterwards commercially. Alan's constraints were that it had to run Smalltalk on battery power and fit under an airline seat. He made a triumphant return from his first road trip with his new computing suitcase. Unfortunately, it was really too cumbersome to be useful in practice and we did not make a large number. However, I recall it was interesting that it ran only Smalltalk as both its application and operating system software. No other operating system! Alan's crew of software wizards were outstandingly ingenious. The lesson I carried forward was that the singular Notetaker without a large user community to explore its real utility was of much less value than the more ubiquitous Alto experimental computer with an interestingly large user population. At Sun Labs when Duane Northcutt's group created the prototype of the SunRay, now a Sun product, they came to me to fund a build of a hundred units for experimental deployment in the company. I said, "*No* to only a hundred, *yes* for a build of two hundred!" It turned out that two hundred prototypes were not really enough either. Alan's and PARC's focus on wide user experience is worth remembering.

Later, around 1995 and quite unexpectedly, my earlier exposure to and experience as a user with object-oriented Smalltalk was extraordinarily useful

to me personally when as the Director of Sun Labs I was tasked by Sun to be the interim manager of the small group developing Java. Alan's tutorials for me about Smalltalk came back vividly. It really felt like *déjà-vu* to have nominal oversight of a rambunctious bunch of programmers developing a new object-oriented language. I was able to listen in once again on all the arguments and opinionated tussles required to compromise strongly-held basic beliefs on what is good software design. Such wonderfully creative people cannot be managed in any conventional sense; they can only be encouraged, shielded, protected, and even occasionally persuaded to align their activities with the sponsor's needs in order to create a result of real utility to the sponsor and society. I found it very useful that I was not a complete novice in the object-oriented software world that was being refined with the Green project, as it was then called. Later when it came time to pick the new name for the language, the team was split between Silk and Java. At the meeting to decide, Eric Schmidt told me to choose since I was nominally in charge. I chose Java on the advice of the lab lawyer that there would be fewer trademark problems in the future. But I think it was fortunate to have been a name that has followed the prior memorable examples of Sketchpad and Dynabook.

Let me summarize some of what I learned from Alan with some generalities. Alan was at the extreme edge of the creative people I have known working in a corporate-sponsored research environment. Alan's vision was bigger than one person could achieve alone and required additional assistance. Alan's nature was not conducive to managing all the overhead detail that a large project entails. He was comfortably the leader but not the manager of his group, and there is a big difference in the activities required. His group of about a dozen people worked very well, and he had his friend Chris Jeffers in the group with a personality to handle some of the necessary details. I learned a lot from the two of them about how I should act to provide the resources the group needed and to protect them from the bureaucracy that a large corporation entails. This was very good training for me in my later role as Sun Labs Director. Corporately-sponsored research is a direct hit to company profits, and the research managers

have an obligation to see that the investment in research returns value to the sponsor. Unfortunately, we computer research managers at Xerox PARC, along with Xerox senior management, “fumbled the future” and saw much of the leading technology developed appear commercially at Apple, Adobe, 3Com, ParcPlace Systems, and other Silicon Valley organizations. I really learned a lot from Alan and my other colleagues, and I was able to apply some of this painfully-acquired wisdom during my time at Sun Labs.

So I remember Alan Kay as a free spirit with supreme self-confidence in his vision of what the world’s future could be. He persisted with his dedicated devotion to see the dream come to fruition. Like other visionaries he was always impatient with distractions that consume his limited time and impede his journey toward his future vision. Alan and my brother Ivan Sutherland have a lot in common with their continuing focus on their vision of the next interesting technical application. “Damn the torpedoes, full speed ahead!”

We are all now reaping the fruits of Alan’s vision with widely-available, contemporary and portable “communicating Dynabooks” in worldwide daily use as a ubiquitous part of modern life. PDAs and cellular phones are being carried all over the world with electronic data exchange capabilities. Alan called his vision a “Search for the Holy Grail” that is now bearing fruit as technology has advanced over the nearly fifty years since his early mock-up of Dynabook. Back then his vision was embodied in an 8½ by 11 inch aluminum picture frame, with cardboard inserts indicating display and input means, to illustrate his Dynabook concept.

In spite of all the progress to date, I do not think the search Alan envisioned is over. There is much remaining to discover before the inspirations in Alan’s Dynabook, Doug Engelbart’s Augmentation work, and Vannevar Bush’s Memex are fully realized in truly useful and natural computer aides. Hopefully Alan’s example will inspire a following generation of visionaries to continue his quest and lead the way onward. They will indeed have very big shoes to fill!

William R. (Bert) Sutherland received his Bachelor's degree in Electrical Engineering from Rensselaer Polytechnic Institute, and his Master's Degree and Ph.D. from MIT.

He was the long-time manager of three prominent research labs: the Computer Science Division of Bolt, Beranek and Newman, Inc., which helped develop the ARPANET, the Systems Science Laboratory at Xerox PARC (1975–1981), and Sun Microsystems Laboratories (1992–1998). In these roles he participated in the creation of the personal computer, the technology of advanced microprocessors, three-dimensional computer graphics, the Java programming language and the Internet. Unlike traditional corporate research managers, Sutherland added individuals from fields such as psychology, cognitive science, and anthropology to enhance the work of his technology staff. He also directed his scientists to take their research, like the Xerox Alto “personal” computer, outside of the lab to allow people to use it in a corporate setting and to observe their interaction with it.

Bob Stein

Do it

I'm not a computer scientist or a programmer. I take it on faith that Alan's contributions to the field are significant but I'm certainly not in a position to comment on them. What I do know about is Alan's remarkable generosity and his ability to change people's lives by believing in them and giving them a chance.

In the fall of 1980 Charles Van Doren, the editorial director of Encyclopedia Britannica, hired me to write a white paper entitled *Encyclopedia Britannica and Intellectual Tools of the Future*. The paper included the observation that Britannica's future could well lie in a joint venture with Lucasfilm and Xerox. I spent a year writing it, and a few months later Alan left Xerox PARC to become Atari's chief scientist. Knowing Alan only by reputation but dreaming of a potential partnership, I screwed up my courage and called. Alan read the paper—all hundred and twenty pages of it—while I sat in his Silicon Valley office. “This is exactly the sort of project I'm interested in,” he said, “Come here and work with me.” So I did.

For the next eighteen months, from late 1982 to early 1984, Alan and I worked on a project we called The Intelligent Encyclopedia. It was a romantic vision of an online resource that could answer most any question to which the answer was known and produce answers customized to the level of the questioner. We assumed a top-down editorial hierarchy similar to the way that print

encyclopedias were developed. The opportunity to use the Internet to harness the wisdom of the crowd (Wikipedia) didn't make it onto our conceptual horizon. We did presume, however, that access to the Intelligent Encyclopedia would be ubiquitous, so people could ask a far wider range of questions than those contained in thirty print volumes. In an experiment that eerily predicted the search engines of today, we actually handed out tape recorders and asked a number of people to try to be conscious of all the questions that occurred to them during the course of a day, in the hope of discovering what sorts of information people might seek out if they were confident of getting the answers.

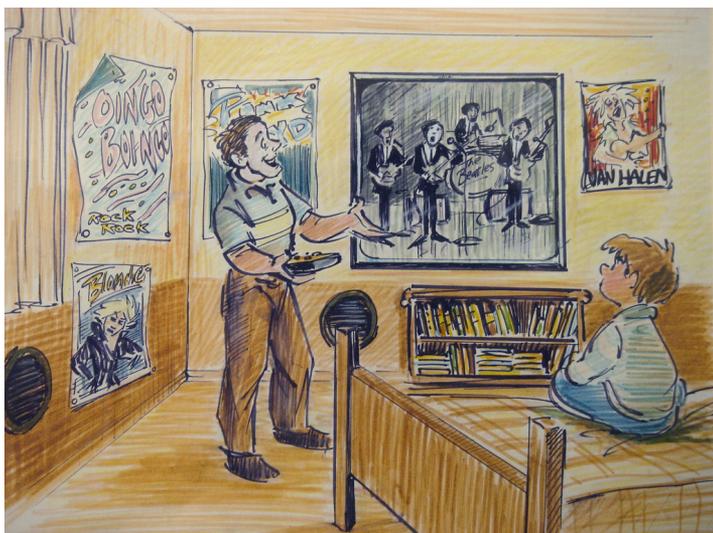
At the time Atari was owned by Warner Communications. Periodically we would put on dog-and-pony shows for the Warner executives to give them a sense of what Alan's group was doing. For one such meeting Alan and I decided to portray the IE project via a series of vignettes drawn by Alan's friend Glen Keane, a well-known Disney animator. The captions beneath the following images are the actual instructions we gave to Glen.



A business man on an airplane reviews stock market trends.



A third grade class studies various aspects of space travel. The group on the right is running a simulation of a Mars landing, while the students on the left are studying solar sails.



A father reminisces with his son about 1960s rock & roll, calling up footage on the IE from The Beatles' appearance on The Ed Sullivan Show.



A vintner in northern California wonders what would be involved in shifting from wine production to sake. On horseback, he is asking the IE about soil and water requirements for growing rice.



An architect in New York studies Japanese design for a project he's working on, while a teacher in Tokyo talks with her class about architecture in New York.



Children in the dinosaur exhibit in the Museum of Natural History walk around with IE terminals instead of audiotape players. Interactive simulations of dinosaur life from the IE are running on the wall monitors.



An earthquake strikes in the middle of the night. The IE, connected to an online service, informs the couple of the severity of the earthquake and makes safety tips readily available.



In a bar, the two men at the right are watching football on the screen and running what-if simulations, which second-guess the quarterback, on the countertop IE. The couple on the left is taking one of the IE's courses in wine connoisseurship.



A mother and her children, looking into a tidepool in Laguna, ask the IE about the plants and animals that they see. (Notice the antenna for wireless communication.)

About a year and a half after we started, the Intelligent Encyclopedia project died along with most everything else at Atari, when millions of unsold PacMan games were plowed into the earth and the Warner overseers lost confidence in the company. Working with Alan over those eighteen months, however, set the basis for my later work founding both The Criterion Collection and The Voyager Company, and all my work since.

Early on in my time with Alan at Atari, I'd written a ten-page memo detailing our internal and external, long-term plan of attack for developing the Intelligent Encyclopedia. Alan scrawled the entirety of his response on the cover: "Do it." I can think of no better example of Alan's deep, abiding, and generous confidence in people—in this case a thirty-four-year-old guy with no scientific or academic credentials whatsoever. My experience with Alan shaped my life in countless ways for which I remain profoundly grateful.

Addendum: A note from Glen Keane

A twinkle in their eyes

In the 1970s Disney Animation went through a transformation as Walt Disney's master animators, known as the "Nine Old Men," began to pass on to young artists the secrets of their art form. I was privileged to learn from these quiet, creative geniuses. It was a master-apprentice relationship as each day Frank Thomas and Ollie Johnston—like grandfathers in cardigan sweaters—patiently passed on the principles of Disney Animation, always with a youthful twinkle in their eyes.

In 1982 Disney Animation went on strike and I had an opportunity to do some freelance illustration for Atari R&D in Sunnyvale, CA.

I was about to meet a genius of another kind.

I flew up to meet a man named Alan Kay, sat down in his secretary's office and waited for him. As it turned out I spent the day catching only glimpses of this whirlwind of creative energy and ideas packed into a muscular frame with intense eyes and a Cheshire cat grin. Finally towards 5 p.m. my moment arrived, however not as I expected it.

Alan needed to leave immediately to catch his plane. He suggested I hop into his car. We could talk on the way. Alan drove and talked, I listened. Concepts rushed out of him like pressurized water from a broken fire hydrant. I struggled with all my might to concentrate on what he was describing to me. He spoke of his vision for a network of information never before imagined and a world of notebook-sized computers. He was asking me to illustrate these ideas so he could better communicate it to others. Arriving at the airport he hustled to his gate as I jogged alongside continuing to download. As he disappeared onto the plane I was left with my head reeling. What had I gotten myself into?

But there was that “something familiar” in Alan’s eyes. I realized it was that same youthful twinkle I had seen in my mentors’ eyes. There was joy and passion. It was irresistible.

I continued to illustrate his concepts as Alan moved to Apple and developed the Vivarium project. During that time I resumed my work for Disney creating and animating *The Little Mermaid*, The Beast from *Beauty And The Beast*, Aladdin, Pocahontas and Tarzan. Most recently I have designed the character of Rapunzel for Disney’s first computer-animated fairytale.

To be challenged creatively is every artist’s goal. Nowhere have I felt it more than working with Alan Kay. He does have that “twinkle.”

Glen Keane studied Experimental Animation (then called Film Graphics) at the CalArts (California Institute of the Arts) School of Art. He graduated in 1974, joining Disney the same year.

At Disney he rose to Lead Character Animator, becoming one of the group sometimes referred to as the “Nine New Men.” Glen received the 1992 Annie Award for character animation and the 2007 Winsor McCay Award for lifetime contribution to the field of animation.

In 2003 he began work as the director of Disney’s CGI animated film Rapunzel scheduled for release in 2010. Glen and his team hope to bring the unique style and warmth of traditional cel animation to computer animation.

Bob Stein was the founder of The Voyager Company where, over a thirteen-year period, he led the development of over three hundred titles in The Criterion Collection, a series of definitive films, more than seventy-five CD-ROM titles including the CD Companion to Beethoven's Ninth Symphony (which Alan paid the supreme compliment of calling "the first CD-ROM good enough to bother criticizing"), Who Built America?, Marvin Minsky's The Society of Mind and, in 1992, the viable electronic books including Douglas Adams' Hitchhikers' Guide to the Galaxy.

Bob is currently the Director of The Institute for the Future of the Book, a little think-and-do-tank exploring, and hopefully influencing, the evolution of new forms of discourse as it moves from printed pages to networked screens.

Leonard Kleinrock

About an Ageless Alan Kay

Alan Kay is a complex, creative and brilliant researcher who neither I, nor most anyone else, can claim to totally appreciate. But those of us who know Alan recognize his brilliance. He sees the world in his unique fashion which allows him to cross boundaries and break new ground repeatedly. I have had the pleasure of knowing Alan for more than four decades in a variety of relationships and I am happy to offer my remembrances and insights into this very special person on the occasion of his 70th birthday.

The first time I met Alan was during those exhilarating early days when I was a young UCLA faculty member. We were planning, designing, implementing and deploying the ARPANET (circa 1968–69). Alan was one of the heroic group members we (the ARPA Principal Investigators) had empowered to design and implement the protocols and software for the emerging network. That group was a self-formed, loosely-configured band of maverick graduate students from a number of universities who took on the challenge we had ceded to them. Alan was working on graphics at the University of Utah at the time. One day, he walked into my UCLA office to meet me and to chat about the ARPANET project. I was totally impressed with him and spent quite some time engaging him and exploring so many issues of interest at the time. I could see how well his natural approach to solving problems and developing systems was matched to the vision of the connected world of users in the Internet and

the style we, as PIs, encouraged of our colleagues. In other words, we hit it off. We have remained connected ever since.

I watched Alan blossom over the next few years (1970s) at PARC with his truly innovative work on the Dynabook, on Smalltalk, the Internet and much more. I continued to work on the grand challenges of the Internet as a UCLA faculty member as the Internet grew in an unbounded fashion. Alan and I interacted occasionally during that period, but less as he moved to Atari, to Apple, and then to Disney. It was not until the late 1990s that we once again reengaged in an exciting and closely coupled fashion. The background is as follows.

In 1976, my wife and I started a company called Technology Transfer Institute (TTI) to put on a three-day technical seminar series based on my newly published book on packet switching. The seminar was an instant success and so I formalized the company and invited my professional colleagues to develop and conduct other related seminars. TTI continued to grow as we put on many many technical seminars over the years, and even ran a major trade show and conference along the way.

Concurrently, Alan and Nicholas Negroponte created a conference series called Vanguard under the auspices of Computer Sciences Corporation. Vanguard was based on a member-supported model and focused on emerging technologies two to five years in the future. To hear them talk about it, Alan and Nicholas say they formed Vanguard in order to create an Advisory Board consisting of people they respected and with whom they wanted to have engaging dinner conversations on a regular basis! What a beautiful idea! And so they did, and Vanguard was launched in 1991. It was a successful conference series with terrific speakers, talented members, and a stellar Advisory Board. I remember one day in June 1995 at a Marconi Fellows meeting in Bologna, Italy, when Bob Lucky described to me this fabulous group of which he was a part (he was one of those Advisory Board members with whom Alan and Nicholas wanted to dine!). Recognizing that TTI was an experienced and successful seminar company, and that I totally understood the needs and the culture

of Vanguard, Bob approached me on behalf of Vanguard asking if I would be interested in joining their Board and having TTI become the managing entity for them. This was a marriage made in heaven and so, in 1998, the two companies merged and TTI/Vanguard was born.

Once again, Alan and I were closely engaged in a shared passion: thinking about the future. To these conferences we invite articulate and innovative speakers who are working at the forefront of their field. Alan does not hesitate to question, challenge, critique, support or embellish a speaker's presentation. His comments often expose a better understanding of the fundamental issues being discussed than those coming from the speaker! As most people know, Alan does not suffer fools well, and the form of these conferences brings out some sharp and insightful comments from Alan, often shining light on very complex issues in a most informative fashion. On other occasions, Alan will sit quietly for most of a presentation, then burst into a penetrating, surgical and relentless critique of the material and the conclusions. He is unyielding on points in which he believes (and he is usually correct) and takes the conversation to a whole new level of understanding and inquiry, drawing analogies and parallels from so many different fields of science in which he has expert knowledge (biology, learning, music, etc.).

I would have loved to have been Alan's teacher in his younger years and in his graduate work; on the other hand, he would have been a real challenge! One does not teach Alan, one learns with Alan. Even now, I know that it's best to be prepared to question all prior knowledge and so-called "self-evident truths" when working with Alan. In that, he is ageless, and has the same level of excitement, enthusiasm and mental leaps that he had when I first met him.

Alan makes clear that society has developed certain skills over millennia, and these skills are not ones that can be re-invented by children without help. He makes the point that language is natural to humans, but writing is not and must be taught; that playing chopsticks on a piano is where a tinkering child will take us, but to play Beethoven, one needs to be instructed. Kids are great

at inventing, but need to be exposed to what he calls “powerful ideas” to really get moving.

His disdain for top-down rigid structures for system design matches my own approach of investigating elements of a system a few pieces at a time, studying their properties and potential, understanding how they work and how they allow for interaction, and then allowing them to interact, grow, build, and eventually expose the fabric of a larger system solution that is natural to their properties. In many ways, this can be called a “distributed” approach—one with no central control, but with the elements themselves sharing that control. The advantages of such an approach are manifold, yielding systems that are robust, self-directing, scalable, etc.

Thanks Alan, for all you have done for me, for the field, for the world, and most importantly for those young minds yearning to learn and create and invent and enjoy this golden age in which we are privileged to find ourselves.

Leonard Kleinrock received his Ph.D. from MIT in 1963, where he developed the mathematical theory of packet networks—the technology underpinning the Internet—nearly a decade before the birth of the Internet. That birth occurred in Len’s laboratory at UCLA which hosted the first node of the Internet in September 1969.

He is a member of the National Academy of Engineering and of the American Academy of Arts & Sciences, and is a fellow of several organizations including the IEEE, the International Electrotechnical Commission and the Guggenheim Foundation. He was awarded the National Medal of Science in 2007.

Len is currently Distinguished Professor of Computer Science at the University of California, where he served as Chairman of the department from 1991–95, and continues to serve as Chairman of TTI/Vanguard. He also serves on the board of directors of Viewpoints Research Institute.

John Sculley

Genius is Seeing the Obvious Twenty Years Ahead of Everyone Else

In 1986, Apple had regained its footing with revenues and profits on the upswing. We now were successfully marketing the evolution of Steve Jobs' Macintosh Office as the Macintosh Desktop Publishing system composed of the new Macintosh 512k (Fat Mac), PageMaker, Adobe's PostScript page description language, and our LaserWriter 2.0 printer. Just when I thought we were out of the woods, Alan Kay came to me and said, "Next time we won't have Xerox," meaning with Steve Jobs gone along with Steve's talent to recognize brilliant technology innovation and convert it into insanely great products, who was going to create Apple's vision going forward? Steve and I had worked closely enough together that I appreciated his methodology of creating end-to-end systems always beginning and ending with the user experience. But I was not qualified to be Apple's next product visionary.

Alan Kay became my mentor, or as he liked to phrase it, like my Oxford Don whose responsibility it was to guide his students to all the good stuff. Alan told me Steve's genius was seeing what would become obvious to the rest of us twenty years later. I asked Alan whether there was any other way one could predict where technology might take us other than pure genius insight.

Alan told me that every innovative technology, no matter how simple or how complex, always takes about fifteen to twenty years to evolve from concept to a commercial ready state. If this were true, then many of the technologies that would be important to Apple's future were already in some stage of percolating their way through this evolving process. Thus began a year of our visiting research laboratories, technical universities and many discussions between Alan, me and various Apple engineers where we tried to map out what might seem obvious to everybody in twenty years.

Previously, Alan had foreseen a future where an individual should be able to create simulations via interactive visual models on a computer screen. This was his genius insight that he conceptualized with Dynabook and Smalltalk, the first graphics-based programming language back in the early 1970s. Alan's innovations set the direction for personal computing as we know it today.

The best innovators are really at heart end-to-end systems designers. This perspective helps explain why several Asian consumer electronics firms have had so many missteps. They tend to focus on technology component invention, treating a product development effort as a set of discrete and very detailed tasks.

In contrast, Alan Kay is an elegant systems designer seeing the most interesting problems to be solved as systemic challenges that have the potential to change how we fundamentally think about things. To use an often quoted Alan aphorism: Point of view is worth eighty IQ points.

The culmination of Alan and my year's investigation together was conceptualized in 1987 in what we called the Knowledge Navigator. While Moore's Law had already predicted that processing power in the next twenty years would be able to manipulate three-dimensional geometries in real time, the Knowledge Navigator envisioned a world of interactive multimedia communications where computation became just a commodity enabler and knowledge applications would be accessed by smart agents working over networks connected to massive amounts of digitized information.

In 1987, Apple also invested in a Cray XMP 48 super computer which enabled our engineers to experiment with what real time manipulation of

multidimensional objects on a screen would look and feel like many years before such computational power would be available on general purpose personal computers.

I was intrigued by Alan's certainty that the Knowledge Navigator was not a far-fetched idea. We asked: couldn't we use Hollywood special effects animation to simulate what the experience of the Knowledge Navigator would be like long before it was possible to build anything like it?

Alan's wife Bonnie MacBird was the screenwriter on Disney's original *Tron* motion picture and we engaged her along with the Apple creative team of Hugh Dubberly and Doris Mitch to create a video simulation which would capture the experience of a professor at Berkeley using the Knowledge Navigator in the year 2009. To me, such an approach was not much different from the techniques we had used when we produced Pepsi Generation or the 1984 Macintosh TV commercials. In marketing, perception always leads reality.

Knowledge Navigator was never intended to be a real product. In fact, it was a pretty controversial project with Apple's engineering community. Stanford University's engineering school even hosted a symposium where opposing sides debated whether the future of computing metaphorically should take the form of anthropomorphic agents or avatars as we showed in our Knowledge Navigator video; or should computing be more like a prosthesis as used by Sigourney Weaver in the film *Aliens*?

Alan saw the Knowledge Navigator as a credible vision of what would be obvious to the rest of us in the future: knowledge-rich collaboration and communication. I saw the Knowledge Navigator as a way to keep Apple in front of the world as a culture rich with innovation and creativity.

Alan and I took the Knowledge Navigator video everywhere. We got it on Soviet television as the focal point for a discussion between Russian scientists Andre Sakarov, Yevgyni Velikov and Rol Sagdiev about what the future for Russians would be like after the Berlin wall came down; Perestroika began and information between Russian citizens could flow freely. Knowledge Navigator was the subject of a cover of Fortune magazine, many technology publications

around the world, television programs, and of world-class university and high-profile technology industry events.

Years later, I was having breakfast with Jonas Salk and John Perry Barlow. Dr. Salk, who unfortunately died shortly after we were together, said the world would soon enter into an evolutionary era far grander than anything Charles Darwin had investigated. This new era would comprise an accelerated evolution of our human species set off through extraordinary access to knowledge over immense networks with computers interacting directly with computers while enhancing the interactions between humans collaborating with humans. He called this the coming age of Wisdom. Others refer to this phenomenon as swarm theory intelligence, where a beehive becomes smarter than any individual bee.

Today, the enthusiasm and self-confident assurance of the future ahead from visionaries like Steve Jobs, Alan Kay and Jonas Salk seems obvious.

I recall an afternoon when Steve Jobs and I went to visit Dr. Land, founder of Polaroid, who had been pushed out of the company he founded and had moved to a laboratory on the Charles River in Cambridge. As we sat together around a large conference table Dr. Land remarked that great products like his Polaroid instant camera aren't really invented by any of us; they've always existed, right there in front of us, invisible—just waiting to be discovered.

Steve Jobs immediately connected with Dr. Land's observation, saying the reason he never did consumer research when he built a new product is he trusted his own instincts more than others who couldn't see what he saw. Steve agreed with Dr. Land's point of view saying he felt that the Mac too had always existed; invisible to the rest of us, just waiting for Steve to come along and reveal it.

Towards my last days at Apple, I related this story to Alan Kay and he broke into a broad smile and said, "Of course that's the way it is."

Fast-forward to twenty years later.

The following is an e-mail to me from my son Jack Sculley, now a physicist and an environmental scientist, whom I had asked to read a draft of this chapter.

Our original Knowledge Navigator video had a visual simulation showing a correlation between extensive tree clearing in the Amazon rain forest and a predicted expansion of the Sahara desert. Twenty years ago, Alan Kay was confident that simulations like this would become obvious in the future.

Hi Dad,

Thanks for the chapter—I’ve always thought this was one of the coolest projects you worked on! Interestingly enough, I just had an experience with a Mac that in broad outlines matches Alan Kay’s predictions for a Berkeley professor in 2009.

Inez Fung, a preeminent atmospheric physicist who is one of my mentors at Berkeley, and I were trying to tackle the problem of what climate change would do to food webs at the nexus of rivers and oceans. We pulled up a NASA agent called Giovanni and requested a time series of data from a satellite called “SeaWifs” for the California coast from Cape Mendocino up to the Eel River mouth. Formerly this would have involved lines and lines of code to FTP ASCII files from a server and read them into a visualization program. Here we just drew a box over the area of interest, clicked the start and end times, and the Giovanni agent did the rest. While we didn’t have voice interaction, it wasn’t necessary, we had the graphical tools to quickly select the data and Giovanni had the AI to get us a presentation quality chart in about 5 seconds. Then I downloaded the data onto my MacBook Pro and pulled up a MATLAB program I had written collaboratively with people at MIT that simulates a marine plankton food web and compared the actual satellite observations with what a planktonic web would do if it were supplied nutrients from upwelling vs. river discharge to see which model matched the data. As it turns out, they both do, with blooms in summer showing an upwelling signature and blooms in winter showing a river signature. Not the Amazon rainforest in your film but planktonic food webs are probably even more important as planetary “lungs.”

As Inez pointed out, to do this in her office at MIT 20 years ago would have taken months of arduous coding. At Berkeley in 2009 it took us ten minutes. My next step is to simulate what will happen to upwelling

and river discharge under different climate change scenarios. We hope to publish the results in a scientific journal next year once we cross-check the data with my submarine cores.

Thought you would be pleased to see your and Alan's predictions come true almost to the letter and day! The chapter reads very well and I hope you expand it with more details of your fascinating interactions with Alan, Dr. Land, Sagdiev, Salk and Barlow.

John Sculley was CEO of Apple Computer from 1983 to 1993.

Since Apple he has been involved with a number of private companies. Each of these has involved technology-enabled, end-to-end platform services, in industries poised for major transformation. John also mentors serial entrepreneur CEOs in areas including home energy management, home testing for sleep apnea, and post-secondary school education.

John says, "Alan's insight that 'POV is worth eighty IQ points' is why I still do what I do."

Bobby Blatt

The Vivarium—a place to learn about learning, and to think about thinking

Out of the blue in 1984, a phone call came to my office at The Open School: Center for Individualization, the magnet elementary school in Los Angeles that I headed as principal. The caller's voice was one I did not recognize but would later know well. The caller was Alan Kay.

The Open School had been in existence for seven years—founded in 1977 in response to parents' expressed need for a non-traditional, open educational environment—when Alan had somehow heard of our innovative approach to public education and called me to arrange a site visit. At the time I didn't fully comprehend Alan's intent, but quickly it became evident to me that joining forces with Alan and Apple's Advanced Technology Group would be an adventure that would profoundly change all our lives. It was called The Vivarium Project.

Alan described the project as: "A long-range research project using children's interest in the forms and behavior of living things to inspire a wide variety of exploratory designs in curriculum, user computer inputs and outputs, and modeling of behavior." We didn't quite know what that meant or what it would look like. I was cautious at first and very concerned. What would this do to the culture of the school? How would this program impact our

belief system about how students learn? Would the technology overshadow the human interactions and relationships that were core to our pedagogy?

Alan at first wanted to work with just 4th and 5th graders. But I knew if we were to be successful and have full buy-in from faculty and parents, it had to be a total team effort. Alan saw the value of this immediately and agreed to a whole school involvement.

Alan said that he selected the Open School because of our emphasis on inquiry, exploration and discovery. He often compared us to a good graduate school, “with its use of organizing projects, highly-literate reports, and considerable freedom in pathways to goals. The Open School was the perfect place for the Vivarium exploration.” Alan was often heard saying, “We didn’t come here because the school needed fixing. We came because we share a common vision and passion about education and learning.” We were very flattered and impressed but still nervous and not quite ready to jump in. Alan, the perfect teacher, guided us slowly and initiated weekly brown-bag lunches to introduce us to projects, answer our questions and begin the dialogues that were to become the signature of how we worked together for the next eight years. We engaged in the important, reflective conversations about the role of computers in education and the definition of education itself. These conversations expanded, and once a year we moved off campus for a two- or three-day retreat called a “learning lab” where we exchanged ideas with experts from various fields in music, art, science, media and literature to “learn about learning and think about thinking.”

The students, teachers and staff of The Open School became full participants in The Vivarium Project in 1985. Alan and his colleague, Ann Marion, brought in an initial cast of researchers, programmers, curriculum consultants, and artists to spend part-time hours observing in their classrooms, and assisting the students and their teachers with the Mac Plus computers and LaserWriters. Alan thought computers should be considered unremarkable, a tool to have handy when you needed it. Thus, the debate between “computer lab” and “computers in the classroom” ensued. Computers in the classroom won hands

down. A small space was also set up for group lessons about VideoWorks (the predecessor to MacroMedia's suite of products) that included exemplary animations drawn in VideoWorks by Vivarium advisor Frank Thomas, the late master Disney animator. Teachers were given computers to have at home to help accelerate their investigation of the use and usability of these new machines. The project hired a liaison between The Open School and Apple personnel to be on campus full time, and a videographer who became a fixture at the school as she unobtrusively documented key aspects of the Project. Another consultant who started out as multimedia designer ended up as part of the faculty in charge of the Life Lab garden and Apple Global Education telecommunication projects.

The school district moved the site of The Open School to Airdrome Street in 1987. The six bungalows of the new school site were rewired to accommodate intranets in each classroom. Networked Mac SE computers for every two students, a LaserWriter, and Mac II file servers were installed in each cluster. Not only computers but also CCD video camcorders, laserdisc players and sound equipment were given to us by Apple. All this equipment raised many concerns. It was impacting the physical space and the instructional climate. The classroom looked like a hi-tech factory. We wanted the technology to be hidden and Alan came up with the solution. He sat with the teachers and together they designed a special "MacDesk" with a Plexiglas top and recessed computer. Now students had instant access to the computers when needed, but could use the desktop for their other school tasks. The computers were invisible.

A room adjacent to the school office was remodeled to be a learning lab for the teachers, where teachers could share their ideas and present their projects to one another for critique and discussion. A section of asphalt covering the playground was removed and fertile soil was trucked in to create a Life Lab garden on site. In this enriched environment, the Vivarium project rapidly began to evolve.

The arts were an important component of the program. Alan believed that a good learning environment included all the modalities and senses. He brought us "Apple Days" with Orff Schulwerk, art, story telling, music ap-

preciation, Thursday lunch concerts and the Apple Hill Chamber Players. Our physical environment was shabby with post-war unpainted bungalows for classrooms and second-hand furniture, but Alan gave us a touch of class with the donation of a Yamaha grand piano. We became more than just a learning community, we became an Apple Vivarium highly-effective learning community.

The Vivarium developed from a project into a program around 1989. The cast of the Vivarium Program had expanded. Alan brought in the brightest and the best as advisory councilors including Seymour Papert, Herb Kohl, Paul MacCready, David Macaulay, Tim Gallwey, Quincy Jones, Marvin Minsky and Richard Dawkins. The school became open in a more literal sense. The first Tuesday of the month we opened our classrooms to the world. Interest in Alan's research drew visitors from around the world—executives from Sony Japan, royalty from Jordan, heads of industry, politicians, educators and all the curious. Many a graduate student earned their Ph.D. studying our students and the Vivarium Program.

The teachers and our students were blossoming from Alan's attention. His insights and vision inspired us to spend long hours inter-cooperating year-round for the common purpose of figuring out how best to utilize computers and networks in our experiential, thematically organized, integrated curriculum. My teachers and I, our students and their parents were totally swept up in the grand adventure of "predicting the future by inventing it," to paraphrase Alan's famous statement. Simply, we were grateful to Alan for giving us the chance to make something collectively of which we all could be proud.

Alan Kay, John Steinmetz, Stewart Brand and others have written extensively about the Vivarium Program and it is unnecessary for me to repeat their thoughts here. However, I will share some lasting outcomes of The Open School's partnership with Alan that are apparent twenty-five years thence.

The five- and six-year-old Green Cluster students were learning important concepts by sorting and categorizing using Venn diagrams, both concretely and then on the computer. Students and teachers were encouraged to explore

and to investigate the possibilities of the technology. Alan always emphasized that computers didn't take the place of tactile, hands-on learning. He saw the computer as an amplifying medium for the concepts that emerged from the learning process.

The seven- and eight-year-olds were scripting and creating small HyperCard stacks (it was called WildCard at first and later changed to HyperCard) as part of their desert studies. These active youngsters learned the concept of a networked fileserver by playing a variation of a relay race game. They also learned fractions kinesthetically by hopping on a piezoelectric floor grid connected to a computer that, when triggered by the students' movements on the floor grid, would produce audio and optical feedback that reinforced students' learning when their steps on the floor grid represented correct fractional statements.

The eight- and nine-year-old students in Yellow Cluster learned to navigate city systems by building a model city and serving on city commissions. Students learned to value their own intellectual property through display of their work on the History Wall, a system of 4 by 8 foot Homasote panels covering the bungalow walls.

The nine- and ten-year-old students in Blue Cluster began to learn to transfer their concrete learning about systems to abstract symbolic systems as they learned to write simulation programs in the Playground language that Alan and the Vivarium computer scientists were developing. Their simulations about the dynamics of marine life were informed by their observations of the aquatic life living in the aquariums the Vivarium team had installed in their classroom bungalow.

The ten- and eleven-year-old Purple Cluster students were studying Jerome Bruner's *Man, a Course of Study*. They were using the technology to graph, chart, and create huge HyperCard stacks to demonstrate connections, relationships and cycles.

The essential questions that drove the thematic units in each cluster were: "What is the relationship between man and his environment?" and "How are plants, animals and humans interconnected and interdependent?"

How did the Open School students' immersion in an idea-based computer design culture—experienced through a thematically-organized, integrated curriculum taught by highly-attentive teachers and a huge cast of teacher's aides, researchers, adjunct teachers, parents, and visitors—prepare the Open School graduates for their middle or junior high school experience? A longitudinal study we did in 1994 addressed this question for Dr. Fred Newman of the Center On Restructuring Schools at the University of Wisconsin at Madison:

Fourteen-year-old Hispanic girl: “[I think the Open School’s instructional approach was] better [than my middle school’s] because I liked having to learn how to do the work myself.” (Point of view really *is* worth eighty IQ points.)

Sixteen-year-old Anglo-American boy: “The computer environment offered at the Open School during the Vivarium project caused me to choose computer classes as electives and try to find any opportunity to use a PC, like in Journalism.”

Sixteen-year-old Anglo-American boy: “Overall my Open School experience was a positive one which opened my eyes to the future and technology. The computer environment I was exposed to during the Vivarium project continues to influence me in school and at leisure. After leaving the Open School, I began to experiment with computers more on my own and made new friends through Bulletin Board services and other special interest groups.”

Fourteen-year-old Anglo-American boy: “The Music Appreciation class [funded by Apple] introduced me to musical performance, which spurred me to take a Band Class. I am now an excellent clarinet player.”

Fifteen-year-old Anglo-American girl: “I think that learning is far easier when there is some type of real-life experience that it revolves around, like it was at the Open School. Textbook lessons leave little room for creativity.”

Fourteen-year-old Asian-American boy: “Open School helped me more for creative thinking than my junior high school. I could think of more ideas than the others in my group [who hadn’t graduated from Open School] because of the influence of Open School.”

To how many people have Alan’s ideas been disseminated? How many students matriculated through the Vivarium Program? How many teachers and administrators have been exposed to the Open School/Vivarium philosophy and methodology? Alan and the Vivarium Program have touched thousands.

Alan’s participatory approach to research and his astonishing powers of discernment that told him when to guide some to find—and when to inspire others to seek—the “good stuff” in effect planted seeds of deep understanding in our minds, seeds of indestructible fertility that qualitatively changed the lives of all of us who participated in the Vivarium Program at The Open School. For that, Alan, I am—and we are—deeply grateful to you. You broadened our horizons, fed our curiosity, and opened a wondrous world of inquiry, dialogue and reflection. Thank you for the amazing journey.

Roberta (Bobby) Blatt served for forty years in the Los Angeles Unified School District (LAUSD) as a teacher, coach and Administrator.

She was principal of the first magnet school in LAUSD, The Open School: Center for Individualization, eventually guiding it to charter status. Working with Alan she turned the school into a model for integration of technology and curriculum. Bobby has since worked with many school districts across the country, integrating technology into the curriculum and implementing project-based learning.

Since 1994 Bobby has been a full-time faculty member at the UCLA School Management Program, providing training and support to over three hundred schools in Los Angeles and other counties throughout California.



Vivarium advisor Glen Keane's impression of kids interacting with animals they have programmed in a simulated, shared ecosystem.

Chunka Mui

Notes on a twenty-five-year collaboration

*Leadership is the lifting of a man's vision to higher sights,
the raising of a man's performance to a higher standard,
the building of a man's personality beyond its normal limitations.*

— Peter F. Drucker

I have the good fortune to have known Alan Kay for nearly my entire professional life. While Alan's activities related to my work were usually at the periphery of his own groundbreaking research, Alan has been a mentor and an active player at almost every critical juncture of my career. I've benefited tremendously from the experience. More generally, my experience with Alan has taught me the immense value of bringing outside perspectives into business deliberations, to provide richer context. Context is indeed worth eighty IQ points.

I first met Alan Kay in early 1985 in a dreary, windowless office in the Chicago headquarters of Andersen Consulting. I was a young associate, less than a year out of MIT, and had been directed to give Alan a demo of my first project. I was young but not unaware; it was hard to be a programmer and not have some sense of Alan's accomplishments. Needless to say, I was a little intimidated.

Alan came in a few minutes late, plopped into a chair, put his feet onto the desk that held up my computer terminal, and said something on the order of, “Ah, finally, something that feels more like home.” Everything about that moment was disorienting and yet liberating. Alan had disheveled hair, a full mustache, a tattered sweater and sneakered feet. Andersen at the time mandated business suits, disapproved of facial hair on men and frowned on women wearing pants; sneakers were outside the realm of possibility. Alan did not fit into any category of the business leaders that I was trying to adjust to in my somewhat ragged transition from college to workplace. Yet he dived into the demo with a business and technical sophistication that dwarfed most who had seen the system. He zeroed in on what was functionally and technically interesting, understood the potential and the limitations, and disregarded the fluff that usually enticed others. Then he pronounced the project good, and made sure that all those he came across knew this to be so. Within a few hours, I was doing demos of the system for the most senior partners of the consulting division.

It was a cathartic event for me and, in a small way, representative of the larger impact that Alan would have on the organization. I was a member of the applied artificial intelligence group in Andersen Consulting’s Technical Services Organization (TSO), a common pool of technical resources for Andersen’s consulting projects across the globe. TSO was essentially the geek squad for the organization; its technical resources were called upon when all else failed. Within TSO, the AI group was the newest and the geekiest. The system that I demonstrated to Alan, an application built for the U.S. Securities and Exchange Commission that performed automated analysis of financial statements, was written in the LISP programming language and ran on a specialized workstation. All this while the rest of the company was building mainframe computer-based transaction processing systems written in COBOL and CICS, and still grappling with the adoption of early-generation IBM personal computers. I was, essentially, at the bleeding edge of the lunatic fringe of an otherwise very buttoned-down organization. Alan’s appreciation and

championing of my work and other similarly advanced technology efforts legitimized and highlighted efforts that might have otherwise languished. His credentials and access to the uppermost ranks of Andersen management raised those kinds of projects from interesting to strategic.

From my vantage point at the bottom of the management hierarchy, I saw Alan's influence across all the levels above me. He energized the lower ranks, giving them context for their work and higher aspirations for how their ideas and efforts could reinvent the firm. He lent his credibility to forward-minded managers like Mel Bergstein (the leader of TSO), Bruce Johnson (the partner who started the AI group) and John Davis (who championed the use of object-oriented programming). Alan helped them persevere in their efforts to make sure that Andersen's service offerings, organization and culture modernized along with the information technology. (See Mel's essay on page 73.) Alan gave Andersen's senior-most management the confidence to invest heavily in enhancing Andersen's technical competencies. And he continually prodded them forward, helping them understand that those investments, while they stretched the firm's practice, were well within the state of the art.

Amid management turmoil at Andersen in 1989, Mel moved to Computer Sciences Corporation, a defense contractor that was attempting to move into the commercial consulting market. Through the doors opened by Alan, I'd had the opportunity to get to know Mel and greatly admired his capabilities. I called to wish him well and soon found myself at CSC.

CSC had bought Index Systems, a small Cambridge-based consulting company that soon became the intellectual driver behind the business process reengineering boom of the early 1990s. Mel, who was helping oversee the entire CSC commercial business, introduced Index's management to Alan Kay and prodded them to utilize Alan's talents as Andersen had. Rather than turn Alan on its internal issues, Index hit upon the idea to leverage his talents directly for its consulting clients.

Even as Index experienced tremendous growth, Index management understood that clients were using reengineering mostly to cut costs. Clients

were not attempting to create strategic change, even though that's how reengineering had initially been envisioned. In large part, the reason was a lack of understanding of the business potential of emerging technologies. To help clients appreciate this potential, Mel and Bob Morison, an Index vice president, conceived of a research program that brought together Alan Kay and Michael Hammer, the former MIT professor who spearheaded the popularization of business reengineering. The program was to explore the strategic implications of information technology. Because of my prior relationship with Alan at Andersen and common MIT roots with Mike, I was drafted to help design and build the program. The research program, Vanguard, which I developed in conjunction with Richard Schroth, was launched in 1991.

Much as Alan helped guide John Sculley through the year of discovery that led to Apple's Knowledge Navigator concept (see John's essay on page 49), Vanguard helped its corporate sponsors develop a rich appreciation for the breadth and depth of technology developments. With Alan's guidance, we assembled a group of advisors that included some of the best-known technologists in the world (including several represented in this book). Our founding advisors included Doug Lenat (the noted AI researcher), Bob Lucky (head of research at Bellcore), Nicholas Negroponte (founder of the MIT Media Lab) and David Reed (former chief scientist at Lotus). Our advisors soon grew to include John Perry Barlow (former lyricist for the Grateful Dead and a leading voice in the politics of the emerging digital environment that he dubbed cyberspace, borrowing a term from William Gibson's science fiction novel *Neuromancer*), Gordon Bell (computer architect and venture capitalist) and Larry Smarr (the noted supercomputing and networking expert). In the course of a few years, senior technology executives from more than a hundred companies in the U.S. and Europe sponsored Vanguard research and relied on our reports and private conferences to help them understand the strategic implications of emerging digital technologies.

Vanguard was in the right place at the right time. Those were the years in which the Internet was racing toward its tipping point. Vanguard helped

its sponsors understand how dramatically the world was changing and how outdated and even counterproductive their basic tools of strategy, planning and information systems development had become. New electronic markets were appearing overnight, under the radar of everyone's long-range plan. The newest technological innovations began not in the corporate arena, where Vanguard's members lived, but in consumer markets, where game computers offered children substantially more processing power and compelling applications than the desktop computers of senior executives. In one memorable Vanguard demonstration, Alan disassembled a first-generation Sony PlayStation in front of a group of corporate executives to highlight the technology that their customers had ready access to, but which was beyond the reach of their IT groups. Corporate computer and communications systems that linked Vanguard's member companies together with their subsidiaries, suppliers, and customers suddenly looked more like liabilities than assets in the wake of the Internet's emerging growth and incredible connectivity.

Those were heady times. Because of the insights, credibility and connections of Alan and other members of our advisory board, our sponsors' understanding of critical technology developments was fed by a visiting cast of researchers, inventors, entrepreneurs, social commentators and senior executives with stories to tell. Among them: the rise of mobile computing and communications, the development of groupware and social media, the evolution of digital media, the inevitable rise of electronic commerce and, correspondingly, the destruction to numerous existing business models.

It was at Vanguard that Bell Labs researcher Bob Lucky asked, "What is a bit?" Attempting to answer that question, Nicholas Negroponte began the series of essays that led to the mega-best-seller, *Being Digital*. It was at Vanguard that many corporate executives were introduced to the Internet. It was at Vanguard that many first saw Mosaic, the first web browser. And at the heart of Vanguard's efforts was Alan Kay.

Following an aspiration set by Alan, we strived to help our sponsors be more than consumers of our research. Real literacy, as Alan often reminded

us, meant being able to read *and* write. Through immersive experiences, active debate and hands-on learning, we tried to help our sponsors develop the deep understanding required for true technological literacy. It was at Vanguard that many corporate executives published their own web page, built their own software agents for business analytics, and absorbed sophisticated arguments about architecture and design. Our goal was not to turn our sponsors into systems programmers but to help give them an appreciation of the context required to make important technology-related business decisions.

The effects were enduring, and not always in predictable ways. One Vanguard sponsor from that time reflected recently, “Alan helped shape my belief that the Constitution can provide insight into how to create IT governance mechanisms that are durable and scalable (e.g., how to balance large and small business unit interests, how to distribute power, and how to balance security and privacy).” Another took it even further: “Alan’s influence and arguments are the *vade mecum* of my design thinking.”

Ironically, even as Vanguard’s sponsors acted on early warning signals of ever more disruptive technology, the executives at Index ignored them. Unlike at Andersen, Vanguard had focused Alan’s energies on clients rather than on itself. Instead of understanding how emerging digital technologies might affect its own consulting business, Index continued to rely on business process reengineering as its primary consulting offering. In one memorable meeting in 1995, Index’s president rejected a proposal to build an Internet strategy consulting service. His explanation: “This might be your religion, but it’s not mine.” It was a fundamental misreading of the market. In a few years, Internet-oriented consultancies experienced unprecedented demand and growth. Business reengineering, however, became a commodity consulting service dominated by very large players. CSC Index withered and soon no longer existed as a separate operating unit.

I left CSC Index in 1995, not long after that memorable meeting. Larry Downes, a former Andersen colleague who had joined me at Vanguard, left at the same time. Together, we began to write a book that captured the lessons

that we learned at Vanguard, and to develop the basic outlines of digital strategy, the consulting service that we had urged Index management to launch. Digital strategy was an approach to developing and unleashing what we would come to describe as “killer apps.” As we worked on the book, I got a call from Mel Bergstein. Mel had started his own firm, Diamond Technology Partners, several years earlier. He called because he was trying to recruit Alan Kay to Diamond’s board of directors, and Alan suggested that Mel call me as well. I soon joined Diamond, and Diamond became the marketing and consulting platform for *Unleashing the Killer App: Digital Strategies for Market Dominance*, which Larry and I published in early 1998.

In addition to the book, I also built on my Vanguard experience and started the Diamond Exchange, an invitation-only learning venue for senior corporate executives. With Alan providing the cornerstone, I recruited a stellar group of contributors to become Diamond fellows and thus a regular part of the Exchange. This time, however, we built a program that was squarely at the intersection of business and technology. The Diamond fellows included technology visionaries like Gordon Bell, Dan Bricklin, David Reed and Andy Lippman. It also included world-class experts in other strategic topics such as economics, business strategy, social trends, and organizational change. This innovative group grew to include Dan Ariely, Vince Barabba, John Perry Barlow, Tim Gallwey, Linda Hill, John Sviokla and Marvin Zonis. And, unlike the technology evangelists that tended to sponsor Vanguard, Exchange members were executives who understood that technology wasn’t just a tool of business but was fast becoming a driver of business change. These executives were positioned to make the necessary changes.

With the Fellows in attendance and the Internet revolution in full blossom, we had little trouble attracting CEOs and their closest advisors to these private gatherings. Diamond conducted world-class research. Alan and the other Diamond fellows were the interlocutors. Soon, an invitation to the Exchange was highly valued, not only for the intellectual content but for the sheer joy of mixing with the best minds on Earth. The result was a virtuous cycle of

research, learning and collaboration that helped us help our clients master their competitive challenges and, in the process, build Diamond into a great consulting firm.

To disseminate the lessons and discussions prompted by the Diamond Exchange, we also launched a great magazine that reached more than 40,000 other senior executives. Led by Wall Street Journal veteran Paul Carroll, the magazine went on to win some of the magazine industry's highest honors. As only fitting for a publication so inspired by Alan Kay's work, we named it "Context."

Alan's influence continues to this day. Among the many things that he taught me is that effective strategies require a deep knowledge of history; otherwise, the same blind alleys are pursued over and over again. That lesson informed my most recent major project, *Billion-Dollar Lessons: What You Can Learn from the Most Inexcusable Business Failures of the Last 25 Years*, a book that I wrote with Paul about how executives can learn lessons from failures, rather than just focus on emulating successes. A key aspect of the consulting practice that we are building around this idea is the power of external perspectives, which has led to a simple principle: Never adopt a new business strategy without independently stress-testing critical assumptions and key design elements. Truly independent interlocutors can bring fresh perspectives and tough questions to any critical business decision and, in the process, dramatically increase the odds of success. Alan is, of course, one of the first interlocutors that we recommend to our clients.

Chunka Mui holds a B.S. from MIT. He has had the pleasure of working with Alan Kay during every one of his professional endeavors since.

He started his professional career at Andersen Consulting, now Accenture, where he was a member of Andersen's world headquarters artificial intelligence group and a founding member of the firm's Center for Strategic Technology Research. Chunka was vice president at CSC Index, where he co-founded and directed the Vanguard emerging technologies research program, and a managing partner and Chief Innovation Officer at Diamond Management & Technology Consultants.

Chunka is currently co-founder and managing director of the Devil's Advocate Group, a consulting firm that helps management and investors stress-test business strategies. He also serves on the board of directors of Viewpoints Research Institute.

Mel Bergstein

Context, Inspiration and Aspiration: Alan Kay's Influence on Business

I met Alan in the early 1980s, when I was a young partner at Andersen Consulting running a small technology group in the New York office.

Andersen Consulting was the consulting arm of Arthur Andersen & Company, which was one of the largest accounting firms in the world at the time. The consulting arm eventually separated from Andersen to become Accenture. It had been around for more than a quarter of a century but was unfocused until the late 1970s, when it began to concentrate its consulting practice on computer-based information systems. The strategy was prescient. The market was enormous, because large enterprises worldwide were digitizing, and the work that Andersen Consulting did in building transaction processing systems was repeatable. Andersen Consulting developed a detailed, if not somewhat rigid, development methodology and made significant investments to train its consultants in that methodology. The strategy was tremendously successful, allowing the business to scale quickly and profitably.

But, as the strategy unfolded and the practice grew, quality problems emerged. The methodology, known as Method/1, was good at leveraging large numbers of smart young people, but was ill-suited for addressing complex technical issues. To combat the problem, senior management began hiring

experienced technical people into a centralized pool in Chicago. But the central group could not satisfy the demand for all the systems projects underway all over the world, and travel schedules created huge attrition. Proving even more difficult, the influx of new experienced technical people was a cultural challenge.

Andersen's culture was very strong because it had a long-held policy of hiring associates straight from college and promoting from within. Following the practice developed for Andersen's accounting business, all consultants followed a common educational curriculum that extended throughout their careers. This curriculum emphasized industry knowledge and the ability to manage large projects using the firm's methodology. To emphasize this commonality and to build a single culture, consultants regularly travelled from across the globe to learn together at the firm's main training facility near Chicago. Partners (Andersen was a privately held partnership at the time) almost all rose through the ranks of the firm and were elected based on their industry expertise, project management skills, and success at generating revenue. It was very hard for technical specialists to fit into this rigid system. Many technical people at Andersen developed their skills in other firms and did not go through the cultural bonding of Andersen's career training. And Andersen's promotion and rewards systems were not designed to accommodate them. The criteria for partner selection, for example, did not include technology skills.

Something had to give. Either changes were going to be made to a nascent systems business, or Andersen's consulting unit would fail as so many others like it had and would.

The first step was to decentralize the technical pool. The first decentralized unit was started in New York, and I was drafted to run it. I accepted with some reluctance.

Enter Alan.

It was the habit of the New York technology group (about eighty people) to come together once a month to review project status and to learn. We always had an outside speaker. John Davis, a gifted intellect and voracious reader with

a strong interest in software development, had begun to develop an interest in object-oriented programming and Alan Kay. We invited Alan to speak to our group. He accepted, and our world would never be the same. Alan connected the group to the world of science and the history of information technology. He gave us historical perspective, high aspirations, and a higher purpose. He inspired us. The genie was out of the bottle.

In late 1984, I moved to headquarters in Chicago to run the central technology group and to coordinate the decentralized technology units scattered across the world. My job also included responsibility for the consulting division's firm-wide training curriculum, technology research, and software products. One of my first moves was to create a firm-wide technical advisory committee, of which Alan was a prominent member. It was a great platform, and it gave Alan a chance to influence Andersen's entire global technology community. Direct consequences of Alan's involvement included the application of object-oriented programming tools and methods to large systems projects and to Andersen's own software development tools, the funding of the Institute for Learning Sciences at Northwestern University, and the establishment of a strategic technology research center that recruited Ph.D.s from a number of technical fields. For a firm that, at the time, still taught COBOL to all incoming associates using paper coding sheets and punched cards, and was building its CASE tools in BASIC running on MS-DOS, these were revolutionary developments.

More generally, Alan helped to educate and inspire a large number of Andersen's consulting people around the world. Alan's involvement gave a generation of Andersen's technical people license to grow and assume leadership positions. He taught them a greater appreciation for technology, problem solving, and design. In my conversations with several Andersen Alumni who are now chief technology officers of significant enterprises, all cited Alan's influence on their worldview and on how they think. One of the technical people on whom Alan had a significant influence was Chunka Mui, who was fresh out of MIT when Alan met him in our Chicago office. Chunka would

play a prominent role in another stage of my career, as I will discuss. (You can read Chunka's story starting on page 63 of this book.)

Alan's involvement also helped Andersen's management appreciate and leverage the deep technology skills within the organization. By doing so, he assisted Andersen in developing its ability to design and integrate complex software architectures—and grow into an industry-leading company. While Accenture's success is certainly due to the efforts of many, Alan had an enormous influence well beyond what we realized at the time.

In 1989, I left Andersen amid management turmoil and joined Computer Sciences Corporation (CSC) to help lead their move from serving government organizations to commercial consulting. My time there was relatively short, but I did have the opportunity to introduce that company to Alan. Again, Alan's influence was pivotal. But I'll leave that story for Chunka, who joined me at CSC.

In 1994, I again had the pleasure of working with Alan and benefiting from the catalytic effect he can have on organizations. That's the year that, along with Chris Moffitt and Mike Mikolajczyk, I started Diamond Technology Partners (now Diamond Management and Technology Consultants). We positioned Diamond to fit in the gap that then existed between McKinsey and Accenture. Diamond's mission was to bring new technology to large companies using a management consulting model, like McKinsey, rather than an integrator model, like Accenture. I was fifty-two years old and had lived through some of the best and worst the consulting industry had to offer. With the lessons firmly in mind, we designed Diamond to foster internal collaboration between talented technologists and industry strategists, something that no services firm had been able to achieve to that point. Additionally, we built Diamond on the assumption that technology would soon become a critical part of all CEOs' arsenals to shape competition within and across industries.

The issues at the intersection of technology and strategy were not top priorities in corporate executive suites at the time. The concept was, however, evidently clear to the best and brightest young people and allowed our

little company to successfully recruit experienced consultants from top-tier consulting firms and talented graduates from some of the best U.S. business schools. With a strong talent pool and a sense of where business was heading, we focused on helping industry-leading companies address complex business problems with strong technology components.

Patience and determination started to pay off in late 1995, when the Netscape IPO put the Internet front and center on every investor's radar and therefore on the agenda of every large-company CEO and board of directors. Sensing our opportunity, we recruited Alan onto Diamond's board of directors. Alan's credibility and long-standing relationship helped us recruit Chunka, who during his tenure at CSC Index had been studying the digital strategy issues that our clients were coming to appreciate. Chunka's book, in collaboration with Larry Downes, was published shortly thereafter in 1998. *Unleashing the Killer App* became one of the most popular business books of the time, presaging the huge move by major businesses to adopt the Internet. The book became Diamond's calling card and its practice guide. More than one CEO invited us in, closed the door behind us, confessed a lack of understanding and asked for help.

Alan, now ensconced on the Diamond board, provided magic similar to his contributions at Andersen. He gave credibility to our upstart company and helped attract great technology talent to the firm. Alan was an active board member and brought a perspective on science and technology to a board of financial and operations people. Alan also brought his experience with the emerging companies like Apple, and with lumbering large ones like Xerox and Disney. He was a rich repository of stories about what worked and didn't work in the technology world. And, of course, Alan was a beacon to the people of Diamond. Just as he had done at Andersen and Vanguard, he inspired us to a higher purpose. He mesmerized us and energized us.

Whereas much of the consulting activity in that time was focused on serving startup companies, Diamond remained focused on industry leaders. We helped world-class companies understand how the Internet allowed, and

indeed required, them to embrace disruptive innovation, rather than just incremental change. Because of this, both clients and investors rewarded us handsomely. At its peak, Diamond was briefly valued at almost three billion dollars. Perhaps a better sign of our relevance was that, since inception roughly fifteen years ago, Diamond has provided more than two billion dollars in services to its clients. The market crash that closed the dot-com era extinguished the irrational investor exuberance and many of our competitors of that period. Time, however, has proven that the major strategic challenges and opportunities of our time do lie at the intersection of business and technology. Alan had a great hand in teaching us this principle and, due in no small part to his efforts, Diamond continues to thrive today.

Mel Bergstein spent twenty-one years with Arthur Andersen & Company's consulting division (now Accenture). He became a partner in 1977 and served as Managing Director of worldwide technology from 1985.

Mel left Andersen in 1989 to take up executive management roles at Technology Solutions Company and Computer Sciences Corporation.

In 1994 he founded Diamond Management & Technology Consultants, Inc., where he was Chairman and CEO until 2006. Mel continues to serve as Chairman at Diamond, in addition to serving on the Boards of Directors of several other organizations.

Larry Smarr

The Emergence of a Planetary-Scale Collaboratory for Data-Intensive Research

Introduction

I had the good fortune to work with Alan Kay as part of the CSC Vanguard team in the 1990s and always valued the insightful critiques he would make of presentations during the Vanguard sessions. Although I knew about Alan's fundamental contributions to user interface design, I came to understand also that he had a longtime interest in developing collaborative multi-user software to support many application areas of interest. This research with his colleagues eventually evolved into the Croquet software development kit (SDK), which can be used to support "highly scalable collaborative data visualization, virtual learning and problem solving environments, three-dimensional wikis, online gaming environments (MMORPGs), and privately maintained/interconnected multiuser virtual environments."¹

During the two decades that Alan and his colleagues were working on what became Croquet, the two institutes I founded, the National Center for Supercomputing Applications (NCSA) and the California Institute for Telecommunications and Information Technology (Calit2), were also deeply engaged in developing a series of collaboration environments, with a focus

¹http://en.wikipedia.org/wiki/Croquet_Project

on collaborative analysis of data. Alan's emphasis on simplicity and natural human-computer interfaces made a deep impression on me. I have kept these ideas in mind as the global team I was part of developed a working version of a collaboration metacomputer [31] as big as planet Earth, but with many of same characteristics as a personal computer.

I briefly describe the two tracks we followed: the first was similar to Alan's notion of a collaborative environment for sharing personal computer desktops and the second a series of experiments on *tele-immersion*, innovative software/hardware environments that enable sharing of entire rooms for data intensive analysis using advanced technologies.

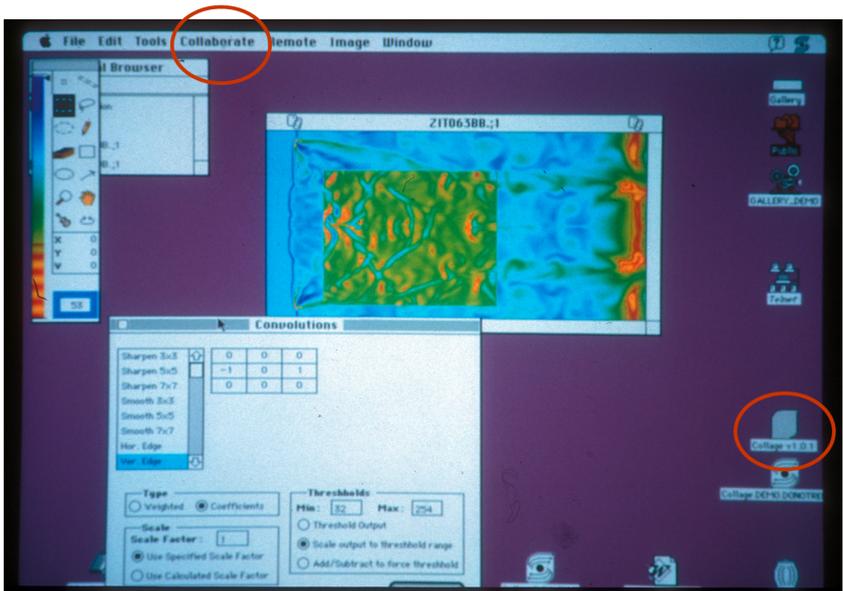
Desktop Collaboration Software Systems

The early 1980s, the period which led to the funding of the National Science Foundation (NSF) supercomputer centers, including NCSA in 1985, coincided with the period of the birth of the IBM PC and the Apple Macintosh. I had early versions of both, even as I was advocating for a national supercomputer with a cost over \$10 million. Even though the computational scientists needed access to powerful vector computers, I was convinced that the correct user interface was through the personal computer. So our NCSA software development team started using the phrase "Hide the Cray," by which we meant making the remote supercomputer appear as an icon on the network-connected PC or Mac. This concept led to the development by NCSA staff of NCSA Telnet,² which allowed multiple remote sessions to be run from a PC or Mac.

In the late 1980s a whole series of PC and Mac software was turned out by NCSA, such as NCSA Image, bringing the flexibility of the Mac to visual and analytic analysis of complex data, often generated by our supercomputers. By 1990 the NCSA Software Development Group (SDG), led by Joseph Hardin, had created NCSA Collage, which was synchronous desktop collaboration

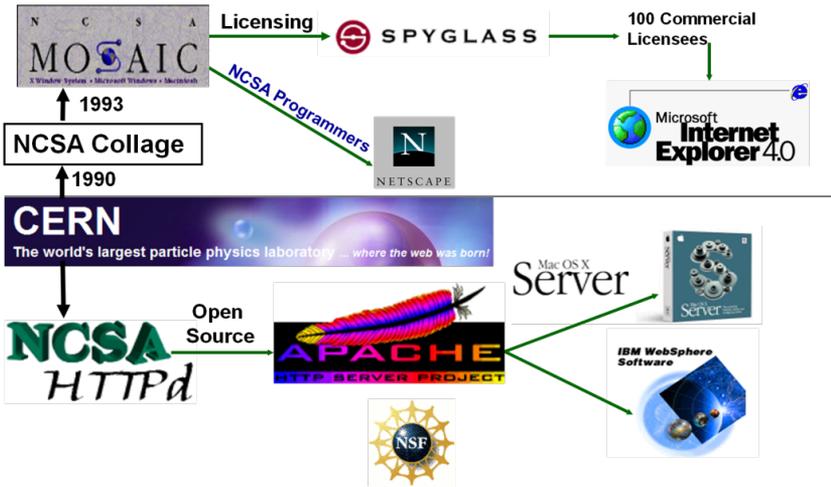
²http://en.wikipedia.org/wiki/NCSA_Telnet

software which could run on Windows, Mac OS, and UNIX. Collage built on the graphic interface ideas in the previous software tools, but provided a common windowed view to collaborating users with shared white boards, image display and analysis, color table editing, and spreadsheet display of floating-point numbers. The image below (from Susan Hardin, NCSA) shows a screen capture of NCSA Collage for the Mac. I have circled the “Collaborate” tab on the menu line and Collage’s icon which appears as just another desktop application.



With the development by CERN’s Tim Berners-Lee of the Web protocols in 1990, the NCSA SDG realized they could introduce not only documents into Collage, but hyper-documents as well, and set up a sub-project to develop the needed software. This project, NCSA Mosaic, quickly became a world of its own as the leaders of the Mosaic team, Marc Andreessen and Eric Bina, developed the Unix Mosaic browser and began releasing it in 1993. Their NCSA Mosaic group grew and soon the HTTPd Mosaic server software, as well as Windows and Mac versions of the Mosaic browser, were made available.

The ability to download freely both a graphical web browser and server software set off exponential growth in the number of people making their own web sites and viewing others. NCSA's web server became the most visited web site in the world, leading us to develop the world's first parallel web server. The rest is history (see diagram below). Andreessen and Bina joined Jim Clark in founding what became Netscape, Microsoft licensed Mosaic through Spyglass, a local company that had taken over licensing from the University of Illinois, and the Apache Software Foundation created the Apache server from the open source Mosaic server software.



Yet in spite of the global transformational nature of Mosaic and its progeny, NCSA Collage attracted very few synchronous collaboration users. It was time consuming for the NCSA SDG to keep the three separate code bases developed in parallel and so eventually the development on Collage ceased. Somehow, the lesson was that single-user personal computer software is adopted much more readily than collaboration software.

With the announcement of Java by Sun Microsystems in the early 1990s, the NCSA SDG realized it could have just one software base for building collaboration software, which would be automatically cross-platform. The introduction of Java led to the NCSA Habanero project [16] in 1995, which

recreated the NCSA Collage functionality, but written entirely as a Java application. The Habanero software system provided the necessary framework in which one could create collaborative work environments and virtual communities, as well as to transition existing applications and applets into collaborative applications. At the time, Habanero was perhaps the largest single Java application yet written. However, in spite of the Wall Street Journal in 1996 saying, “NCSA hopes Habanero will take the Web one step further—into collaboration,” its use was quite limited and again development eventually stopped.

Although it was frustrating to me that in spite of how useful these collaborative software systems were, they did not take off in adoption like the web browser, it was still clear to me when I watched people using synchronous collaboration software that sooner or later this is what software and the Internet were destined to make possible. Since full desktop collaboration systems are still not widely used, nearly twenty years after NCSA Collage appeared, perhaps we were just a bit too early in our view of what the Internet could make possible...

Perhaps more successful in terms of adoption was a parallel track at NCSA, starting a little before the NCSA Collage project, which was to build collaboration environments using the most advanced technology available that would “sew” whole rooms together, whether those rooms were physical or virtual, to allow for tele-immersive collaborative analysis of data-intensive research.

A Vision of the Collaborative Future

The first prototype of this idea was produced in 1989 when NCSA, together with Sun Microsystems and AT&T, put on a demonstration termed *Televisionization: Science by Satellite*, which was meant to illustrate how collaborative use of high performance computing with visualization might be made possible in the future using fiber optic networks. Since availability of those networks for academic researchers was a decade in the future, we conceived of using

analog video technology, transmitted over TV satellites, to emulate that future. AT&T put large satellite dishes next to NCSA in Champaign, Illinois and outside Boston's Museum of Science, close to the SIGGRAPH'89 meeting, to establish the link from UIUC to Boston.



UIUC Professor of Theoretical and Applied Mechanics Bob Haber used a track ball on stage to send commands over a 9,600 baud return dial-up line to rotate a dynamic visualization being computed on an Alliant FX graphics mini-supercomputer, which was creating a visualization of the simulation of a crack propagation in a plate being computed in real-time on a Cray-2 supercomputer at NCSA. All the while (see screen capture image), there was a larger-than-life video image of Professor Bob Wilhelmson at NCSA on the stage (center) in Boston discussing the event with Donna Cox (extreme right), Bob Haber (standing left), and myself (center right). While we had to use an analog video stream sent by satellite to emulate the future digital transmission of data, reviewing the recording of the event³ is eerily similar to what we actually can do today with 10 Gbps dedicated fiber optic networks, as described later.

As then-Senator Gore said in a pre-recorded video played as part of the demo, “[we were] using satellite technology ... to create a demo of what it might be like to have high-speed fiber-optic links between advanced computers in two different geographic locations.” I stated during the demo, “What we really

³Video of the session is available from Maxine Brown, EVL, UIC. A digitized version can be viewed at: <http://www.youtube.com/watch?v=3eqhFD3S-q4>

have to do is eliminate distance between individuals who want to interact with other people and with other computers.” This has been the holy grail of the next two decades of research that I have pursued with my co-workers.

Leading-Edge Collaboration Environments: Shared Internet

The development of Silicon Graphics computers, putting the power of a graphics mini-supercomputer into a workstation, enabled new immersive versions of virtual reality (VR) to be conceived, such as the CAVE [17] and ImmersaDesk [18] created by Tom DeFanti, Dan Sandin, and their colleagues at the University of Illinois at Chicago’s Electronic Visualization Laboratory (UIC/EVL) in the early 1990s. These various interactive stereo interfaces used the CAVELibrary [27] VR software API to display images on the walls of the CAVE, and the CAV-ERNsoft library [26] to link remote



virtual spaces over networks. In 1996, NCSA industrial partner Caterpillar [24] used ATM networks between Germany and NCSA to support a collaborative VR session containing a three-dimensional stereo life-size rendering from a CAD database of a new earth mover. In this shared data-space, Caterpillar used video streams as avatars to represent the remote participant, creating arbitrarily oriented virtual video screens floating in the shared virtual space. With this international collaborative VR infrastructure they discussed possible CAD modifications so as to make maintenance easier in the field. Caterpillar was an innovative industrial partner, driving virtual reality advances at NCSA for over a decade.

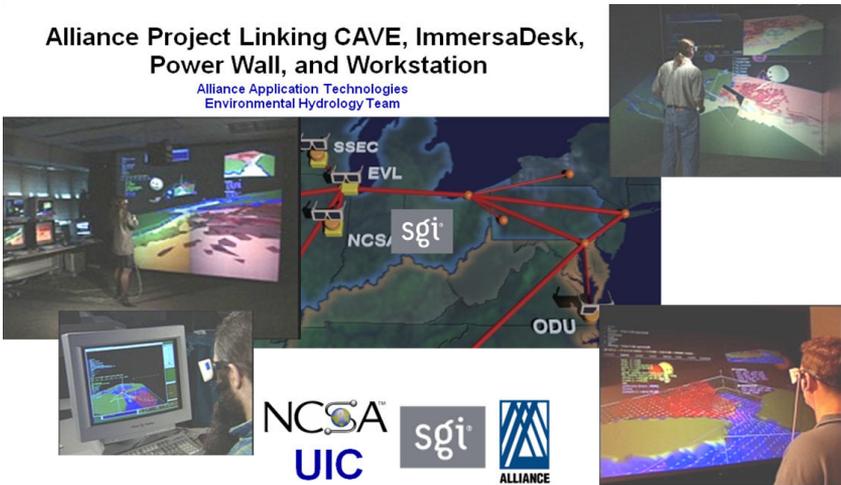
By 1997, the NSF had expanded two of the NSF supercomputer centers, NCSA and SDSC, into Partnerships for Advanced Computational Infrastructure (PACI). The PACIs were able to use the newly NSF-funded very

high-speed Backbone Network Service (vBNS)⁴ to explore innovative modes of collaboration. The NCSA PACI was called the Alliance and one of its first activities was developing tele-immersion [25]—the union of audio and video conferencing, networked collaborative VR, and image-based modeling for data-intensive applications.

Tele-immersion was accomplished by combining CAVERNsoft with specific application domain visual analysis software, such as the Vis5d,⁵ an OpenGL-based volumetric visualization program for scientific datasets in three or more dimensions, to form CAVE5D.⁶ CAVE5D was augmented with remote interaction techniques and camera choreography capabilities provided by the VR application Virtual Director developed by Donna Cox, Bob Patterson, and their co-workers at NCSA, with colleagues and students at UIC/EVL.⁷ All this was run over the vBNS, which supported speeds of 155 to 622 Mbps on the shared Internet.

Alliance Project Linking CAVE, ImmersaDesk, Power Wall, and Workstation

Alliance Application Technologies
Environmental Hydrology Team



⁴http://www.nsf.gov/od/lpa/nsf50/nsfoutreach/htm/n50_z2/pages_z3/47_pg.htm

⁵<http://vis5d.sourceforge.net>

⁶<http://www.mcs.anl.gov/dmickelso/CAVE2.0.html>

⁷Virtual Director was originally created at NCSA by Donna Cox, Bob Patterson and Marcus Thieboux. The software was further developed by Cox, Patterson, Stuart Levy and Matthew Hall.

In the image above one sees Donna Cox in front of a PowerWall (tiled wall with rear video projectors upper left), Bob Patterson in a CAVE (upper right), Stuart Levy at a workstation (lower left), and Glen Wheless at an ImmersaDesk (lower right). Donna, Bob, and Stuart are all at different locations at NCSA and Glen is at Old Dominion University in Virginia. They are sharing the Virtual Chesapeake Bay,⁸ a visual representation of data produced by a coupled physical/biological simulation, using Virtual Director to navigate the space; it could also record the session. Note the three-dimensional smiley face avatars floating in the various spaces, which represent the location in the 3-space of the remote collaborators.



Argonne National Laboratory (ANL) drove the next stage of innovation for tele-immersion, utilizing the vBNS capability to use IP multicast to develop the Alliance Access Grid (AG), which allowed a large number of sites to join into a collaborative session, each with its own video and audio streams. Development of AG was led by ANL's Rick Stevens and its Math & Computer Science Division, one of the principle Alliance partners, as a part of the Alliance National Technology Grid. It has been widely used over the last decade to support multi-site video conferencing sessions. The image above was taken during one of the Alliance digital "chautauquas"⁹ on September 14, 1999. The

⁸<http://www.computer.org/portal/web/csdl/doi/10.1109/38.511854>

⁹<http://access.ncsa.illinois.edu/Releases/99Releases/990713.Grid.Chautauqua.html>

collage of live video feeds shows me giving a lecture from Boston University (along with my streaming Power Point slides) and multiple video feeds from six sites across the U.S. (including Rick at ANL left and below me), plus one from Moscow in Russia (left of me).

Thus, besides driving the early use of IP multicast video streams over the Internet, the Access Grid also drove early international video collaborations using the Internet. To provide a national and international peering point for advanced research and education networks, NSF funded the Science, Technology, And Research Transit Access Point, or STAR TAP,¹⁰ located in Chicago and managed by the UIC's EVL and ANL, with Ameritech Advanced Data Services. STAR TAP grew into a major exchange for the interconnectivity and interoperability of both national and international research networks. The Alliance Access Grid used the STAR TAP to support the broad collaboration shown in the image.

High Performance Collaboration Environments: Dedicated Internet

At about the same time that the AG took off, our team realized that the traditional shared Internet was blocking innovation. We wanted to keep the Internet Protocol, but the enormous build-out of fiber optics in the 1990s meant we no longer needed to live in a “bandwidth scarcity” regime. Rather, by doing the heretofore unthinkable, giving a fiber, or at least a 10 Gbps wavelength on the fiber, to an individual user, we could jump several orders of magnitude in bandwidth capability into the future. In Illinois NCSA, ANL, and EVL worked with the Governor's office to create the I-WIRE¹¹ “dark fiber” network for the state. About the same time Indiana created the I-Light fiber network. Today there are over two dozen state and regional optical research and education networks.

¹⁰<http://www.startap.net/startap>

¹¹<http://www.iwire.org>

This major change in architecture of the Internet, arguably the biggest change since the creation of the Internet, created global availability of dedicated 1 Gbps and 10 Gbps optical fiber networks, providing a research network parallel to the shared Internet, but used only by researchers engaged in data-intensive projects. These networks retain the Internet Protocol in the Internet Layer of the Internet Protocol Suite, but do not necessarily use TCP in the transport protocol layer. Whereas the traditional shared Internet traffic uses Layer 3 in the OSI Model, the dedicated optical networks most often use Layer 2 or even Layer 1.

The usual mode of usage is to have a point-to-point uncongested optical link, or a few such fixed links, which means that there is fixed latency, removing jitter. Finally, the bandwidth available to a single user is between a hundred and a thousand times that of the jittery shared Internet, which typically provides end-users only tens of Mbps bandwidth. This gets around a lot of the technical difficulties experienced by the AG, since streaming media is now predictable, high speed, and jitter-free. Also it changes the mode of moving gigabyte- to terabyte-sized data objects from FedEx to FTP. For instance, it takes ten days to move a 1 TB data object over 10 Mbps (typical of today's shared Internet), whereas it takes approximately 10 minutes over a 10 Gbps lambda.

In the early 2000s there was a rapid growth of state and regional networks (e.g., CENIC in California, Pacific Wave in the Northwest), national networks (National LambdaRail, NLR, and more recently the Internet 2 Dynamic Circuit Network, I2DCN), and international interconnection networks (Global Lambda Integrated Facility, GLIF) which led to an explosion of innovation and experimentation. For instance, by iGrid 2005,¹² hosted by EVL's Maxine Brown, Tom DeFanti, and myself, in the new UCSD Calit2 building, there were fifty real-time application demonstrations from twenty countries [21]. This included the first transpacific transmission of the new 4K digital cinema (approximately 4000 by 2000 pixels at 24 frames per second), compressed

¹²<http://www.igrid2005.org>

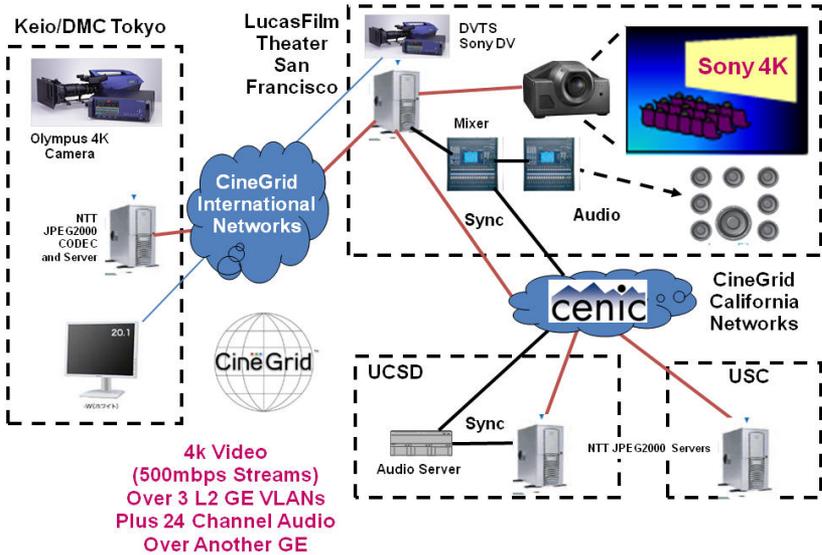
using NTT Network Innovation Laboratories' JPEG2000 codecs to streams of about 0.5 Gbps running over dedicated gigabit fiber channels between Keio University in Japan and Calit2 at UCSD.

This new-found ability, to have jitter-free optical paths that have larger bandwidth than the underlying high-resolution video and audio streams, meant that digital media artists became one of the major drivers of this new collaborative fabric. In particular, universities and private sector companies from the U.S., Canada, Japan, and the Netherlands came together to form a non-profit project called CineGrid [33].¹³ CineGrid's mission is "to build an interdisciplinary community focused on the research, development and demonstration of networked collaborative tools, enabling the production, use and exchange of very high-quality digital media over high-speed photonic networks." It has an annual meeting every December hosted by Calit2 at UCSD. This brought the focus of a wide community of practice on new forms of digital collaboration.

As an example, one year after iGrid 2005, on October 25, 2006, the CineGrid team set up four dedicated gigabit Ethernet vLANs to form a collaborative network between Keio University's Research Institute for Digital Media and Content (Tokyo), Lucasfilm's Letterman Digital Arts Center (LDAC in San Francisco), USC's School of Cinematic Arts (Los Angeles), and Calit2 (San Diego).¹⁴ Working with engineers from ILM and Skywalker Sound, the CineGrid team re-configured the LDAC Premier Theater, normally used to show traditional movies, to enable network delivery of up to 10 Gbps for real-time playback and control of 4K digital motion pictures and 24 channels of uncompressed, 24-bit digital audio from three remote sites. Then for the first time, 2K (HD) and 4K (digital cinema) resolution digital motion pictures and 24-channel digital audio were streamed from three different locations in real time, then synchronized and mixed live for an Audio Engineering Society audience in the LDAC Theatre.

¹³<http://www.cinegrid.org>

¹⁴<http://www.calit2.net/newsroom/article.php?id=958>



Chris Sarabosio, a sound designer at Skywalker Sound, said: “With the experimental system used at the CineGrid@AES event, I was able to control playback and mix 24-channel audio interactively while watching the synchronized picture on the big screen just like I do normally, only this time the audio servers were 500 miles away connected by CineGrid. This approach clearly has the potential to eliminate distance as a barrier to collaboration.”

The beginning of the rise of the new optical fiber Internet infrastructure led me in 2001 to organize what became the NSF-funded OptIPuter project¹⁵ [22], which supported major teams at Calit2 and EVL plus a number of other academic and industrial partners. The application-driven OptIPuter project set out to explore how the availability of these new dedicated 10 Gbps Internet lightpaths (“lambdas”) [29] would transform data-intensive science. Use of these lambdas provided end-users “clear channel” access to global data repositories, scientific instruments, and computational resources from the researchers’ Linux clusters in their campus laboratories. These clusters can be

¹⁵<http://www.optiputer.net>

configured as “OptIPortals” [20], providing the end users with local scalable visualization, computing, and storage. Using the 10 Gbps lightpaths available over the NLR, I2DCN, and GLIF, this new distributed architecture creates an end-to-end “OptIPlatform” for data-intensive research [30].

For collaboration purposes, the OptIPlatform is being used today for combining high-resolution video streams (HD, 4K) with OptIPortals in a variety of ways, so that virtual/physical workspaces can be established on demand. We have been fortunate to work with the talented group at the University of Michigan, which has multiple OptIPortals, and a long and distinguished history of research on scientific collaboration modes, to better define the social science and human interface issues. The psychological effect for end-users is that their rooms are “sewn together,” regardless of distance, and massive amounts of data can be interactively visualized and shared—essentially realizing the vision of the Science-by-Satellite experiment twenty years ago. The manner in which the audio-video streams are coupled with the OptIPortals or CAVEs is an area of active research, so I will end by briefly describing three current modalities.

First, rooms such as auditoriums that have HD or 4K projectors can use optical networks to link to remote sites that have OptIPortals. The video streams can range from heavily compressed commercial H.263 (typically less than 1 Mbps) up to uncompressed (1.5 Gbps HD) video. In the photo we see Professor Ginger Armbrust at the University of Washington explaining to me in the San Diego Calit2 auditorium the single nucleotide polymorphisms which are marked along the chromosomes of the diatoms she is visualizing on her OptIPortal. Using the methodology developed by the UW Research Channel, we are using an uncompressed HD video stream to link her lab with the Calit2 auditorium using a point-to-point 10 Gbps lambda over CENIC and Pacific Wave optical fiber infrastructure [32]. This experiment was in support of the Moore Foundation-funded Community Cyberinfrastructure for Advanced Marine Microbial Ecology Research and Analysis (CAMERA) project. This method has also been used extensively, with different levels of



HD compression, between the two Calit2 campuses, Calit2 and Australia, and Calit2 and NASA Ames [28].

The Scalable Adaptive Graphics Environment¹⁶ (SAGE) developed for the OptIPortal by EVL enables the highest performance version of lambda collaboration yet through its Visualcasting [23] feature, which distributes HD video and visualizations in real time to multiple sites. It does not require IP multicast in routers as Access Grid did, but rather achieves multicast by using commodity clusters (SAGE Bridges) to replicate and to broadcast real-time ultra-high-resolution content to multiple sites. To scale up the resolution or number of sites, one just increases the number of cluster nodes.

The photo below was taken during an HD teleconference at EVL in Chicago. One sees on the EVL OptIPortal behind EVL director Jason Leigh

¹⁶<http://www.evl.uic.edu/cavern/sage>



the HD video streams from lambda connections to University of Michigan (upper right); the SARA supercomputer center in The Netherlands (lower right); the Gwangju Institute of Science and Technology (GIST) in Korea (upper left); and, the Korea Institute of Science and Technology Information (KISTI) (lower left). In this experiment, EVL, Michigan, SARA, KISTI and GIST sent video from their facilities to two 10 Gbps SAGE Bridges at StarLight (which had evolved directly from STAR TAP, mentioned previously), and received only those videos they wanted to receive. For example, while SARA sent its video stream, it chose to only receive streams from EVL and Michigan. The video was lightly compressed (approximately 600 Mbps per video stream), requiring around 2 Gbps to be streamed over TransLight/StarLight to/from SARA. Here one can see there are five rooms “sewn together” over three continents, creating a planetary-scale collaboratory.

Finally, in November 2009 at Supercomputing 2009, Calit2’s Jurgen Schulze and Kara Gribskov did a demo reminiscent of the televisualization event between NCSA and Boston two decades earlier. The photo (from Tom DeFanti) is taken in Portland, Oregon on the SC’09 exhibit floor—Jurgen is in San Diego, in the Calit2 StarCAVE [19], a 3m³ virtual reality display,



and is engaged in an HD teleconference with Kara who is using a ten-panel NexCAVE portable virtual reality display. The videoconferencing HD stream uses commercial LifeSize HD units and the CENIC network is used to interact with the data in three dimensions, which is shown simultaneously on both VR displays. The LifeSize uses 6 Mbps and the interaction, mainly navigation in this demonstration, is done by low latency/low bandwidth exchange of tracker information, once the models are downloaded to each display's cluster. When the models are updated in any significant way, the data exchange can consume every bit of bandwidth available. To facilitate large data updates and low latency joint navigation, CAVE systems are generally connected by 1GE or 10GE Layer 2 vLANs and use UDP-based transmission protocols to maximize transfer rates and minimize latency, as compared to the 1997 tele-immersion demo which used the shared vBNS Internet.

Summary

This quest for tele-immersion has come a long way in two decades and the dream that fiber optics could eliminate distance on a global basis has begun to

come true. There are currently between fifty and a hundred OptIPortals, and a similar number of CAVEs, in use around the world. Many demonstrations are carried out each year over the global OptIPlatform. However, for this new global infrastructure to really take off we dearly need the techno-socio-computer science insights that Alan would naturally give us!

Acknowledgements

The author would like to thank all his colleagues at the National Center for Supercomputing Applications, the National Computational Science Alliance, and the California Institute for Telecommunications and Information Technology for the 25 years of joint work on these ideas. The partnership with the UIC Electronic Visualization Laboratory and the Argonne National Laboratory was essential to the innovations described. Much of the work was supported by the National Science Foundation's Supercomputer Center and PACI Programs, OptIPuter grant, many other NSF and DOE grants, as well as the Gordon and Betty Moore Foundation CAMERA grant.

Larry Smarr was for fifteen years a founding director of the National Center for Supercomputing Applications (NCSA) where he helped drive major developments in planetary information infrastructure: the Internet, the World Wide Web, scientific visualization, virtual reality, and global telepresence.

In 2006 he received the IEEE Computer Society Tsutomu Kanai Award for his lifetime achievements in distributed computing systems. He is a member of the National Academy of Engineering, and a Fellow of the American Physical Society and the American Academy of Arts & Sciences.

Larry is currently the Harry E. Gruber Professor of Computer Science and Engineering at UCSD, and the founding director of the California Institute for Telecommunications and Information Technology (Calit2).

Andy van Dam

*Reflections on what Alan Kay has meant to me,
on the occasion of his 70th birthday*

I'm delighted to have the opportunity to make some personal comments on a few of Alan's many contributions. Since several of his close colleagues are far better equipped than I am to deal with the many technical contributions that his unusually fertile mind has led to, I can focus my comments on Alan's personal impact on me: I've had the pleasure of knowing him for more than four decades, breaking bread with him, engaging in many a spirited discussion and, most importantly to me, being inspired by him in multiple ways.

During my own long career I've had the good fortune to have stimulating personal interactions with several gurus that went well beyond what one can learn from papers. Their modes of thought helped shape my thinking and sometimes shook up my overly comfortable and somewhat limited world view.

Among these I count J. C. Licklider in the mid-sixties as the earliest, followed in short order by Doug Engelbart, whom I got to know at his "Mother of All Demos" in 1968, and Alan, whom I also met there. It was clear to me that Alan was a fellow rebel and contrarian but one with far broader grounding in computing (not to mention physical sciences, engineering, biology, music, philosophy, ...) than I had, despite my having an undergrad degree in Engineering Sciences and a Ph.D. in Computer Science. I remember thinking that here

was a wonderfully erudite fellow whose ideas and work I should track. But regrettably that didn't happen often enough—he was on the west coast, at the epicenter of the “PARC phenomenon” and all the great inventions it spawned, while I was on the east coast, working on my own interactive graphics and hypertext agendas (the latter inspired by working on the Hypertext Editing System with fellow Swarthmorean Ted Nelson, who introduced me to the idea). Neither bicoastal travel nor electronic communication was as common then as it is now.

During the next several decades our paths did occasionally intersect, and I have fond memories of unleashing Alan, in one of his typical visionary rants, on a standing-room-only class of my students and others who wanted to hear him. He kept his audience completely enthralled by his passionately and articulately expressed ideas, and intact well past the allotted time, despite the fact that they couldn't necessarily follow all the cryptic comments and references, nor connect the dots in real-time.

We had a number of great technical discussions, especially in those early years when computer scientists were still a small fraternity. I always came away from such conversations, which often lasted into the wee hours, simultaneously exhilarated, frustrated that we didn't have more time, and totally exhausted by the far higher bandwidth at which Alan operated. Indeed, I often felt like a graduate student sitting at the feet of a mentor, being given three times as many assignments—ranging from things to look into and think about, to *gedanken* experiments, to new system design ideas—as I could possibly handle. He clearly acted as, indeed was (and fortunately still is!), an intellectual agent provocateur, challenging orthodoxy and prodding me out of my overly rigid engineering mindset. Both he and Doug made me think far more deeply about levels of abstraction, and the fact that ultimately collections of soft components can be far more malleable than permitted by a more traditional top-down, hierarchically decomposed systems design.

I especially recall a graphics standards workshop held in Seillac, France in 1976. I went to Seillac prepared to argue for a procedure-based standard like

GPGS—the General Purpose Graphics System—to support interactive three-dimensional vector graphics that my research group and I had created during my sabbatical at Nijmegen University in the Netherlands, in collaboration with graphics experts at Delft Technical University and the GINO group from the Cambridge Computer-Aided Design Centre in the UK. Alan beat up on me for that “old-fashioned” procedure library point of view, and introduced me to a far deeper and more language-centered view of what graphics support could be; indeed, he taught me the rudiments of the then-developing object-oriented approach he was using with Smalltalk.

I was fascinated, but didn’t entirely grok that vision, and because I had no prior experience with it, advocated doing the first standard using the much less revolutionary, procedural approach that had already proven viable on a variety of graphics platforms. While such procedure libraries were created and eventually became both ANSI and ISO standards (GKS, PHIGS, and PHIGS++), they didn’t become widely used, and were driven out by other procedural libraries created as de facto standards (in particular SGI’s OpenGL and Microsoft’s DirectX).

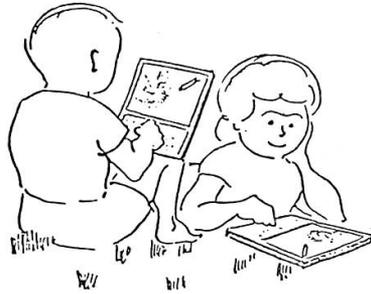
Today we see some of Alan’s “linguistic” approach in the increasingly wide-spread replacement of procedure-based libraries for fixed-function GPU pipelines by a far more flexible strategy that has influenced both hardware and software design based on shader languages driving reconfigurable, programmable GPU components typically configured not as a pipeline but as a computation graph.

Alan also greatly influenced me in the area of educational software, by the Dynabook idea and all the ideas floating around in the primordial soup seeded by that brilliant vision (or should we call it a quest?). I had become an academic because, on a whim, I taught the first course in computing for high school students and their teachers, during the summer of 1962 while still a graduate student at the University of Pennsylvania. Not only did I catch the teaching bug, but I also developed some educational software to help my students in that course learn how to program. Educational software then became a lifelong

interest. Seeing Sutherland's mind-blowing Sketchpad film a few years later caused me to switch fields from information retrieval to interactive graphics, especially in aid of educational simulation and visualization tools, as opposed to drill-and-kill Computer-Assisted Instruction. For example, my introductory graphics course uses a library of "exploratories," each illustrating a concept in computer graphics with an interactive visualization.

Thus, I was the ripest possible candidate to be converted to the Dynabook gospel, despite my profound skepticism about this science fiction idea, i.e., an affordable personal computer that one could carry around and that would have far more compute and graphics power for simulation, visualization and interaction than the multi-million dollar mainframes and display stations of the day. It articulated all the fundamental ideas about the importance of making interaction fluid and pleasant to empower users (from children on up) and reduce cognitive overhead at a time when design was almost exclusively focused on performance and functionality for professional tasks.

Today, I use my TabletPC to show Alan's Dynabook sketch with the kids interacting with fingers (pens?) in my talks on pen- and multi-touch computing, and try to get the younger generations to understand what an incredibly daring and expansive vision that was at a time when the field and we were young, and (digital) dinosaurs still roamed the earth.



So Alan, this is a heartfelt vote of appreciation for all you have done for our field: for creating tools and visions that enable the entire world to use not only computers but any gadget that has computational power (and that is most of them these days), but most of all for having challenged, indeed goaded me, over more than four decades to think more deeply, more broadly, more expansively. And the fact that most of these discussions happened in the context of good food and wine certainly was a major bonus for me!

Andries van Dam first met Alan at the monumental “Mother of All Demos”, Doug Engelbart’s demonstration of his pioneering NLS system at the 1968 Fall Joint Computer Conference, and they have maintained their spirited conversations since.

Andy graduated from Swarthmore College in 1960 and earned a Ph.D. from the University of Pennsylvania in 1966. He joined Brown in 1965 where he co-founded, and was first chairman of, the Computer Science Department. In 1967 he co-founded ACM SIGGRAPH, and was Director of the NSF Science and Technology Center in Graphics and Visualization from 1996–1998. He was Brown’s first Vice President for Research from 2002–2006. Andy is a member of the National Academy of Engineering, and a fellow of the American Academy of Arts & Sciences and of the American Association for the Advancement of Science.

He is currently the Thomas J. Watson Jnr. Professor of Technology and Education, and a Professor of Computer Science, at Brown University.

Raj Reddy

Alan Kay and the Creation of the Centre Mondial Informatique et Ressources Humaines in Paris

On the occasion of Alan's 70th Birthday, it is fitting to reflect on one other chapter of his many contributions, i.e., Alan's role in the creation of the *Centre Mondial Informatique et Ressources Humaines* in France by President François Mitterrand and Jean-Jacques Servan-Schreiber.

I was sitting in my office at CMU one Spring morning in 1980 when my friend Alan walked in the door unannounced and revealed news about an exciting new development: "There is this new venture being started in Paris that I think you should be involved in." He went on to explain about Jean-Jacques Servan-Schreiber, author of the best-selling book called *The American Challenge*, JJSS's friendship with President François Mitterrand, who had been elected President of France the previous year, and their belief that Information Technology would revolutionize the world. In particular Alan emphasized that based on the intuition of JJSS, that IT would become central to the development and enhancement of the capabilities of the "human resource," Mitterrand had approved the creation of the *Centre Mondial* in Paris.

Alan went on to say that Nicholas Negroponte and Seymour Papert of MIT and Terry Winograd of Stanford had already signed on to be present full time at the *Centre* along with Alan. Since one of the primary goals of

the *Centre* was to help the populations of the developing economies through the innovative uses of Information Technology, Alan thought I should also become involved. I explained that I had just agreed to become the Director of the Robotics Institute and there was no way I could move full-time to Paris, like the rest of them were planning to do. However, given my long-term interest in this issue, I agreed to become a commuting partner in this exciting enterprise.

I had known Alan since 1968 when he came to Stanford AI Labs as a postdoctoral fellow and had a high regard for his ideas on Dynabook and other innovations. At that time, I was an Assistant Professor in the Computer Science Department at Stanford and was part of the faculty at Stanford AI Labs. Alan was dynamic and full of ideas even then, and while at Stanford he worked on Architectures for LISP machines, among other things.

Alan subsequently moved to Xerox PARC as one of the Founding Researchers and I was a consultant for Xerox PARC during the 1970s. I was aware of his seminal ideas that went on to contribute to almost all the projects at PARC besides Smalltalk, an object-oriented programming language. Given our past associations and my background, it was natural for Alan to recruit me to become part of the *Centre Mondial* core founding team.

Jean-Jacques Servan-Schreiber and Sam Pizar's vision and Mitterrand's support were essential to producing a thriving enterprise. JJSS was a colorful figure in French politics. He was courageous enough to oppose the accepted policies of the government. His controversial book, *Lieutenant en Algérie* gave an account of the brutality of French repression in Algeria. Arguably, this book ultimately led to the decolonization of French Territories. His fascination and intuitive understanding of Information Technology led him to propose to President Mitterrand the establishment of the *Centre Mondial*. Sam Pizar, a holocaust survivor, author, and international lawyer was another influential voice at the *Centre Mondial*. His deep understanding of deprivation and loss of hope among the poor and the illiterate led to many of the policy directions of the *Centre Mondial*.

At the *Centre Mondial*, Alan and Seymour Papert launched a number of initiatives to introduce Microprocessor Systems into classrooms in countries like Ghana and Senegal. Nicholas Negroponte—as the Secretary General, providing the overall leadership—produced significant interest and excitement on both sides of the Atlantic for the *Centre*. I was the Chief Scientist for the venture.

The team drifted away over a four-year period, given the demands of their permanent jobs in the U.S. When Mitterrand was in the U.S. in 1984, he was gracious enough to visit CMU and award me the Legion of Honor Medal. I've always viewed it as a collective medal to all of us, especially to Alan without whom I would not have been involved.

For various political reasons, the *Centre Mondial* was ultimately disbanded after Mitterrand's presidency was over, but the ideas it generated were powerful and continue to be relevant for the people at the bottom of the pyramid. The legacy that emerged from the *Centre Mondial* experiment in the 1980s continues to resonate to this day. Negroponte unveiled the OLPC (One Laptop Per Child) hundred-dollar sub-notebook for k–12 students. This provided the impetus for many new low-cost laptops and netbooks. I continue my own quest to create models for educating gifted youth from below-poverty-level families, creating Digital Libraries of books available to anyone, anywhere and anytime, and to formulate health care solutions for developing countries.

Raj Reddy's friendship with Alan goes back to their meeting in 1968 when Alan went to the Stanford AI Lab as a postdoctoral fellow.

Raj was Assistant Professor of Computer Science at Stanford from 1966–69 before moving to Carnegie Mellon University as an Associate Professor of Computer Science. He served as the founding Director of the Robotics Institute from 1979–91 and as the Dean of School of Computer Science from 1991–99.

Dr. Reddy is currently the Mozah Bint Nasser University Professor of Computer Science and Robotics in the School of Computer Science at Carnegie Mellon University and also serves as the Chairman of Governing Council of IIT Hyderabad, and as the Chancellor and Chairman of the Governing Council of Rajiv Gandhi University of Knowledge Technologies in India.

Nicholas Negroponte

The Book in Dynabook?

We forgive many things when somebody reaches seventy, especially when that person has changed so many lives and taught us so many things. I forgive Alan for his paper books. It is ironic that the inventor of the Dynabook has been so wedded to them and here we are writing in that medium.

The paper book is dead, Alan. Long live the narrative.

Notice film is gone, but photographs are not. Vinyl and CDs are gone, but music is very much around. In fact people are undeniably taking more pictures and listening to more music than ever before. Namely the reach of, and appetite for, both photography and music are not only unrelated to the physical medium but enhanced by divorcing themselves from it. Literature is next.

E-book owners buy twice as many books as non-e-book owners. What's up and why now? Why not earlier?

The answer is a simple and classic tale of manufacturing. Cars are physical. We need raw materials, engineering, an assembly process, and a distribution system that includes both transportation and the careful control of inventory. By contrast, stories are not physical.

Why am I so sure the e-book issue is a matter of making bits versus atoms? Because I can imagine a near future when people have devices that have flexible displays, every bit (pun intended) as good as paper, even multiples of them bound with something that looks and feels like a spine and cover. While I

am not advocating this nostalgic format—a book with electronic pages, all of them blank—its existence would mean there is no need to make books. Instantaneously and wirelessly a book can be loaded (don't think about today's slow and unreliable networks). Since this kind of reloadable book with electronic pages can be imagined (and built), there remains absolutely no reason for us to assume a factory is necessary to make books with paper pages or for us to associate any one story with any one physical object.

The paper book, as a tightly coupled container of words and pictures, is increasingly a holdover from the past and will soon be as relevant to the written word as a sundial is to time. Yes, they both work under certain prescribed conditions, more charming than practical

Alan has eleven thousand books, more than anybody (but him) has read in a lifetime. But there will be no successor to them (or him).

I'll make the case in a number of ways, but start with the most obvious reason, which is rarely used and, in my opinion, is the most dramatic and irrefutable: the world's poorest and most remote kids. The manufactured book stunts learning, especially for those children. The last thing these children should have are physical books. They are too costly, too heavy, fall out-of-date and are sharable only in some common and limited physical space. Paper books also do not do well in damp, dirt, heat and rain. Not to mention that 320 textbooks require, on average, one tree and produce about 10,000 pounds of carbon dioxide in manufacturing and delivery. This makes no sense. Kids in the developing world should not be sent physical books.

The only way to provide books to the two billion children in the world is electronically. It is a simple bits and atoms story: you cannot feed children or clothe them with bits, but you can certainly educate and provide hope with these weightless, sizeless and mostly costless ones and zeros. Therein is the most practical reason that books cannot have paper pages. From here on in, it gets more subtle.

Right now, a paper book is undeniably more comfortable to read than a computer display. No nuance needed. It plainly is. Furthermore, the physical

book and the library, its institution of storage, are emblematic of and loved by a literate and informed society, as well as a gathering place and social venue.

To suggest books are dead is considered heathen or geeky or both. And while today's reading experience on an e-book is less than perfect, the basic concept is in the right direction. They never go out of print. There is no marginal cost in making more. They are delivered at the speed of light. They take no storage space. They can be automatically translated—badly today, perfectly in the future.

When people argue against digital books they tell me about touch, feel and smell. The smell of a book, I ask myself. What smell? Leather is a nice smell, but few books have that. The only smell I can remember is mildew after leaving a book in a damp basement. There is a certain romantic conceit taking place. In this sense, the emergence of digital books is like indoor plumbing. Some people argued against indoor plumbing (yes they did) on the force of the damage that it would do to the social fabric of a village (if the women did not meet daily at the river's edge). Likewise, people will argue that the death of books is the death of a literate society.

That said, before we discard books, we have a lot to learn from paper and bound volumes as we know them today. A lot. And Alan always said that. What book lovers are really talking about is the physical interface, from which you get a grasp (literally) of size and a sense of place (in the story). As you are reading, the amount you have read is in your left hand and how far you have to go is in your right. We all tend to remember the size and weight of books, the color of their jackets and, for that matter, where they are on our shelves, because our body was involved in all those actions (motor memory reinforcement it is called). We lose most of that with the little sliding bar (known as a fuducial) at the bottom of an e-book's screen.

Also true about paper: the white is white, the black is black and the resolution is so high we do not think of it. All soon to be true with e-books.

Some of what we can learn from paper books has transference. Some of it never will have, and society will have to get over them. This is not new.

Think of music. Older readers (most of us authors) will remember how much attention was spent on high-fidelity sound systems to reproduce music with utter verisimilitude (itself an old-fashioned word). Today kids care less. It is about mobility, access anywhere all the time. We have gone from hi-fi to Wi-Fi.

Other examples: telephone is not face-to-face conversation, cinema is not theater. But both have differences, advantages and deficiencies, compared with their predecessors, which at first they were mimicking. Remember we invented the word “hello” for telephones. The advantages of telephones are obviously tremendous, if only measured as an alternative to travel. In the case of motion pictures (a telling name), new skills and art forms arose around cinema and thereafter video. The same will happen with digital books.

As digital books unfold (so to speak) three transformations will occur that are very different from what we know in books and reading today. At the risk of being too cute, call them: wreading, righting and utterly new economic models.

Wreading. All things digital blur. Any formerly crisp boundary in the physical world becomes porous and fuzzy in the digital world by the mere fact that content is no longer captive to the container. Bits are bits and they commingle easily. While the ideas behind any piece of fiction or non-fiction are intangible (in the literal sense), rendered as ink on paper, they are (excuse the expression) carved into stone, literally immutable. Kept in the native form of bits, by contrast, the expression of an idea is not only fungible, but the reader can become a writer—what I am calling a wreader. A previously solitary experience can become a social experience (unlike this one, so far).

Righting. The wikipedia is an example. It is about intellectual property seen differently, as a collective process. The expansion and correcting of content is admittedly more germane to non-fiction than fiction. The point is that text with digital readers can evolve both in terms of facts and point of view on those facts. If you disagree, click [here](#) ... sorry about that. To date with physical books, the closest approximation we have is reading somebody’s annotations in the margin. Another modern example is commentary at the end of a digitally

published article; post your comment in this box. You might argue that the original narrative of such an article is often more considered, deliberate and refined than the comments that follow. True. But the volume (in the sense of loudness) and tone of the feedback is a form of self-correction of ideas, one that we have never had before.

Finally, the collapse of current business models behind print media of all kinds is starting to alarm more than just those in it and (in some cases) losing their jobs. What is so fascinating to me is that we are consuming more and more words. It is easy to dismiss many or most of them as noisy, senseless chit-chat, or the cheapest (both senses of that word) form of self-publishing. But boy, there are some beautiful blogs. *Orangette* and *20r3thingsiknow* are every bit as well-written or illustrated as any cookbook or commentary that I have ever read. By any stretch of the imagination these are not published and vetted only by their popularity, a Zagat versus Michelin process.

For these reasons the really new frontier, the explosive opportunity, is editorial or what I will call: “the expressions of point of view.” In fact, I deeply believe that the number of people who make a living from writing will skyrocket, not go the other direction. The industrial middleman will vanish. The people who help us determine what to read are ever important and new ones will arrive on the scene. The economics of reading and writing will be a cacophony, many small players, some big players, new players, but the total business will be huge.

There will be no single economic model behind it. Stop looking. Deliberation about which one—advertising, subscription, taxation and direct payments (large and small)—is close to irrelevant. Far more interesting (to me) is that we pay twice as much for entertainment and information (almost two hundred dollars per month) than we did ten years ago. I am also reminded of telephone companies complaining that many people were making free phone calls with Skype. But if you looked at the new market of Wi-Fi devices, computer subscriptions and short-term access fees, the total was far higher than the previous telephone business. Ditto books.

Think of it this way. Reading and writing are the primary human/computer interface, the way we interact with content. Many of us spend far more time reading and typing than we do speaking. That is a big change. We also look at far more images than we ever did before. We play with words much more than we did before. For example, we Google something and then Google the results, getting ourselves so carried away we sometimes do not remember what we were looking for in the first place. At one level this is a scatter-brained activity and we suddenly realize it is midnight. At another level, it is a natural curiosity amplified by interactive digital media, unimaginable in the paper age.

So if you are a publisher, sell differently or retire soon. If you are an author, don't just write for dead trees. If you are a reader, enjoy the unending opportunities, keeping in mind that free sex is different from love. Quality writing, clear thinking and good stories are to be loved and cherished. Physical books, as we know them today, needlessly limit these.

Alan, I hope Google is digitizing your books.

A graduate of MIT, Nicholas Negroponte was a pioneer in the field of computer-aided design and has been a member of the MIT faculty since 1966. He has known Alan since 1968.

At MIT he was Jerome B. Wiesner Professor of Media Technology, and the co-founder and director of the Media Laboratory. He is the author of the best-selling book Being Digital.

Nicholas is currently on leave from MIT, serving as the founding chairman of the One Laptop per Child (OLPC) non-profit association.

David P. Reed

Get the verbs right

Thinking about computing, like thinking about thinking, occasionally (very rarely) leads to ideas that are catalytic and transforming. Several times in my own career I've encountered an idea that started a reaction that *lives* in the same sense that a biological organism lives—a self-sustaining and energetic reaction that continues to move, to grow, to shape its environment, and, ultimately, to evolve in directions that I never expected.

My long-time friend and mentor, Alan Kay, seems to be a Johnny Appleseed of catalytic ideas. In this short essay, I'd like to share one of them whose genetic material continues to change my thinking, drive my work, and inspire my curiosity. There are many large-scale, famous contributions of his that I value, ranging from the Dynabook to the object-oriented computing framework around Smalltalk, but some of his ideas have had their impact by infecting other vectors, the thinking of others, both individually and in groups, who host them and bring them to maturity. I've been infected with a number of Alan's ideas—in this essay, I hope to be an effective vector for one that has yet to reach its tipping point.

As I recall, what Alan said, sometime around 1992, was: “The most important thing about object-oriented thinking is getting the verbs right.”

He said it in passing, in the middle of a talk, almost as an off-hand remark. I had heard Alan talk about object-oriented thinking, object-oriented design,

object-oriented programming many times over many years. I took note because I'd never heard it before. I *think* he said it because someone else in the Vanguard audience listening to him had challenged him by asking, “how do you define the right set of objects in a system?”¹

I immediately thought I understood why Alan's remark was true. So I jumped into the conversation, and congratulated Alan—“Exactly. It's got to be all about the verbs, because the verbs are where you get the conceptual efficiency of an architecture or design framework ... by selecting a small number of powerful verbs that apply to everything, you reduce complexity and you organize your ability to think.” Alan was clearly happy that I was enthusiastic, but I got the feeling that I'd missed something he had in mind. In the event, he went on talking about objects.

This idea, that it's all about the verbs, has stuck with me. Because whatever Alan meant at the time, it's a “pink plane” kind of idea. It wasn't until later, when working on extending Smalltalk during the Croquet project, that I again encountered the benefit of thinking about the verbs and not the nouns—about the methods and not the objects.

Most of the practitioners of object-oriented programming spend much of their time dealing with class hierarchies, deriving specialized classes from general classes. In languages like Java (the ones that Alan and I hate, because of their confusion of strong type checking with object orientation) specialized classes have more data, more methods, and more overrides compared to their parent classes. The specialized classes are more tightly bound to implementation tricks, and are also the classes that are actually used by programmers.

This way of thinking unconsciously has led most object-oriented programmers to be highly resistant to adding or modifying methods of the root classes—in whatever language, the root class is often called “Object.” The intuition, I guess, is that modifying the root classes risks breaking everything—

¹Vanguard is a research and advisory program in which Alan and I, along with a number of our dear friends, continue to participate, to understand how technology and ideas change the world of business and life. See <http://ttivanguard.com>

and breaking everything is the opposite of every principle of modularity or separation of concerns.

But Alan and his close associates urged me to be fearless in increasing the expressiveness of the root class by adding and enhancing methods, changing instance variables, etc. Something that seems to be a scary idea, especially bad for programming discipline. However, Smalltalk makes it relatively easy to do this, and languages like JavaScript (and Self, a prototype instance language) make it very easy.²

My discomfort about this led me to reflect more deeply, and I recognized Alan's earlier comment in a new form. Changing the methods of the root object has to do with getting the verbs right. Not the trivial, special case verbs, but the ones that are universal, heavy with meaning, and powerful. Adding a verb called **compress**: to the root class means defining a universal compression algorithm. To do so elegantly, one needs to think through what is general about compression, and what is type-specific or instance-specific about compression, *in a general and elegant way*.

So the highest form of conceptual efficiency about creating a program and an architecture is embodied in creating and modifying the meanings of the methods of the root classes or ultimate prototype objects. That is true, even if there is some variation about how the particular verb's meaning is implemented at each derived class or instance.

Is this what Alan meant? I concluded it didn't matter much—because it was the meaning I took from what he said. And that meaning was consistent both with Alan's earlier point, and with this practice, which was accepted within Alan's Smalltalkers. (I have since realized that students taught OOP from other traditions view such ideas as heresy. Strange, since Alan and the Smalltalkers around him are their intellectual forebears. Religions become sects, and disciples become heretics.)

²JavaScript/ECMAScript is one of my favorite object-oriented languages since LISP and Smalltalk. The book *JavaScript: The Good Parts* [34] is one of the few books that captures the flavor of good JavaScript programming, so for the skeptical, I recommend it.

So on my own journey, the idea that designing and evolution should push verbs back towards the root class or prototype became an important tool, and a re-confirmation of Alan's seed comment.

When you can do this shoving artfully, elegantly, and powerfully, it gives you amazing leverage. Ultimately it is about unifying many different actions into one verb that can stand for all such actions. Of course, generalizing about ideas is part of thinking—it's thinking about thinking about acting, and we've been doing this kind of recursive thinking since the time of the Greeks, who first wrote about thinking about thinking.

But verbs are most commonly viewed as second-class things. I think we've inherited this from mathematics. In abstract algebra, we talk about sets of objects that have certain properties, such as groups, rings, fields. These are nouns, and the sets are objects. We define functions as subsets of relations, which are sets of tuples of objects. From our language in explaining algebraic concepts, one would think that everything in algebra is composed of elements that can be referred to by nouns.

There is no action in most fields of mathematics, nothing in mathematics that seems like a verb. The only verb in mathematics is "to prove" or "to calculate," both of which are profoundly non-mathematical, since they involve human action in the first case, and human or mechanical action in the second case. (Mathematics as practiced by engineers and computing experts is an exception, but in fact, pure mathematicians tend to prefer to consider computing only when it is coded into noun-like objects that can participate in relations, sets, functions, etc.)

I don't want to get into an argument with mathematicians here, so let me say that my concern in this paper is not to criticize "blue plane" mathematics, which I love, and which has a power and wonder all its own. But let's stay on the "pink plane" and consider that it seems true that "verbs" are second-class citizens in the space of thinking about thinking, which includes design, architecture, and many aspects of practice.

Have you ever noticed that the English language has very few verbs compared to the number of nouns it contains? Why would a language evolve few verbs, and many nouns?

In contrast, if we view a typical computer program's subroutines as verbs, and its data types as nouns, there are many, many more verbs than nouns in any reasonably complicated programmed system. And if we view every line of code or the body of every loop or conditional as a verb, we have an even higher verb-type/noun ratio.

So of course, the problem in programming is that we can't seem to figure out how to invent the conceptually powerful verbs—while in English and most other languages, we have figured out how to gain conceptual economy in the population of verb-types. One might wonder if that is why humans are far more effective than AI robots in thinking about thinking about surviving and functioning in the world—we have evolved powerful tools in the set of verbs that we use.

While we are focused on the English language, before we return to more technical subjects, consider a couple more examples of the power of verbs. Reflect. (The only one-word sentences in the English language are verbs themselves, and notice that the previous sentence becomes self-referential, reflexive, and recursive in a single word, in the context of this paragraph.) The second example: observe the construction of most poetry. The poet must use figures of speech—metaphor, simile, metonymy, synecdoche, etc.—to generalize the poem's nouns' meaning, while the verbs generalize powerfully and without effort, easily amplifying the trickery needed to extend the meaning of the nouns.

Much of my work with computers has focused on building systems that must coordinate actions across time and space, often with individual users and groups of users. These are *active* systems. Their whole purpose is about continuous interaction, continuity, learning, etc. I have come, over many years, to think that designing such systems involves thinking about how they *act* in concert with humans.

I think that requires a verb-centric way of thinking. Such systems' actions can't be described by a still picture, or a statement about an "output," except by defining some abstract representation or picture that somehow captures behavior in time. Even then, however, the resulting abstract visualization is unsatisfactory to me, and probably to all others who try to represent a design of such systems. Typical attempts to describe such systems involve artifacts like mappings from sequences of inputs arriving over time to sequences of outputs produced over time. (Communication protocols often show sequences of messages as linear histories of states of the endpoints linked by diagonal connections representing message transmissions.)

To avoid treating verbs as first-class meanings (as verbs), describing or defining system-wide actions is often done by treating them as nouns. They become "acts" that are instances of templates (or recipes) composed of parts, which are also acts. There's a "pun" here, because "to act" and "an act" are the same word, even though one is a verb and one is a noun.

Yet a great deal is lost in treating action as a noun. It's similar to what is lost when one thinks about a "river" as a "set of water molecules" or an amoeba as a "set of molecules". At first, the difference seems trivial. After all, isn't a river just a set of molecules?

Well, the saying is "you can't step in the same river twice." That is not true of a set of molecules—a true scientist would recognize that what we call a river is not the set of water molecules, nor is it the riverbed, nor is it a connected region on a map. It is something that is quite different, and the difference is obvious to every human who walks up to and touches the river.

Similarly, an amoeba is not the set of molecules that makes up the amoeba at any particular time. Yet these are unitary concepts that we can study, explore, understand, etc.

It is useful sometimes to model a river as a set of molecules, but it does not tell us everything about the river. But worse, it does not tell us enough to *design* a better river. Reflect. What would help us design a better river, or a river never seen before? We'd need to understand *action* and *behavior*. In fact,

we'd have to invoke a range of knowledge that touches on nearly every part of science and our experience.

Now, come back to the question of designing systems that coordinate activities with people and other systems. What does “coordinate” mean? Well, it requires a notion of “verbiness” to even get started—it requires a notion of what it means to *act together*. A part of the meaning expressed by many verbs is causality—knowing what causes what is part of what makes a verb different from a noun. But also, real verbs and the actions they describe are not “sequences of actions,” but often better described as continuous coordinated connections. It is quite easy to understand what it means “to dance,” but to write a computer program that allows a robot to dance with other robots and computers involves describing something other than a series of poses, and even something other than a sequence of reactions to inputs.

So the problem in designing systems that interact with other systems, and with humans, is critically a matter of “getting the verbs right.” How can we do that if we view verbs as second-class elements, ones that have no power and no central role?

I would claim that you cannot do it. Yet the tools we build, and the way we teach our students to build systems even more so, are focused away from this challenge.

We don't know how to deal with verbs, and we don't deal with verbs. I hope by now you understand that I am not claiming that we don't use verbs in programming or system design. We do use them. But we use weak verbs only, we get scared whenever someone proposes to invent or to generalize our verbs, and we try almost anything to avoid talking about much of what is contained in the notion of verb, whether it is causality, generality, etc.

In contrast, even the most popular forms of programming are focused on objects and their noun-like qualities, on states and state transitions, on formal mathematical systems and the formal notions of sets, relations, functions, etc. All focused on understanding nouns and the constellations of concepts that surround them, such as properties, components, relationships, and so on.

What to do? Well, I often think back to Alan's inspirations in creating Smalltalk. The comment that he has made often is that he was inspired by his training in biology, and by his love of music, especially musical performance. From biology, I think he understood that to describe living things, in all their chemical and enzymatic glory, you cannot merely look at them as physical elements. Instead you must think of their activity, and not their parts, as being the essential thing. The essence is contained in their verbs and not their nouns. As I have mentioned earlier, it is not the specific instances of molecules or the specific contents of their cells, or the specific enzymes or even the specific genes that make up a living being. (A being is what it is by means of its be-ing—that is, it is a verb, not a noun...)

In *The Triple Helix: Genes, Organism, Environment* [36]—a book that transformed my thinking almost as much as Alan says *The LISP 1.5 Programmer's Manual* [37] transformed his, and for the same reason—Richard Lewontin makes the case that life is not at all determined by DNA replication. Instead, he contends that life is a *recursively intertwined* process that inseparably combines genetic inheritance, parental shaping of the environment, and the organism's mutual interaction with its environment. In Lewontin's view, it makes no sense to make DNA replication primary while subsuming all of the other activities defining or creating life to second-class roles. In this, he focuses on activity, rather than a reductionistic approach focused on genes creating genes, the simplistic idea promoted by Dawkins in *The Selfish Gene* [35].

I would argue that computer science and design of systems must begin to think about “getting the verbs right” in two ways. One, in defining the key verbs and getting comfortable with what it means to think about verbs rather than nouns alone. I hope I have given some hints about what I think that means. Two, learning how to think and design in verbs, while teaching each other and our students to do so. This is likely to require much experimentation, since it is an open-ended problem.

I don't know if any of this is the seed that Alan meant to plant when I first heard him say that the key design problem is “getting the verbs right.” I know

that Alan's original inspiration for Smalltalk was Simula—a programming language for simulation of activities and modeling of systems. Alan and I share a passion for using computers as tools of expression, modeling, learning, communication and design. None of these are about creating “nouns” or systems of “nouns”—instead we focus on creating processes that will live and evolve beyond our time.

With that, I hope I have served at least part of my role as vector for this infection. I hope I have infected at least one, and I hope more, readers with a small flame that will continue to burn in their thinking—regarding “getting the verbs right.”

David P. Reed met Alan sometime around 1977 during a visit to Xerox PARC. Of his mentors (including Alan) he says he inherited “their remarkable common fascination with the interplay between concepts and mechanism, principles and pragmatics.”

As a graduate student David contributed to the design of the Internet protocol suite now called TCP/IP and worked on novel approaches to concurrency control and coordination in distributed systems. He was awarded a Ph.D. in in Computer Science and Engineering by MIT in 1978.

David spent five years as a faculty member at MIT before leaving to serve as vice president of R&D and chief scientist at Software Arts and later at Lotus Development Corporation. In 1994 he joined Interval Research, remaining there for four years before becoming a technology consultant in 1996.

Among several prizes and awards David's favorite is the IP₃ award from Public Knowledge, for his contributions to creating the architectural principles of an information commons.

David currently spends part of his time as Adjunct Professor of Media Arts and Sciences at the MIT Media Lab, devoting the rest of his time to consulting, family, and inventing new technologies and principles. He calls his most recent focus the “Third Cloud”—an architectural framework for programming and controlling the rich computing and informational contexts centered on us, as individuals and groups, as we experience the world around us. Of this he says: “Just as the Dynabook embodied the ‘personal,’ the Third Cloud disembodies it.”

Chuck Thacker

A Tiny Computer

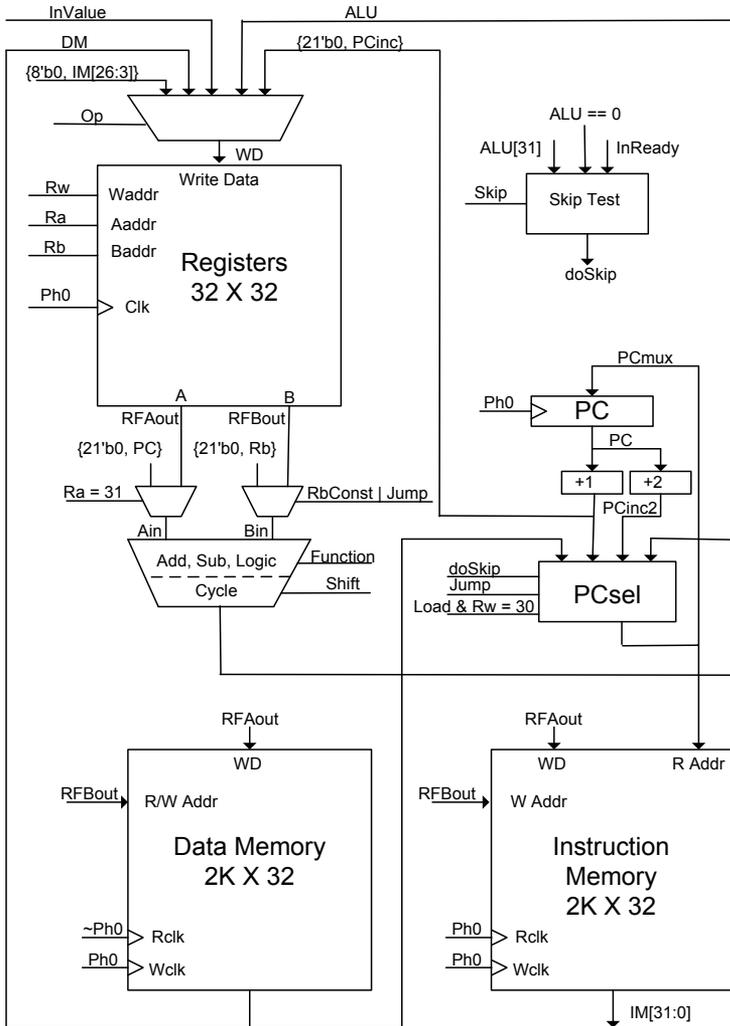
In late 2007, Alan Kay said to me: “I’d like to show junior and senior high school kids the simplest non-tricky architecture in which simple gates and flip-flops manifest a programmable computer.”

Alan posed a couple of other desiderata, primarily that the computer needed to demonstrate fundamental principles, but should be capable of running real programs produced by a compiler. This introduces some tension into the design, since simplicity and performance are sometimes in conflict.

This sounded like an interesting challenge, so I designed the machine shown below.

Implementation

Although it is impractical today to build a working computer with a “handful of gates and flip-flops,” it seemed quite reasonable to implement it with an FPGA (Field Programmable Gate Array). Modern FPGAs have enormous amounts of logic, as well as a number of specialized “hard macros” such as RAMs. The basic logic is provided by lookup tables (LUTs) that can be configured to produce any Boolean function of their inputs. Each LUT is accompanied by a flip-flop that may be used to construct registers. All wiring between the LUTs and registers, as well as the functions done by the LUTs, is configurable by a



“bit stream” file, which is loaded into the chip at initialization. The Spartan-3 part used for the TC had 4-input LUTs. More modern FPGAs have six-input LUTs.

Xilinx Corporation sells evaluation boards for about \$150 that include an FPGA, and some auxiliary components for connecting the chip to real-world

devices and to a PC that runs the design tools, which are free to experimenters. This was the approach I selected. The evaluation board was the Spartan-3E Starter Kit. There are similar evaluation boards available today that contain much larger FPGAs.

These days, hardware is designed by doing programming. It's just like writing a program to calculate sums, except that the output of the program is a specification of what the system being designed should do, rather than the immediate result. When you write " $x \leftarrow A+B$ " you are not asking for the value of x , you're asking for an adder, which can give you the value of x for any A and any B . The Verilog synthesizer politely produces an adder.

Although the machine was designed primarily for teaching, it may have other uses. It is small, fast, and has 32-bit instructions. This may make it competitive with more complex FPGA CPUs. The section below on extensions describes some possibilities for making it a "real" computer, albeit a fairly limited one. The evaluation board provides a number of interesting components that could also be used to extend the design.

I chose a "Harvard" architecture with separate data and instruction memories, because it is possible to access the data and instruction memories simultaneously. It is still possible to write self-modifying code (although this is usually considered a bad idea), since writes into both memories are supported.

At the time the design was done, the Spartan-3E was the most cost-effective FPGA made by Xilinx, but it was implemented in a 90nm silicon process which was already obsolete by one generation. Today's FPGAs are implemented in a 45nm process, so they are both faster and less expensive. Xilinx FPGAs have an interesting feature that contributes to the small size of the overall design—a dual-ported static RAM with 1024 words of 18 bits. This RAM is used for the data and instruction memories (four for each memory). The register file uses memories built from the FPGA's lookup tables.

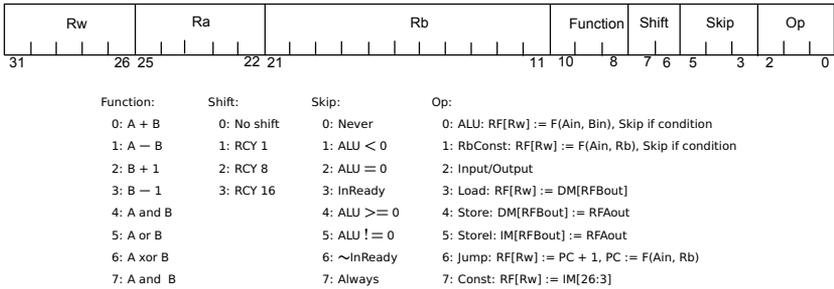
The machine has 32-bit data paths. Most "tiny" computers are 8 or 16 bits wide, but they were designed originally in an era in which silicon was very expensive and package pins were scarce. Today, neither consideration applies.

The design of the instruction set for the machine was determined primarily by the instruction and data path widths. It is a RISC design, since that seemed to be the simplest arrangement from a conceptual standpoint, and it is important to be able to explain the operation clearly.

Although the memories are wide, they are relatively small, containing only 2048 locations. The section on extensions discusses some ways to get around this limit. For pedagogical purposes, a small memory seemed adequate.

The memory is word-addressed, and all transfers involve full words. Byte addressing is a complexity that was introduced into computers for a number of reasons that are less relevant today than they were thirty years ago. There is very limited support for byte-oriented operations.

The primary discrete register in the design is the program counter (PC). PC is currently only 11 bits wide, but it could easily expand. The memories used for IM and DM register their addresses and input data, so we don't need to provide these registers. We do need PC, since there is no external access to the internal IM read address register. PC is a copy of this register.



The instruction set is very simple and regular. All instructions have the same format. Most operations use three register addresses. The ALU instructions do $RF[Rw] := \text{function}(RF[Ra], RF[Rb])$. This seemed easier to explain than a machine that used only one or two register addresses per instruction. Note that the Rb field is 11 bits wide, rather than the 5 bits needed to address RF. This field is used in Jump instructions to provide a constant value that

can address the entire instruction memory or provide an 11-bit offset. The `RbConst` instruction uses this field, instead of `RFBout`, as the B input of the ALU.

Writes to `RF[0]` are discarded. Because the registers contain zero when the FPGA is configured, `r0` will remain zero, and is both a source of zero and a destination in which to discard unneeded ALU results.

Instructions with `Ra = 31` replace the ALU's A input with the (zero-extended) `PC + 1`. This is provided for subroutine linkage.

The `Const` instruction loads `RF[Rw]` with a 24-bit constant from the instruction. The `In` and `ReadDM` instructions load `RF[Rw]` with data from an input port or from the data memory. The `Jump` instruction loads `RF[Rw]` with the incremented program counter. This provides a subroutine return address. `Jump` also uses `Rb` as the `Bin` input to the ALU rather than the register file B output, which provides additional flexibility in specifying the target address. In addition, if `Rw = 31` in a `Jump` instruction, the PC is loaded with `DM[10:0]` rather than the ALU output. This saves one instruction in a subroutine return that uses a stack, since it is not necessary to put the return link in a register before jumping to it. Leaf routines (those that do not call other routines) don't need to use a stack, and return by jumping to the `Link` register.

The ALU, `RbConst`, and input-output instructions conditionally skip the next instruction if the condition selected by the `Skip` field is true.

The `Store` and `StoreI` instructions write the A output of the RF into the memory location selected by the B output. The `Load` instruction writes `RF[Rw]` with `DM[RF[Rb]]`.

The machine executes instructions in a single clock cycle, fetching an instruction from the IM location addressed by PC, accessing RF, and doing whatever operation the instruction specifies. At the end of the instruction, the results are written to the register file, and the next instruction is fetched from IM. This is unlike essentially all modern computers, which use pipelining to improve performance, but it is much easier to explain. Pipelining things would make it faster.

Using skips and unconditional jumps is quite powerful and succinct. It was first employed in the Data General Nova, a simple machine that still has a lot to teach us about computer architecture, since it was arguably the first commercial RISC machine. The major differences between the Tiny Computer and the Nova are:

- There are more registers (32 vs. 4).
- There are three register select fields instead of two.
- The Function field has different meanings.
- The Nova's Carry field has been eliminated.
- The Skip field is different.
- There is no "no load" bit.
- The Nova had a 16-bit data path.

There is no call stack. Programs that need a stack must construct it themselves. The example program below shows how this is done.

The input-output facilities are primitive. Devices are accessed using the IO instruction, which can send RF[Ra] to a device or load RF[Rw] with data from a device. Devices are selected using Rb. A selected device may assert a signal, InReady, that may be tested by a branch condition. There is no support for interrupts, since these add complexity and are difficult to explain.

Size and Speed

In Spartan-3E technology, the machine occupies about 900 4-input LUTs, of which 256 are used for the register file, and 8 block RAMs. This is about 10% of the logic the device contains (although it is 50% of the block RAMs), so there is a lot of room for additional features. It runs at 40 MHz, which is adequate to run nontrivial programs. The Verilog program describing the entire design is a few pages long (see appendix on page 135). As an experiment, I recently built the design for a more modern FPGA, the smallest that Xilinx now sells. It occupies less than 10% of the available logic, demonstrating that Moore's law has not run out yet.

Software

Programs for the TC are written in assembly language, and assembled using the TCasm program.

This program takes a single source file and produces a .coe file, which the Xilinx tools place into the portion of the bitstream file that specifies the contents of the instruction memory. This entails rebuilding the entire FPGA whenever a program is changed, but since this takes at most a couple of minutes, it has not proven to be a problem in practice.

The assembler is very simple, doing most of its work by looking up textual tokens from the source in a symbol table, and emitting the instruction fragment into the current instruction. The symbol table is populated with definition directives placed at the start of the source file, and by symbols representing labeled statements when these are encountered during assembly. An instruction must occupy a single line of source text, and must be terminated with a semicolon. Any remaining text on the line is ignored. Case is significant. Numbers may be decimal or hex ($0xnnn$). Tokens are separated by white space.

The assembler is two-pass. During the first pass, it skips tokens that are undefined. Isolated strings that resolve to fields are placed in `currentValue` with the indicated offset. Isolated numbers are also placed in `currentValue`, using an offset of 3.

Symbol table entries have a `key` and `contents` which is a field containing a `value` and an `offset` in the instruction (the bit position into which the value should be placed). Other built-in reserved words are treated as follows:

field *string value offset* declares a symbol. The symbol has `key=string` and `contents={value, offset}`.

rfref *string number* defines three symbols, with keys that are concatenations of “a,” “b” and “w” with *string*. The contents’ values are *number*, and their offsets are `aoff`, `boff` and `woff`, respectively.

(These constants must be defined as fields before the `rfréf.`) For the TC, `aoff=22`, `boff=10`, and `woff=27`. The resulting symbols are used in later instructions to name the register file's a, b, and w instruction fields for register *number*.

mem *number* or **mem *string*** (where *string* evaluates to a number) makes `M[number]` the current memory. Memories are numbered 0 to 2. Usually the *string* form will be used, after a preceding field definition of the memory name. Token processing continues after the `mem` and its operand.

loc *number* or **loc *string*** (where *string* must evaluate to a number) sets the current location in the current memory.

string: provides a way to label statements. A symbol is defined with `key=string` and `contents={currentLocation, 11}`. This is a Rb constant. Execution of a `Jump loc` instruction substitutes this (11-bit) constant for `RF[Rb]` as the b input to the ALU, providing that location `loc` has been labeled. The default value for Ra is 0, and the default function is `add`, so the right thing happens.

When token processing for a line is finished, if any field has been set in `currentValue` then the value is placed into the current location in the current memory, and the current location in that memory is incremented; `currentValue` is then cleared and scanning resumes at the start of the next line.

Because the assembler syntax is so loose, it is easy to write programs that don't work. The usual error is to use the incorrect port variant for named registers. This would usually be worrisome, but we don't expect to write very much assembly code for the machine. To make it a more useful teaching tool, a better assembler would be needed. A simulator would also be useful to try out programs before committing them to the hardware.

The example below shows a very small program, with the machine definition preceding the source code.

```
field aoff 22 0; Field offsets for rref.
field boff 11 0; These must be defined and must not change.
field woff 27 0;

field instruction 0 0; name for instruction memory
field rf 1 0;          name for register file
field data 2 0;      name for data memory

field := 0 0; noise word
field PC 0 0; noise word

field + 0 8; the "plus" function
field - 1 8; the "minus" function
field ++ 2 8; the "Rb + 1" function
field -- 3 8; the "Rb - 1" function
field & 4 8; the "and" function
field | 5 8; the "or" function
field ^ 6 8; the "xor" function
field &- 7 8; the "and not" function

field rcyl 1 6;
field rcy8 2 6;
field rcy16 3 6;

field skn 1 3; skip if ALU < 0
field skz 2 3; skip if ALU = 0
field ski 3 3; skip if InReady
field skge 4 3; skip if ALU >= 0
field sknz 5 3; skip if ALU != 0
field skni 6 3; skip if ~InReady
field skp 7 3; skip always

field RbConst 1 0; Opcodes
field IO 2 0;
field Load 3 0;
field Store 4 0;
field StoreIM 5 0;
field Jump 6 0;
field Call 6 0; as Jump but clarifies intent. Will specify Rw for the Link.
field Const 7 0;
```

```
mem instruction loc 1; Make location 1 of instruction memory current.

rfref Trash 0; r0 used for both the trashcan and the source of zero
rfref Zero 0;
rfref Link 1; subroutine linkage register
rfref Stkp 30; stack pointer
rfref PC 31;

; Rb[0] = 0 is In, Rb[0] = 1 is Out
field readRS232Rx 0 11;
field readRS232Tx 2 11;
field writeRS232Tx 3 11;
field writeLEDs 5 11;

; Registers
rfref DelayCount 2; count this register down to delay
rfref OutValue 3;

start: wStkp := Const 0x7ff; last location in DM
blink: wDelayCount := Const 0xfffff;
      Jump delay wLink; subroutine call
      IO writeLEDs aOutValue;
      wOutValue := bOutValue ++;
      Jump blink;

delay: Store aLink wStkp := bStkp -- ;
delay1: wDelayCount := bDelayCount -- skz;
      Jump delay1;

ret: wStkp := bStkp ++ ;
     Load wPC bStkp;

End
```

This program is not very interesting. We have written a few other programs for the system, including a debugging program that communicates with its user using the RS-232 interface. We have not gone as far as providing a compiler for the architecture. Perhaps this should be left as an exercise for the reader.

Extensions

The limited size of the data and instruction memories is the main thing that makes this computer uncompetitive. This could be mitigated by using the memories as caches rather than RAM. The 2 kB BRAM holds 256 blocks of 8 words, which is the usual transfer size for dynamic RAM. We would need to provide I and D tag stores, but this wouldn't be very difficult. The evaluation board contains a 16 MB DDR synchronous dynamic RAM, which could be employed as main storage.

Successors and Conclusions

The Tiny Computer was designed at a time when an interest in using FPGAs as platforms for computer architecture research was growing. In our laboratory, we designed and implemented an example of such a platform, the “BEE3” (Berkeley Emulation Engine version 3). This system contains four Virtex 5 FPGAs, 64 GB of DDR2 memory and a variety of input-output devices. The design was licensed to BEE cube Corporation,¹ which now produces and distributes the systems to researchers throughout the world. Using the BEE3, it is possible for a small team to design and implement serious systems. It has been used by researchers in a number of universities to build systems that are used to explore advanced computer architectures.²

While Alan's “handful of gates and flip-flops” was over-optimistic, the Tiny Computer demonstrated that it is possible to build nontrivial systems. Thirty years ago it was common for researchers to build their own computers, program them, and use them in their daily work. The high cost of building silicon chips cut this line of research short. With FPGAs we have seen a resurgence of this sort of activity. In our laboratory we have built a computer system that is used to explore “many-core” architectures, in which a large number of very small processors can be used to build systems of considerable complexity and

¹<http://www.beecube.com>

²Some examples can be found at: <http://www.ramp.eecs.berkeley.edu>

power. The design can be targeted to the BEE3 or to a much less expensive Xilinx development board (XUPv5). On this board, it is possible to build a system with 16 processor cores, a Gigabit Ethernet interface and a DDR2 memory controller. We are using this system as a tool to support our research in computer architecture.

The advent of low-cost FPGA-based boards, coupled with the availability of programming tools to make use of them, makes it possible for students to easily create designs of their own. Perhaps the availability of these devices will enable innovation not only in computer architecture, but in other areas of digital system design. Given the large amount of logic available in modern FPGAs, the high cost of implementing “real” silicon chips need no longer be a barrier to innovation in these areas.

The first computer that I designed that Alan Kay used seriously was the Alto (1973). Alto had a slower clock rate than the TC (170 ns vs. 25 ns). This was the rate at which the machine executed its *micro*-instructions. Real programs, written in real languages such as BCPL and Smalltalk, required several microinstructions to execute each instruction. The Alto had 128 kB of memory and a 2.5 MB disk. The single RAM chip on the least expensive Xilinx development board (in 2007) had six times this amount of storage. The Alto cost \$12,000, at a time when \$12,000 was a *lot* of money. The Tiny Computer hardware costs \$125.

Hardware technology has certainly advanced. Has software? I am still using a program to write this paper that is the lineal descendant of one of the first programs for the Alto: the Bravo text editor. It provided WYSIWYG (what you see is what you get) editing. The Alto had a network (Ethernet), and the first laser printers. It provided a user experience that wasn't much different from the system I'm using today, although today most people have computers, which is quite different.

So we still have a long way to go. Perhaps Alan's most recent attempt to “redefine the personal computer” will help us move forward.

Appendix: Tiny Computer Verilog description

```
'timescale 1ns / 1ps

module TinyComp(
  input ClockIn,    //50 Mhz board clock
  input Reset,     //High true (BTN_SOUTH)
  output [7:0] LED,
  input RxD,
  output TxD
);

wire doSkip;
wire [31:00] WD; //write data to the register file
wire [31:00] RFAout; //register file port A read data
wire [31:00] RFBout; //register file port B read data
reg [10:0] PC;
wire [10:0] PCinc, PCinc2, PCmux;
wire [31:00] ALU;
wire [31:00] ALUresult;
wire [31:00] DM; //the Data memory (1K x 32) output
wire [31:00] IM; //the Instruction memory (1K x 32) output
wire Ph0; //the (buffered) clock
wire Ph0x;
wire testClock;

wire [2:0] Opcode;
wire [4:0] Ra, Rw;
wire [10:0] Rb;
wire Normal, RbConst, IO, Load, Store, StoreI, Jump; //Opcode decodes
wire [2:0] Skip;
wire Skn, Skz, Ski, Skge, Sknz, Skni, Skp;
wire [1:0] Rcy;
wire NoCycle, Rcy1, Rcy8;
wire [2:0] Funct;
wire AplusB, AminusB, Bplus1, Bminus1, AandB, AorB, AxorB;
wire WriteRF;

wire [31:0] Ain, Bin; //ALU inputs

reg [25:0] testCount;
wire InReady;
wire [31:0] InValue;
reg [7:0] LEDs;

//----- The I/O devices -----

wire [3:0] IOaddr; //16 IO devices for now.
wire readRX;
wire charReady;
wire [7:0] RXchar;
wire writeLED;
wire writeTX;
wire TXempty;
wire [7:0] TXchar;

assign IOaddr = Rb[4:1]; //device addresses are constants.
assign InReady = ~Rb[0] &
```

```
((IOaddr == 0) & charReady) | //read RS232 RX
(IOaddr == 1) & TXempty)); //read RS232 TX

assign InValue = (IOaddr == 0) ? {24'b0, RXchar} : 32'b0;
assign TXchar = RFAout[7:0];
assign readRX = ~Rb[0] & (IOaddr == 0) & IO;

assign writeTX = Rb[0] & (IOaddr == 1) & IO;
assign writeLED = Rb[0] & (IOaddr == 2) & IO;

always @(posedge Ph0) if(writeLED) LEDs <= RFAout[7:0];
assign LED = LEDs;

rs232 user(
    .clock(Ph0),
    .reset(Reset),
    .readRX(readRX),
    .charReady(charReady),
    .RXchar(RXchar),

    .writeTX(writeTX),
    .TXempty(TXempty),
    .TXchar(TXchar),
    .TxD(TxD),
    .RxD(RxD)
);

//----- The CPU -----

always @(posedge testClock)
    if(Reset) testCount <= 0;
    else testCount <= testCount + 1;

always @(posedge Ph0)
    if(Reset) PC <= 0;
    else PC <= PCmux;

//Opcode fields
assign Rw = IM[31:27];
assign Ra = IM[26:22];
assign Rb = IM[21:11]; //larger than needed to address RF.
assign Funct = IM[10:8];
assign Rcy = IM[7:6];
assign Skip = IM[5:3];
assign Opcode = IM[2:0];

//Opcodes
assign Normal = Opcode == 0;
assign RbConst = Opcode == 1;
assign IO = Opcode == 2;
assign Load = Opcode == 3;
assign Store = Opcode == 4;
assign StoreI = Opcode == 5;
assign Jump = Opcode == 6;
//assign Const = Opcode == 7;

//Skips
assign Skn = (Skip == 1);
assign Skz = (Skip == 2);
```

```
assign Ski = (Skip == 3);
assign Skge = (Skip == 4);
assign Sknz = (Skip == 5);
assign Skni = (Skip == 6);
assign Skp = (Skip == 7);

//Cyclic shifts
assign NoCycle = (Rcy == 0);
assign Rcy1 = (Rcy == 1);
assign Rcy8 = (Rcy == 2);

//ALU functions
assign AplusB = Funct == 0;
assign AminusB = Funct == 1;
assign Bplus1 = Funct == 2;
assign Bminus1 = Funct == 3;
assign AandB = Funct == 4;
assign AorB = Funct == 5;
assign AxorB = Funct == 6;

//The Skip Tester.
assign doSkip =
  (Normal | RbConst | IO) & //Opcode can skip
  (
    (Skn & ALU[31]) |
    (Skz & (ALU == 0)) |
    (Ski & InReady) |
    (Skge & ~ALU[31]) |
    (Sknz & (ALU != 0)) |
    (Skni & ~InReady) |
    Skp
  );

//The PC-related signals
assign PCinc = PC + 1;
assign PCinc2 = PC + 2;
assign PCmux =
  Jump ? ALU[10:0] :
  (Load & (Rw == 31)) ? DM[10:0] : //subroutine return
  doSkip ? PCinc2 :
  PCinc;

//Instantiate the WD multiplexer.
assign WD =
  (Normal | RbConst | Store | StoreI) ? ALU :
  IO ? InValue:
  Load ? DM:
  Jump ? {21'b0, PCinc}:
  {8'b0, IM[26:3]}; // 24-bit constant

assign WriteRF = (Rw != 0); //Writes to r0 are discarded.

//The input multiplexers for the ALU inputs
assign Ain = (Ra == 31) ? {21'b0, PC} : RFAout;

assign Bin = ( RbConst | Jump ) ? {21'b0, Rb} : RFBout;

//Instantiate the ALU: An adder/subtractor followed by a shifter
assign ALUresult =
```

```
AplusB ? Ain + Bin :
    AminusB ? Ain - Bin :
    Bplus1 ? Bin + 1 :
    Bminus1 ? Bin - 1 :
AandB ? Ain & Bin :
    AorB ? Ain | Bin :
    AxorB ? Ain ^ Bin :
    Ain & ~Bin; //A and not B

assign ALU =
NoCycle ? ALUresult :
Rcy1 ? {ALUresult[0], ALUresult[31:1]} :
Rcy8 ? {ALUresult[7:0], ALUresult[31:8]} :
{ALUresult[15:0], ALUresult[31:16]};

//Instantiate the instruction memory. A simple dual-port RAM.
ramx im(
//the write port
    .clkA(Ph0),
    .addra(RFBout[10:0]),
    .wea(StoreI),
    .dina(RFAout),

//the read port
    .clkb(Ph0),
    .addrb(PCmux),
    .doutb(IM)
);

//Instantiate the data memory. A simple dual-port RAM.
ramw dm(
//the write port
    .clkA(Ph0),
    .addra(RFBout[10:0]),
    .wea(Store),
    .dina(RFAout),

//the read port
    .clkb(~Ph0), //use ~Ph0 since we can't read DM until the address (from IM) is ready.
    .addrb(RFBout[10:0]),
    .doutb(DM) //the read port
);

//Instantiate the register file. This has three independent addresses, so two RAMs are needed.
ramz rFA(
    .a(Rw),
    .d(WD), //write port
    .dpra(Ra),
    .clk(Ph0),
    .we(WriteRF),
    .dpo(RFAout) //read port
);

ramz rFB(
    .a(Rw),
    .d(WD),
    .dpra(Rb[4:0]),
    .clk(Ph0),
```

```
.we(WriteRF),
.dpo(RFBout) //read port
);

BUGF ph1Buf(.I(Ph0x),.0(testClock));
BUGF ph0Buf(.I(Ph0x), .0(Ph0)); //Global clock buffer

//The design won't actually run at the 50MHz supplied board clock,
//so we use a Digital Clock Manager block to make Ph0 = 40 MHz.
//This can be ignored, unless you want to change the speed of the design.
DCM_SP #(
.CLKDV_DIVIDE(2.0),
.CLKFX_DIVIDE(10),
.CLKFX_MULTIPLY(8),
.CLKIN_DIVIDE_BY_2("FALSE"),
.CLKIN_PERIOD(20.0),
.CLKOUT_PHASE_SHIFT("NONE"),
.CLK_FEEDBACK("1X"),
.DESKEN_ADJUST("SYSTEM_SYNCHRONOUS"),
.DLL_FREQUENCY_MODE("LOW"),
.DUTY_CYCLE_CORRECTION("TRUE"),
.PHASE_SHIFT(0),
.STARTUP_WAIT("FALSE")
) TCdcm (
.CLK0(),
.CLK180(),
.CLK270(),
.CLK2X(),
.CLK2X180(),
.CLK90(),
.CLKDV(),
.CLKFX(Ph0x),
.CLKFX180(),
.LOCKED(),
.PSDONE(),
.STATUS(),
.CLKFB(),
.CLKIN(ClockIn),
.PSCLK(1'b0),
.PSEN(1'b0),
.PSINCDEC(1'b0),
.RST(Reset)
);

endmodule
```

Chuck Thacker received a B.Sc. in Physics from the University of California at Berkeley in 1968.

He remained at Berkeley with the University's project Genie until leaving for Xerox PARC in 1970. It was at PARC that he met and worked with Alan. Chuck was chief designer of the Alto and co-inventor of the Ethernet local area network.

After some time at Digital Equipment Corporation's Systems Research Center, he joined Microsoft to help establish their laboratory in Cambridge, England. On returning to the U.S. in 1999 he joined the newly-formed Tablet PC group and managed the design of the first prototype machines.

In 2004, Chuck, Butler Lampson, Robert Taylor and Alan were awarded the National Academy of Engineering's Charles Stark Draper prize for the development of the first networked personal computers.

Chuck is currently setting up a computer architecture research group at Microsoft Research in Silicon Valley.

Douglas B. Lenat

The K Factor

Factoring in General

Our universe is shaped by several physical forces (strong, weak, electromagnetic, gravity). Here on Earth, our economies and technologies are shaped by several types of terrestrial energy sources (oil, solar, geothermal, etc.). Between our ears, our cognitive modeling and problem solving capabilities are likewise shaped by several sources of power (reasoning by analogy, change of representation, natural deduction, etc.). This essay is about one of the latter sources of power, namely the power of *factoring*.

At one extreme, this includes literally factoring polynomial expressions—converting, e.g.,

$$x^4 - 41x^2 + 66x + 70$$

into

$$(x - 2)(x + 1)(x - 5)(x + 7).$$

At another extreme, “factoring” dons the costume of the weak method we call “divide and conquer”.¹ A third face of this same source of power, a form of it

¹“Weak” here meaning: imposing only weak requirements on a problem in order to be able to apply to it; i.e., very widely applicable [50].

which is so ubiquitous we are scarcely conscious of it, is the factoring out of less important aspects and variables and details, to facilitate the important ones to be perceived or attended to or figured into computations—e.g., assuming point-masses and ignoring air resistance when solving Physics 101 kinematics problems.

A fourth aspect of factoring is parallelization. In the physical world, sometimes there is some synergy that makes it more than just a linear speedup in the number of “processors.” For example, a friend helps you carry your couch upstairs, and then you help them move their couch, and neither of you could have lifted a couch by yourselves. But down at the level of running software, the power of n -way parallel processing is usually “just” a factor of n speedup.

Sometimes it’s less: One poignant moment at a Japanese Fifth Generation hardware demonstration in Tokyo in 1990 was when the developer proudly announced that, for some weather forecasting application, half the time all one million processors in their machine were productively busy simultaneously. I asked about the *other* half of the time, and was told that only one processor was active then—which meant that the total speedup, running that application on a million-fold parallel machine, was only a single factor of two.

Sometimes it’s more, in the sense that swapping a program in and out of memory in a conventional computer architecture can lead to thrashing, in extreme cases, which might be sidestepped through parallelization. But that superlinear speedup is the exception.²

Parallelization works when the components have some meaningful functional capability that allows them to work on their own for periods of time

²Many people today believe that the gain can be *much* more than a linear speedup, using quantum computers. I’ll go on record here as a skeptic—so far the only verifiable gain from quantum computing has been in the levels of funding for continued research in quantum computing.

large compared with the intervals over which they need to communicate with other components. *Engineered* parallelization works—gives its full factor of n —when each module performs an action and hands off an intermediate result to one neighbor just as it takes some other intermediate result from its neighbor(s) on the other side(s), *and* that process can be repeated over and over, *and* the ultimate functionality is nothing more than that composed process. In one dimension, think of a bucket brigade transporting water from a river to a burning barn.

Most real world situations are not so perfectly engineered, though, and parallelism works because each module is capable of somewhat independent, meaningful behavior. Think of the human beings in a factory—or in a society, or one person’s internal organs, or the cells of a single organ, or the nations of the world. James Grier Miller explores and analyzes this phenomenon at many levels in his marvelous book *Living Systems* [49]. It covers aspects of decomposition that go beyond simple parallelization, and so should we. So, with that, let us now turn back to the general issue of factoring.

Factoring in Software

In software engineering, Alan Kay recognized circa 1970 the value of factoring the overall functionality required of a program into modules that each had their own smaller, meaningful specialties.

In traditional software—both then and now—such modules are subroutines, functioning together like the parts of a car engine function together: the ensemble of subroutines is flowcharted, engineered to work together in completely well understood interactions to solve a particular class of problem. Their “range of independent thought” is as null as, say, a carburetor’s.

In the early 1970s, Alan stretched that paradigm and warped it into a new one, imagining the subroutines not like the parts of a mechanical device but more like a set of humans tasked with solving a problem, where one might

relax the degree to which their actions and interactions are prescribed. As with human beings, Alan reasoned, it might be worth *educating* the modules—investing resources in making each module smarter and more flexible—and then relying on the resultant community of agents to solve problems in ways not fully preconceived by their developers. In other words: make them smarter, better informed, give them higher level responsibilities, and then give them more freedom to meet their goals.

This zeitgeist thundered through Carl Hewitt's ACTORS [43], Seymour Papert's LOGO [51], my own BEINGS [46], and of course Alan Kay and Adele Goldberg's Smalltalk [40], and has reverberated ever since, sometimes explicitly as in the case of Mixins (in Lisp Machines FLAVORS [52] and, more recently, in Java) and sometimes almost unrecognized as in the case of Expert Systems [42] and, more recently, the Semantic Web.

In the case of BEINGS, we took the most sophisticated AI programs that had just been written, such as Pat Winston's ARCH Learning program, and made the task of the BEINGS program to automatically synthesize those AI programs, from an interactive English dialogue with the user. In the case of ARCH, this involved writing out a dialogue that was a thousand lines long. We took that and, for each line of dialogue, asked what actor—what specialized expert module—would be the appropriate speaker of it. When this process was completed, we had identified about eighty such expert modules (e.g., Psychologist, Loop-Writer, Sorting Expert) and dozens of things that each module would need to know, in order to speak up when/as needed and thereby carry out its role in that dialogue. Each such BEING was then coded, with each of those pieces of knowledge suitably generalized. Sure enough the community of BEINGS was able to carry out the dialogue with a human user, and produce ARCH and a few other substantially different programs such as an airline reservation system. Having the target programs in mind ahead of time enabled us to cut too many corners, and inspired us to make our next system, AM [47], not have any specific goal at all—its only goal being to discover interesting new things in mathematics.

Representation Matters

Each representation makes certain operations doable, or doable efficiently, at the expense of others. For example, after factoring

$$x^4 - 41x^2 + 66x + 70$$

into

$$(x - 2)(x + 1)(x - 5)(x + 7)$$

we can just read off 2, -1, 5, -7 as the four roots of the equation

$$x^4 - 41x^2 + 66x + 70 = 0.$$

Having an exploded-view diagram of a bicycle may help you orient the pieces as you assemble it, but may not easily capture the order to do things, tools to use, and similar details; a textual assembly instruction sheet fills those gaps but may not easily capture the relative positions and orientations concisely and clearly. That's why having multiple representations of more or less the same knowledge is often cost-effective. As Alan Kay quotes Marvin Minsky: "You don't understand something until you understand it more than one way." [45] A more familiar example is how few people in ancient Rome knew how to multiply two numbers, given the cumbersome Roman numeral representation they had to contend with, compared with our modern decimal notation. But perhaps that was a better example before calculators enabled us to lapse back into widespread innumeracy. Another famous quote of Alan's along these lines is that point of view is worth eighty IQ points [44].

It would be inappropriate to convene a committee of experts to tackle the problem of factoring a particular quadratic (or cubic or quartic) polynomial, but *failing* to take that sophisticated an approach to tackle a hard problem like medical diagnosis is just as inappropriate.

And yet we see that sort of "when all you have is a hammer..." under-representation all too often. Complex problems like diagnosis, or financial investment, are straight-jacketed into a format where they can be treated purely

statistically. The tens of thousands³ of important distinguishable relations that we would naturally employ to represent human knowledge, is flattened into a handful (subsets, synonyms, etc.) so it can be handled by the Semantic Web, or even worse into one single relation (word-occurrence) so it can be handled by Google.

Lack of factoring sometimes “creeps up” in term-lists, leading to a larger and larger number of terms. The resulting terminological standard ends up with a combinatorial explosion of *pre-coordinated* terms, instead of having a mechanism for expressing a term as a factored expression (what is called *post-coordination*). For example, in place of having a million medical terms like

possible-compound-fracture-of-the-distal-radius

in one terminological standard, one could have just a thousand terms which can be composed into nonatomic formulae such as

(possible (compound (fracture (distal radius))))

with that whole expression denoting a term. Anywhere that the original pre-coordinated term could have been used, that nonatomic term-denoting expression could be used in its stead [38].

In a way, this is akin to the way that each natural language breaks up the virtually infinite number of possible expressible thoughts into a manageable number of words (and rules for grammar, discourse pragmatics, literary devices, and so on).

At a **lower fractal level**, it’s akin to the power of having a small alphabet of characters composed to form words, rather than having a unique character for each word.

³As evidenced by, e.g., the number of relations named by a single word in English; or as evidenced by the number of relations (17,000 and counting) that we have had to add—kicking and screaming, one by one—to Cyc, over the past twenty-five years, to represent knowledge from newspapers and other publications.

At a **higher fractal level**, it's akin to the way in which a large article, e-mail thread, novel, etc. sets a *context* and then can much more tersely say a large number of things *in that context*.⁴

We are making use of this principle today in Cyc [48] to understand the meaning of a user's *ad hoc* natural language queries which are too long and/or complicated for early twenty-first century natural language parsers to handle. We recognize MadLib-like fill-in-the-blank fragments, the user verifies (or crosses off) some of those guesses, and then the system tries to find a single meaningful query that would have all, and only, those fragments. Remaining ambiguities may be resolved by bringing in common sense knowledge, general world knowledge, domain knowledge/context, etc. Often, there is only one plausible query with that property; it is paraphrased to the user, and then the formal predicate calculus version of it is handed off ultimately to database systems, web services, etc., to answer.

Many of our current applications of this are in the medical research domain, but consider this example from everyday news: "Cases in the 1980s of state politicians reversing their stances on healthcare policy issues and campaign reform issues after appearing on 60 Minutes." Fragments would include:

```
(x1 is a politician)
(x2 works for a government of a US state)
(x3 is for/against some issue)
(x4 changes their mind about something)
(y is a healthcare policy issue)
(z is a campaign reform issue)
(w is an episode of the 60 Minutes television program)
(v is an issue covered in a segment of w2)
(r happened before s)
(q happened during the 1980s)
...
```

⁴Context-setting can be done implicitly or explicitly, using phrases or sentences, choice of tenses, etc.

Part of fitting these fragments together involves deciding which variables almost certainly do, or don't, unify—refer to the same thing. For example, all the x_i in this example probably unify, but events r and s definitely don't unify (since an event can't happen before it happens). More subtly, understanding the scope of coordination (in this case, “stance on” coordinates over both healthcare issues and campaign reform issues), understanding when “and” means “and” and when it means “or” (in this case, the individual is probably looking for a case where the politician reversed their position on a healthcare issue *or* a campaign reform issue).

This works for reasons analogous to the reasons that DENDRAL [39] worked: bringing multiple bodies of knowledge to bear, each acting as a set of constraints on the possible answer. In DENDRAL's case, the task was to zero in on possible structural isomers having a particular chemical formula, and the sources of knowledge constraining that space included valence and bond constraints, three-dimensional folding constraints, mass spectrography results on the compound, etc. Often a search space of tens of millions of possible isomers would end up with just one candidate that wasn't ruled out. In the same way, our Cyc-based query system for, e.g., answering *ad hoc* clinical medical researchers' queries, brings to bear common sense, general factual information, medical knowledge, Gricean pragmatics (be truthful, be brief, don't add much more or much less than the enquirer needs for their current purpose, and so on) [41] and other discourse conventions, and so on. Often a search space of tens of millions of possible interpretations for the query—ways that a given set of “fragments” could fit together, given various variable unifications and choice of quantifiers and order of quantifier nesting—ends up with just one candidate interpretation that isn't ruled out.

Conclusion

This has shown the power of functional factoring, each module recognizing when it can help (messages it can handle) and stepping up to do its part, adding

its constraints, adding its power into the mix. While some of the more flamboyant sorts of factoring may capture our imagination and attention more than this low-level mundane sort of factoring, it appears that this “obvious” idea is the mainstay of much of the reasoning that we do as humans and which has been a bottleneck, for the last fifty years, to getting machines to understand what we’re trying to ask them.

Factoring is an important source of power which is used fractally up and down many scales of magnitude of phenomena, and which takes on many guises. Just as one would not want to build a house without a saw, or with *only* a saw, it should be viewed as an important tool in our toolkit of problem solving methods and equipment.

Alan Kay has been explaining this “obvious” idea to the world in various forms, ever since I met him almost forty years ago. Much of the world ignores it, or understands it intellectually but hasn’t internalized the lesson, to its peril and woe: Complacently ignoring it is a recipe for systems that appear to work, in the small, but that fail to scale up.

Peer review and the scientific literature aid and abet this tragedy, often publishing articles about some system, with a few examples, that is capable *only* of succeeding at those few examples. Refereed journals allow research programmers to be painters rather than performers, allow their programs to be snapshotted paintings; as Joni Mitchell said:⁵

That’s one thing that’s always been a difference between the performing arts and being a painter. A painter does a painting, and he paints it, and that’s it. He has the joy of creating it, it hangs on a wall, and somebody buys it, and maybe somebody buys it again, or maybe nobody buys it and it sits up in a loft somewhere until he dies. But nobody ever said to Van Gogh, “Paint *A Starry Night* again, man!” You know? He painted it and that was it.

⁵*Miles of Aisles*, Asylum Records, 1984.

The size and complexity of Internet content is a welcome forcing function in this sense. It is the deep end we find ourselves thrown into; inability to scale up is no longer an option, is no longer easy to hide.

We as humans may benefit from discussing this further from different points of view, but the real point is: if we engineer things properly, so may our programs.

Douglas Lenat was a Ph.D. student in Computer Science at Stanford in the early 1970s when he first met Alan. Doug's thesis was the "Automated Mathematician," getting computers to act like the sort of inquisitive kid that Alan (and others such as Seymour Papert) were building software tools for. The thesis won the biannual Computers and Thought Award.

As a professor at Stanford, Doug collaborated with Alan on the Intelligent Encyclopedia (Knoesphere) project at Atari. From that work grew Cyc, a knowledge base that Doug's team has continued to build up to many millions of axioms operated on by over a thousand specialized inference engines.

Doug was one of the original fellows of the American Association for Artificial Intelligence. In 2009 he was elected a Fellow of the American Academy of Arts & Sciences. He and Alan are TTI/Vanguard Advisory Board members, which affords them the opportunity to catch up several times a year.

Doug currently heads Cycorp, the Austin-based company that continues to build and apply Cyc.

Butler Lampson

Declarative Programming: The Light at the End of the Tunnel

*Ah, but a man's reach should exceed his grasp,
Or what's a heaven for?*

— Robert Browning, *Andrea del Sarto*

Goals

I started out to write about declarative programming, which seemed like a good topic because in a way it is the opposite of the kind of programming that Alan Kay introduced in Smalltalk and has been working on ever since, and also because Alan Borning's ThingLab [53], one of the first examples of general-purpose declarative programming, was developed in Alan's group. As I thought about it, though, I realized that I don't really know what declarative programming is. In fact, it seems to be an umbrella term for "the kind of programming we wish we had."

What kind of programming do we wish we had? We want to be able to tell the computer what to do, in a way that is easy for us and that reliably and promptly gets us the result we want (or an intelligible explanation of why we can't have it, and how we might change the program to get it).

The problem, of course, is that what the computer natively knows how to do is very far removed from this ideal. It *knows* how to perform very small, very precisely-defined state changes on a state space whose main component is an array of a few billion eight-bit numbers. We *want* it to look through a few attached cameras and drive a car through New York city traffic, or find integers x, y, z and $n > 2$ such that $x^n + y^n = z^n$. This is a gap too great to be bridged in any way we can currently imagine, so we must lower our aspirations.

There are two things we know how to do that make the gap smaller. One is to make the machine operate on more interesting datatypes than bytes—for example, on arrays of floating point numbers, on relations, or on images—and to do big operations on these datatypes, such as finding the eigenvalues of a matrix, or a list of all the registered voters in electoral precincts that went Republican in the last presidential election but are in cities that went Democratic, or the faces of women in a picture. The other is to make the machine optimize some function subject to a set of constraints, perhaps approximately. The challenge is to use these two methods (and anything else we can think of) to come closer to our goal.

The most common banners under which people have tried to do this carry the labels *domain-specific languages* (DSLs) and *declarative programming*. The first is fairly easy to understand. The idea is to restrict the scope of programming enough that the machine can do a good job, albeit within narrow boundaries. Parser generators and MATLAB are examples at the two poles of this approach. A parser generator meets our ideal perfectly, as long as the only thing that can vary in what we ask for is the language to be parsed. MATLAB is very good at handling problems that can be solved with standard operations on vectors and matrices; if you want to do something non-standard, it pushes you closer to programming in FORTRAN. The most common fate of a DSL is to be absorbed into a general-purpose programming language as a library, perhaps with a little additional syntax as in Linq;¹ this happens because when the DSL is successful people try to stretch its boundaries, adding more and more

¹<http://msdn.microsoft.com/netframework/future/linq>

general-purpose facilities, and you don't have to go very far down this path before you have a clumsy general-purpose language that is hard to understand and hard to support.

*By relieving the brain of all unnecessary work,
a good notation sets it free to concentrate on more advanced problems,
and in effect increases the mental power of the race.*

— Alfred North Whitehead, *An Introduction to Mathematics*

Declarative programming

Declarative programming is more puzzling, and it is the main topic of this paper. The fact that no one knows what it is gives me free rein to reflect on a wide range of ideas and techniques.

Two somewhat unrelated goals seem to motivate the idea of declarative programming:

1. Make it easier to get from a precise specification of what the program is supposed to do to a working program. Two examples of this are SQL queries and parser generators.
2. Make it easier for a non-programmer to get from a fuzzy idea of what they want to a working program. Two examples of this are spreadsheets and search folders.

This paper is mostly about the first goal, though it has some things to say about the second.

It's easier to say what declarative programming is not than to say what it is. Certainly programming in the native instruction set of the computer is not declarative programming, and neither is programming in C, Visual Basic, or Smalltalk. In fact, any program that explicitly gives the computer a long sequence of small steps to carry out is not declarative; this means that a program with loops or recursion is not declarative. One consequence is that there's not

much hope for using declarative programming all the way down to the bare machine, as one can do in Smalltalk: it's not turtles all the way down.

At the opposite extreme, "do what I mean" is not declarative programming either. In other words, a declarative program is not magic, and it doesn't make wild guesses about the user's intent. It is just as precise as any other program.

It is common to classify programs as imperative (with object-oriented as an important case) or declarative (with functional and logic as important cases). In practice, however, these categories are not strict. Imperative programs often have large parts that are functional, and functional programs in systems like MapReduce and Dryad usually have computational kernels that are written imperatively, though their external behavior must be functional.

Successful declarative systems usually have a few things in common:

1. They give you a way to write the program that is a **good match** to the user's view of the problem. Another way of saying this is that the system synthesizes a program that the computer can run efficiently from a specification that the user writes, which may have a very different form. The purest version of this is the planning that has been part of robotic systems for many years [63]. It used to be called *automatic programming*, but that term has fallen out of favor. An important aspect of a good match is that the user can employ a familiar vocabulary. Thus declarative systems often involve a DSL, or a database schema that has been worked out by someone else. Another important aspect is that you can debug the program at the same level that you write it; macro recorders generally fail this test.
2. They are **compositional**, which means that you can write a program in small pieces that are fairly independent, and the system will put them together automatically. A spreadsheet is a simple example of this, and a solver for an optimization problem with constraints such as linear programming is a more sophisticated example. Functional programming is the most basic composition mechanism.

3. They give you **big primitives**, so that you can get a lot of work done without having to write a lot of code, and your program only needs to have a few steps. A primitive can be big by operating on *big data* (arrays, graphs, relations), by *solving* a nontrivial system of equations or constraints (such as linear programming or Boolean satisfiability [57]), or by embodying a *powerful algorithm* (such as scale-invariant feature transforms in computer vision [60]) or a *powerful data structure* (such as a balanced tree for storing ordered data).
4. They have clean **escape hatches**, so that you can fall back to boring old imperative programming when efficiency, familiarity, or legacy code demands that. An escape hatch may be internal, allowing the declarative program to invoke a primitive written in some other way, or external, allowing an imperative program such as a shell script to invoke a declarative program.

Another characteristic of most declarative systems is that you can get started (do the equivalent of “Hello world”) with very little effort, though certainly other systems like Python have this property too.

Don't ask what it means, but rather how it is used.

— Ludwig Wittgenstein, unknown source

Examples

Another way to characterize declarative programming is to look at some examples of successful declarative systems:

Spreadsheets such as Excel. A spreadsheet is functional programming with a human face, without recursion, and with powerful primitives for tabular layout, for charts and graphs, and for aggregating data (pivot tables). Excel has a rather clumsy escape hatch to Visual Basic. Hundreds of millions of people

have learned how to make a spreadsheet do useful work, though only a few can use more than a small fraction of its capabilities.

SQL queries. This is functional programming with big arguments (relations), powerful primitives (for aggregation), and good optimization. It has also been enormously successful, though it's a tool for professionals—the general user needs a front end to generate SQL, such as a form to fill in, and these front ends only expose a small fraction of SQL's power.

Parser generators such as yacc are a successful example at the opposite pole from these two. They produce a parser for a context-free language from a grammar. Where Excel and SQL share an expression language with ordinary imperative languages, and have escape hatches to general imperative programming, a parser generator is as domain-specific and declarative as possible. It takes a specification that *is* the user's intent (the grammar defining the sentences to be recognized), often produces a parse tree by default, and usually has a very stylized escape hatch that just allows you to write patterns to define what the output tree should be (though some let you attach to each grammar rule some arbitrary code that runs in a context where the results of the parse are accessible).

Streaming data flow systems like DryadLINQ [64] (which grew out of Unix pipes and the AVS graphics system [62]) are an interesting variation on functional programming. They let you write arbitrary kernels that take a set of input streams and produce a set of output streams (some of which might be much smaller if the kernel does aggregation). Then you can compose many such kernels into a dataflow graph that can be deployed automatically over a number of CPU cores or cluster nodes. Dryad (and MapReduce, which is less general) can automatically partition the computation, run it on thousands of compute nodes in a cluster, and recover from detected failures of some of the nodes. Here the main value is that you can easily express complex operations on very large data sets, and the system handles partitioning, scheduling, concurrency and fault tolerance automatically. This kind of composition and scaling is similar to what you get from a transaction processing system. Dryad has escape

hatches both above and below: you can program the kernels any way you like as long as their only communication with the rest of the world is through Dryad's streams, and you can take the result streams and process them with ordinary .NET programs; this works because Dryad's datatypes (collections) are also .NET datatypes.

Mashups are a DSL that exploits two powerful features of HTML and XML: a hierarchical namespace that extends all the way down to the smallest elements (even to single characters if you like) and fairly elaborate two-dimensional layout of text and graphics. When you combine these with the web's ability to fetch information from anywhere in the Internet and the existence of more or less functional web services for search, mapping, financial and demographic information, etc., you can easily produce nice-looking displays that integrate a lot of disparate information. The escape hatch is JavaScript.

Mathematica is a DSL that deals with symbolic mathematical expressions. It gets its power by embodying sizable pieces of mathematics (polynomials, differential equations, linear algebra, etc.) so that it can solve a wide range of equations. In addition, it can evaluate expressions numerically and solve equations numerically if symbolic methods fail, and you can easily turn numerical results into two- and three-dimensional graphics. It incorporates its own general-purpose imperative programming language, so it doesn't need an escape hatch. MATLAB is a similar system that specializes in numerical linear algebra and digital signal processing. Numerical computation is steadily becoming more declarative.²

Security policy is not very declarative today; usually you have to specify the access controls for each object individually, which is time-consuming and error-prone. Experimental systems such as Chang's [55] show the possibilities for expressing policies in a way that is very close to the user's intent.

Table 1 summarizes these examples.

²http://www.nag.co.uk/market/trefethen_future.asp

Example	Themes / composition	Data model	Algorithms / primitives	Escape hatch
Excel	FP	2-D tables	Incremental evaluation	Imperative (Visual Basic)
SQL	FP, scaling	Relations	Queries, aggregation	Imperative (TSQL)
yacc	DSL for context-free languages	Context-free grammar	Context-free parsing	Output patterns
DryadLINQ	FP for streams, scaling	Data streams	Partition, fault tolerance	Arbitrary kernels; embed in .NET
Mashup	Path names	Labeled tree	Graphical layout	Imperative (JavaScript)
Mathematica	DSL, recursive FP	Math expressions	Math	Native imperative
MATLAB	DSL	Matrices	Linear algebra	Native imperative
Web site security	DSL	Relations	SAT solver	None

FP = Functional Programming, without recursion unless otherwise noted.

DSL = Domain-Specific Language. They come with powerful built-in algorithms that operate on the domain.

Table 1: Examples of declarative systems

Time is nature's way of keeping everything from happening at once.

— variously attributed

Failures

Many attempts have been made to do less domain-specific declarative programming. I think it's fair to say that all of these have been failures: they are based on powerful ideas and can do some very impressive toy examples, but so far at least, they all turn out to have limitations that keep them from being widely adopted. The basic problem seems to be that these systems are solving a system of equations or constraints, and it's too hard

- to write down everything needed to avoid undesired solutions,
- to keep the solver's problem from becoming intractable, and
- to make the program modular, which is essential for large problems.

These systems fall into three main classes: constraint programming, logic programming, and algebraic specifications for datatypes.

Constraint programming, as in ThingLab, is very appealing, since it's often easy to obtain constraints directly from a specification, and a constraint solver is a very powerful primitive. A variation is to add a goal function of the variables to be optimized subject to the constraints. The difficulty is that the only general solution method is some kind of search of the solution space, and you have to choose between very stylized constraints for which there's an efficient search algorithm, as in linear programming, and more general constraints for which the only known method is exponential. If the goal function is differentiable then hill climbing sometimes works, but usually there are many local maxima.

Logic programming, as in Prolog, has the same intractability problem, even though the domain is just Boolean values rather than reals, lists, or whatever, and usually this simplification is counterbalanced by wanting to deal with many more variables. There are periodic waves of enthusiasm for Prolog or for

the closely related rule systems that underlay the expert systems of the 1980s, but they don't last.

Algebraic datatypes are rather different, since they are a way of writing a specification, not a program, but their failure illustrates some of the same points very clearly. The idea is that you can specify the behavior of a datatype such as a queue by specifying the primitives (*put* an item on the end, *get* an item from the front, *test* for empty) and a set of axioms that they satisfy, given in the form of equations. This strategy falls foul of the fact that it's amazingly difficult to write down a set of consistent axioms for even a simple datatype that doesn't allow all kinds of undesired behavior.

All of this is not to say that constraint solvers, optimizers and theorem provers are useless. On the contrary, they are very valuable primitives, just not able to bear all the burden of expressing a program. Whether it's a linear equation solver, a polynomial root finder, a linear programming package, a regular expression matcher, a polymorphic type inference system, or a SAT solver, it can be a good tool as long as it works on a closed system whose interactions with the rest of the world are the responsibility of the programmer rather than themselves being governed by automatic search. There are a few examples of solvers that can be extended cleanly, such as SMT theorem provers [56], but they are conspicuous by their rarity and don't eliminate the fundamental intractability.

Make no little plans. They have no magic to stir men's blood.

— Daniel Burnham, as quoted by Charles Moore

Big Primitives

Most of our steadily increasing ability to use computers to solve increasingly large and complex problems is based on the availability of more and more powerful primitives that you can use in a program with confidence that they will deliver what they promise. I have mentioned some of these already, but it's instructive to see a (necessarily partial) catalog:

- Matrix operations
- Linear programming
- Symbolic mathematics
- SAT solvers
- Synthetic graphics, both two- and three-dimensional; games show what is routinely possible
- Image processing; striking examples of what's possible are Street View and Streetside, Photosynth [61], and the World Wide Telescope [58]
- Vision, still much inferior to human vision but able to extract three-dimensional models of buildings from video
- Relational queries
- Full text search over corpora of many terabytes
- Typesetting and layout from HTML (or other forms of text, such as T_EX)
- Graph algorithms such as PageRank [54]
- Views on relations, and especially two-way mappings between relations and forms
- Machine learning (a specialized form of program synthesis)

In addition, there are techniques that make it easier to get a computer to do our will and are more broadly applicable than individual primitives, but that are not instances of declarative programming:

- Transactions, which make it easy to do complicated operations atomically, and to abandon them when necessary without having to worry about side effects
- Undo and versions, which make it easy to experiment in more general settings than transactions and to keep track of the evolution of a big project
- Static analysis of programs to infer their properties
- Lazy and speculative execution, powerful general methods for matching the work that a computer does to the needs of the problem

- Indirect references and incremental execution, which make it much easier to adapt a program to a changing environment

*And the users exclaimed with a laugh and a taunt:
“It’s just what we asked for but not what we want.”*

— unknown

Non-programmers

A non-programmer is someone who is uncomfortable with precision and abstraction, which seems to cover most people. For the most part they can only tell a computer what to do by pushing buttons. Sometimes one button push does a lot, but if there’s not a button (perhaps with a few accompanying form fields) that does what they want, they are reduced to leading the computer by the hand with a sequence of manual button pushes. Except for spreadsheets, we have not been very successful in finding better ways for them to adapt the computer to their needs. Macro recorders help a little, but very often a recorded macro needs to be edited to make it useful, and in every system that I know about this is too hard for a non-programmer. Because most declarative systems depend on precision and abstraction, they are not much help.

I can only speculate on how we might improve this situation: by making it possible for a person to engage in a dialog with the computer, explaining either in natural language or by example what they want the computer to do (change all the references in this paper into PMLA form, or turn on the heat when a person gets up but ignore the dog, or tell me which of my Facebook friends knows fewer than two of the others). The computer’s side of the dialog is its expression, in terms meaningful to the person, of what it’s supposed to do or of what it doesn’t understand. The person then can give corrections or more examples. This is obviously a form of program synthesis, and it’s declarative in the sense that it’s not a long sequence of small steps. For it to work, the

computer and the user have to share a conceptual model of the problem domain. A small step in this direction is Miller's keyword programming [59].

Conclusion

For forty years people have been working to make programming easier, faster, and more reliable. For non-programmers it's also important for the machine to help the users state their needs precisely. So far the biggest successes have come from domain-specific imperative languages and from providing powerful primitives that you can invoke from imperative languages. Declarative programming seeks to go further, allowing you to state what you want from the program and have the computer synthesize it, or less ambitiously, to explicitly give the machine only a few steps for it to take. This works to some extent, and it works best for specific domains and when you have big primitives.

As machines get better at reasoning, as computers are integrated more deeply into application areas, and as we build bigger primitives, surely declarative programming will get better. Two things that will help are codifying more information in a form the machine can understand, and building primitives in a form that declarative programming can easily use.

Butler Lampson holds a B.A. from Harvard and a Ph.D. from the University of California at Berkeley. He has been a member of the faculty at Berkeley, the Computer Science Laboratory at Xerox PARC, and the Systems Research Center at Digital.

He is a member of the National Academies of Sciences and Engineering, and a Fellow of both the ACM and the American Academy of Arts & Sciences. He has received numerous awards including the Turing Award in 1992 and the NAE's Draper Prize in 2004.

Butler is currently a Technical Fellow at Microsoft and Adjunct Professor of Computer Science and Electrical Engineering at MIT.

Vishal Sikka

Some timeless lessons on software design

Alan is an amazing historian. A walking encyclopedia, full of anecdotes, examples and analogies that illuminate, challenge and stimulate. He often provides extreme examples and comes up with wonderful quotes that illuminate a concept starkly. He once compared the ability of a business to solve the fundamental problems facing it, while keeping a very narrow and near-term focus, to a bunch of monkeys trying to get to the moon by climbing up ever taller trees or of people trying to save a sinking ship when what they really need is to build an airplane to fly out of there. He spoke of an individual computer from the 1950s that was the size of a football stadium, when reminded of modern data centers. When I discussed with him the challenge of renovating a 400 million-line codebase he described to me a large company that had a single long-lived software system with an ongoing list of 139,000 requirements for it! A big fan of the use of logic for expressing meaning, when talking about logic as a way of executing specifications, he once compared pure logical reasoning to building a cathedral from the steeple down. Alan's has been a lifelong pursuit of extremes, of the difference between news and new. And he usually motivates us with extreme examples and extreme challenges. Extreme simplicity: can you write an entire computer system, from screen to metal, in 20,000 lines of code? Extreme late-binding: can you execute this *specification*? Extreme optimization: can you bake this system down into a FPGA?

Alan's work at Xerox PARC was a singular example of a lab engaged in extreme experiments whose impact is still growing, more than thirty years later. And while this has inspired many of us in the industry to have dedicated teams that, untethered, look to the future and its fundamental implications on the present, Alan's approach of identifying extreme problems as a way to move the needle in a non-incremental way is an even more important element of conducting tangible research.

Alan once told me that a mystic is one who finds common ground amid diversity. And that this, in turn requires identifying, and disentangling, those things that we often assume are interconnected, but which enable new beauty and innovation when disentangled. One doesn't often think of Alan as a mystic, but it is in this aspect of him that I have come to learn a great deal from him. Alan has taught me a great deal about such disentanglement, as well as various ways in which concepts and principles can be combined and divides overcome. This in turn has shaped my own work from the days of my Ph.D. thesis to my work in enterprise software systems. So I want to share lessons that I've learnt from him on three fundamental topics related to this.

Lesson 1: We need to build systems around the *viewpoints* of end-users

As we know, Alan's research institute is called Viewpoints. One of Alan's teachings is that systems need to be built, as he says, "from the screen backwards," i.e., systems need to reflect the intent and views of their users. Smalltalk was one of the first serious efforts towards this. Views are more than projections on software artifacts: they capture our intent, our focus. And therefore a good purpose-built system is one that morphs itself to the perspective, the viewpoint, of its users. Many techniques have emerged to systematize this principle. Design-thinking, articulated by David Kelley, is one of these. Hasso Plattner, SAP's founder and chairman, has been a strong proponent and champion of using design-thinking to build systems that serve users in a well-defined way. A great challenge in constructing systems, especially software systems, is to

simultaneously achieve generality or broad reach, and yet enable individuality. Alan's work teaches us that the best way to achieve this is to have systems that reflect a given viewpoint of the users, and are designed to be adaptive so as to easily bring together the artifacts that serve this viewpoint.

Business applications need to achieve this duality of reach and individuality. But a key challenge is that core business applications, such as financial accounting systems or systems to manage customer relationships, are often very long-lived—a typical installation of such a system at customer sites could easily live for more than ten years, whereas the user-interaction (UI) technologies that people use to reach such systems, often evolve very rapidly. To achieve this dual goal of not disrupting a long-lived system, and yet enabling access to it using new technologies, I have found Alan's thinking to be very instructive in shaping our architectural work. Building systems from scratch that are adaptive to the end-user's viewpoint is rare enough; but enabling easy construction and adaptation for end-users on top of legacy systems is an altogether extreme challenge; and that too on systems that don't even have clean interfaces. (Firelanes, carved to combat a forest-fire in pieces, was a metaphor Alan used once to describe this to me.) We, at SAP, enabled such a construction.

We enabled constructing views atop a legacy system that are adaptive to an end-user's viewpoint, are container-independent and yet use the native experiences and services of that container. This meant first of all decoupling the content of a view from the container that renders it. Our system would have end-user view objects, i.e., container-independent or rendering-independent views on the legacy system, which could be coupled with any specific rendering technology. Assemblies of such objects could create adaptive new ways for users to interact with a system, in a way that is completely decoupled from the underlying system, and yet can use every artifact necessary from that system. Recall that such a thing was first done systematically in Smalltalk [66], although thankfully Alan and his team did not have to plug these onto old systems. Another key challenge was to express the view in a way that it could be constructed efficiently from the underlying system. This meant expressing

the view in the language of the consumer, but executing it in the language of the provider system. So we needed a way to connect these two worlds, an adaptation layer, where the intent of the outside world could be executed in the inside world. We managed to achieve this, in several ways in fact, thereby creating an easy-to-use mechanism to efficiently expose the traditional SAP applications to outside consumers, including consumers yet to be invented. This technology is beyond a concept, and already being used in SAP's products that enables SAP's traditional applications to be accessed natively from Microsoft's Office products, IBM's Lotus products, and various mobile devices, among others.

Lesson 2: Systems need to separate meaning from optimization

Nearly twenty years ago, in my doctoral work at Stanford, I started looking into the issue of combining the power and simplicity of expressing meaning using logic, with the optimization and efficiency that was possible using individual implementations of that meaning. I looked at this problem from the viewpoint of combining declarative systems with efficient specialized reasoning [67] procedures. Alan summed it up simply, as he often does, as an example of separating meaning from optimization. So logic could be used to express what it is that we want to achieve in a very generic manner, and yet for execution we would rely, when possible, on specialized procedures with highly optimized execution. These optimizations could be in the form of highly efficient code in some specialized language that would make all or some of the cases of the meaning run faster or conditions/heuristics that could be placed on the optimizations depending on certain conditions of use. More than a decade later I realized that this principle was fundamentally needed in enterprise systems and Alan helped make this connection.

Once again the underlying reason was that various software components used by businesses evolve at different speeds and have different pedigrees. And these need to interoperate. So an obvious need was to use logic as a way to

describe systems so they could interoperate in a way that is independent of any particular integration technique (or “glue”). Beyond integration, these descriptions could help describe systems, their behavior, their internal artifacts to help manage the long lifecycles of these systems, and also to manage change over these long lifecycles. But there is an even more fundamental issue in large-scale packaged applications. There are many—often many thousand—components within such applications and programmers who understand one particular domain and can program some of these components well, often are not experts at others. To achieve an economy of scale, it is necessary to achieve rapid time to learn, and to have development teams write code in cost-efficient ways. Hence there isn’t usually a single way to enable the construction of such systems. We need to find ways to enable large teams of programmers with different backgrounds to build different aspects or components of systems very efficiently and at scale, and yet in a way that preserves an overall coherence of a system, that prevents barnacles from forming (another analogy from Alan). So we need efficient ways to express meaning coherently. At SAP we have looked into various ways of achieving this. One promising way, which Alan often brainstormed with us on, was to have a programming model where a base language, such as JavaScript or Ruby or Java, was extended by means of multiple domain specific languages. These languages were all designed for individual domains, but were joined at the base by a shared glue, which was also the way to extend the base language [65]. This way, specialized programmers could build code using constructs and conveniences from their domain, that they were comfortable with, and yet produce systems that were coherent underneath via the common base. And the execution of systems expressed in this way was kept independent of the meaning expressed so, opening up the room to optimize the system’s execution across the layers, even down to the metal if feasible and desirable. So this simple principle, of separating meaning from optimization, achieves this duality of easy expression of large types of meaning without compromising on performance and optimization.

Lesson 3: Systems need firelanes and to be able to “spoon” across these

I mentioned firelanes in lesson 1 above as a fundamental means of enforcing system boundary, decoupling and encapsulation. Exposing long-lived interfaces for systems is not a new concept in the software industry. However, doing one firelaning exercise to expose multiple types of interfaces for multiple types of use, is relatively new. The best way to achieve this is via machine-readable semantic descriptions of artifacts inside a system. Such well-defined interfaces on legacy systems are very helpful because they enable a system to be reassembled in service of a new purpose or viewpoint and enable optimizations across layers of abstraction. Ian Piumarta, a colleague of Alan’s, has a nice analogy for this: spooning through the layers, just as a spoon cuts through the surface of a *crème brûlée*, rather than having to cut through a concrete surface with a drill. Interfaces that are machine-readable, i.e., the semantics of a system’s artifacts are exposed to the world outside, achieve such a result. A well-specified system, with machine readable documentation, can not only enable itself to be adaptively reassembled, but is much easier to maintain and change over a very long lifecycle.

Yet, as Alan often says, firelanes are not to be confused with unnecessary layers of abstraction. In modern software design, one often sees a temptation to achieve ease of expression using layers of abstraction. And yet naïve implementations of layers of abstraction can cost dearly in optimization. Large software systems cannot afford this. For one, underlying technology infrastructure changes very rapidly. Moore’s law and other non-linear improvements in price-performance make it necessary every once in a while to redeploy software in new deployment patterns. Beyond deployment optimization, there are other forms of optimization as well. Given the nature of an application, there are various different ways of optimizing the triplet of managing its state (and the underlying data stores serving that state), the transmission of this data over networks, and the processing in CPUs. We recently implemented an example of such elastic optimization across layers of abstraction by replacing traditional

relational databases with a memory-based data manager that organizes data by columns in memory, instead of rows on disk to achieve significant performance improvements especially for analyzing data. This speedup is achieved because columnar organization of data enables far better selectivity in querying complex object structures, far better partitioning and easier parallelization for very fast data processing in modern multi-core processors. This performance improvement was achieved while the relational algebra expressions (e.g., SQL) stay unchanged as the mechanism to express meaning: a classic example of separating meaning from optimization, and optimizing across firelanes.

Epilogue

The three perspectives above are a small inheritance from Alan's immeasurable contributions to our understanding of software. These are elements of a collection of software principles that I have referred to as Timeless Software:¹ a set of principles I've learnt along my own lifelong pursuit of unearthing constants amid continuous change, and of divides and couplings around us that are unnecessary. Alan's work, teachings and numerous conversations and whiteboard sessions with him, have helped shape these in a very fundamental way and for this I owe him an enormous debt of gratitude.

¹<http://sdn.sap.com>

Vishal Sikka holds a Ph.D. in computer science from Stanford University. He has worked in the fields of automatic programming, information and application integration, and artificial intelligence at Stanford, at Xerox Labs in Palo Alto, and at two start-up companies.

Vishal was founder and CEO of Bodha, Inc., developing technology for integration of enterprise applications and semantic information. When Bodha was bought by Peregrine Systems he became responsible for application development, integration technologies and architecture.

He moved to SAP as head of the advanced technology group responsible for strategic innovative projects, and later as senior vice president of architecture and chief software architect.

Vishal is currently Chief Technology Officer (CTO) and member of the executive board of SAP where he is responsible for the company's global research and innovation efforts and leading the company's technology, strategy and delivery.

Vint Cerf

Thoughts on Alan's 70th Birthday

The best way to predict the future is to invent it.

— Alan Kay

How many times have we heard this quoted over the past several decades? How many of us have been driven, in some sense, by this observation to explore new possibilities by experimentation? What an implicit and compelling challenge this observation has posed for so many!

Alan's inventions have always had a kind of stimulating quality to them. I think of FLEX, Smalltalk, Squeak and I also think about his deep devotion to understanding how kids learn. Indeed, I think Alan persuaded me to dispense with the word “teach” from my vocabulary on the grounds that teaching is not what learning is about. Learning is about discovery, often by doing as opposed to reading about or hearing about some activity. Doing an experiment to derive the basic formula for gravitational acceleration is far more compelling and memorable than looking at a formula already prepared for you.

Alan is an iconoclast who is not shy about breaking eggs to make omelets. He's also a major proponent of simplicity and elegance. Inventing languages that can compile themselves, building and playing elegant organs, distilling ideas until they become like crystalline Platonic absolutes: that's Alan. I've

often thought that working with young people has the wonderful property that they are too young to know “you can’t do that,” so they just do it! That’s Alan, too. He’s still too young to know “you can’t do that!”

So what about this “inventing the future” idea? It seems to me that we have barely begun to appreciate what is possible with the use of digital technology. Some ideas seem to get recycled. Mainframes become departmental machines and these morph into workstations, desktops, laptops, mobiles and then, suddenly we get cloud computing systems like gigantic, time-shared mainframes of old. Neural electronics are emerging to restore hearing, vision and motor function. We are discovering new ways to interact with computing systems including touch, gesture and voice, in addition to mouse and keyboard.

Google recently announced a capability to produce automatic captions for videos with sound tracks containing speech. While by no means perfect, and only for English so far, the announcement illustrates a stepping stone towards new and more natural interaction with computer-based systems. Video material can be indexed based on the text of captions. English captions can be translated into fifty other languages (with varying degrees of idiomatic quality). Putting computing technology to work to bridge gaps between various media and between groups of people who might otherwise be unable to interact seems like a perfect example of Alan’s inventing the future.

How far can we push this paradigm? There seems to be no end in sight. Software is an endless frontier, to paraphrase Vannevar Bush. If you can imagine it, you may well be able to program it. The Internet, World Wide Web, and the languages that go with them (HTML, XML, Java, JavaScript, Python and many others) serve as platforms for innovation. Open Source software is rewriting the mechanisms of invention, allowing creativity to flywheel from one contributor to the next. Standards contribute to this process by establishing interoperability among independently written packages of software and hardware.

We carry our information windows on our hips or in our purses. Through them we have access to an increasing amount of human knowledge and online services. Our “mobiles” have long-since ceased to be merely telephones and

have become the “Swiss army knives” of online functionality. New “blades” are added by simply downloading new applications. Cloud-based services augment what is not done locally. Collaboration is becoming easier and is facilitated by efficient sharing of information. Scientific work is accelerated through access to common databases maintained in collaborative fashion. Works that were not online are retrospectively given new digital life, opening them up to computer-aided searching and aggregation.

Ordinary appliances have not escaped the digital revolution. Increasingly they are found on the Internet or at least in local networks (at home, at work, in the automobile or on our persons). Their injection into our networked world opens up new opportunities for remote reporting and control and even more efficient power management.

We would be remiss in not observing, however, that this plethora of online access has also opened up new avenues of vulnerability and attack. Along with the benefits of these online capabilities come the risks of foul play or accidental cascade failures, annoying e-mail (spam), fraud and other kinds of abuse. While we are inventing the future, we must also attend to the social and economic effects these inventions may have. National and international law enforcement and cooperation must evolve to match the fountain of digital innovation that produces new applications daily.

Ultimately, we will find that Theorem 206 is forever true:

Everything is more complicated!

Widely known as one of the “Fathers of the Internet,” Vinton G. Cerf was the co-designer of the TCP/IP protocols and the architecture of the Internet.

Vint served as chairman of the board of the Internet Corporation for Assigned Names and Numbers (ICANN) from 2000 –2007 and has been a Visiting Scientist at the Jet Propulsion Laboratory since 1998. He was founding president of the Internet Society (ISOC) and is a Fellow of several institutions including the American Association for the Advancement of Science, the American Academy of Arts & Sciences, the International Engineering Consortium, and the National Academy of Engineering.

He is currently Vice President and Chief Internet Evangelist for Google, where he is responsible for identifying new enabling technologies and applications on the Internet and other platforms.

Mitchel Resnick

Life as a Learning Lab

Over the years, many of my interactions with Alan have come at his Learning Labs. For twenty-five years, Alan has organized these gatherings once or twice a year, bringing together people from a variety of backgrounds to engage in new types of learning experiences and to rethink ideas about learning and education.

For me, these Learning Labs capture the essence of what makes Alan special. Many people see Alan as a great computer scientist, given his pioneering work in object-oriented programming. Other people see Alan as a great visionary, given the profound and enduring impact of his Dynabook concept, which provided a glimpse of mobile computing with dynamic media years before other people had thought about these ideas and decades before the technology was actually ready.

These perceptions are certainly valid: there is no doubt that Alan is a great computer scientist and a great visionary. But as I see it, what's most special about Alan is that he is one of the world's great learners. He is excited about all aspects of learning: doing it, analyzing it, understanding it, encouraging it, promoting it.

Alan's Learning Labs last only a few days. But for Alan, all of life is a Learning Lab.

Anyone who has spent time with Alan knows that he is a voracious reader. He reads more books in a week than most people do in a year. At his home in Los Angeles, Alan has put together not simply a book collection but a full library, with books organized on shelves according to Dewey Decimal classification.

But Alan also understands that book knowledge is only one dimension of knowledge. At Alan's Learning Labs, people are encouraged to put their inhibitions aside and immerse themselves in new learning experiences. I have fond memories of learning to play drums with Arthur Hull, learning to build rockets with Modesto Tamez, learning to draw with Betty Edwards, learning to create toys with Arvind Gupta, and learning to sing with Don Lewis. In each case, the leaders of the activities were not so much teaching us as inviting us to join them on new learning adventures.

Many of these activities pushed me outside of my comfort zone—but that's precisely the point. Alan understands that learning involves trying new things, exploring the unfamiliar, taking risks, experimenting and re-experimenting. When I was in fourth grade, my teacher told me to lip-sync during the holiday concert, since my voice was so off-key. At Learning Labs, Alan got me singing again—and learning from the experience.

Learning Labs combine immersion with reflection. People spend part of the time diving into new and unfamiliar activities—then stepping back and reflecting on learning experiences—then imagining new ways to share these activities, ideas, and experiences with others.

Sometimes people associate this type of immersive, experiential learning with a lack of rigor or systematicity. Not Alan. Indeed, this is where Alan parts company with some progressive educators. Alan puts a very high priority on people becoming systematic thinkers, and he knows that won't happen without the right type of structure and support. Alan is a firm believer in what Seymour Papert has called "powerful ideas"—a set of concepts that provide special leverage for understanding the workings of the world. As Alan often explains, these concepts have been developed by societies over

centuries, so we shouldn't expect students to re-invent these ideas totally on their own.

At Learning Labs, we've spent hours and hours discussing how we can help students follow their interests and passions, and also help students learn powerful ideas and develop as systematic thinkers. It's not easy to accomplish both goals. At some Learning Labs, people have presented projects or ideas that supported the first goal but paid insufficient attention to the second. That's when I've seen Alan become most animated and argumentative. It's clear that no one is a stronger proponent and defender of powerful ideas than Alan.

Given Alan's expertise and interests, it's not surprising that some of the discussions at Learning Labs revolve around new technologies. But learning always remains an important theme. Indeed, technology and learning are woven together in many activities at Learning Labs, as they are in Alan's thinking and in his life. From one Learning Lab to the next, technology and learning co-evolve. Strategies for learning and education evolve based on the availability of new technologies, and new technologies evolve based on new ideas about learning and education.

Alan initiated the Learning Labs in the early 1980s, when personal computers were first entering the world in significant numbers. I started participating in Learning Labs in the early 1990s, and Alan soon invited me to bring my whole research group. For the next decade, we made it an annual tradition: my group joined Alan's group for the August Learning Lab held at Apple Hill in New Hampshire. For my research group, it was a great way to start the new academic year—diving into new learning experiences and sharing ideas with Alan's group.

Indeed, my group's Scratch software, our biggest and most important project, can be traced directly to the Learning Labs. It was at the Learning Labs that I first saw Etoys, the children's programming environment developed by Alan's group. I was inspired by the way Etoys connected powerful ideas with youth culture, adding programmability to the media-manipulation activities that are popular with many children and teens. Etoys had a strong technical

foundation, building on top of Alan's own Squeak software, and also a strong conceptual foundation, building on top of the powerful ideas represented in Seymour Papert's Logo.

I saw great potential in Etoys, and hoped that some of my group's ideas might contribute to the further development of Etoys—particularly in making the software even more accessible to children. I suggested to Alan that we get together, along with my colleague Brian Silverman, to discuss possibilities for collaboration. Alan generously set aside an entire weekend to meet with us. The discussion, as is typical with Alan, was incredibly wide ranging, touching on everything from the advantages of late-binding programming languages to the challenges of educational initiatives in under-served communities to the latest research in evolutionary biology.

At the end of the weekend, Alan made a suggestion: Rather than setting up a direct collaboration between our two teams, he proposed that each team work on its own project but freely share ideas and code with each other. That way, he said, we would explore a wider range of possibilities and continue to learn from one another.

And that's how our Scratch project was born. In developing Scratch, we borrowed lots of ideas from Etoys—and even borrowed John Maloney, one of the main programmers from the Etoys team. As Scratch has continued to grow and evolve, Alan has been generous not only with his time and ideas but with his support and encouragement—even though Scratch is, in some ways, a competitor to his own Etoys project.

So I'd like to end with a special thanks to Alan as a wonderful mentor. While I've been inspired by Alan as a computer scientist, visionary, and learner, I am most grateful that Alan has been a great friend and mentor.

Mitchel Resnick earned a B.S. in Physics from Princeton, and an M.S. and Ph.D. in Computer Science from MIT. He participated in a course that Alan taught in 1986, in the early days of the MIT Media Lab, and has been influenced by Alan's ideas ever since. Mitchel and his graduate students have been regular participants at Alan's Learning Labs.

His research group developed the Scratch programming language and online community, the "programmable bricks" underlying the LEGO Mindstorms robotics kits, the StarLogo massively-parallel programming language, and the Computer Clubhouse network of after-school learning centers for youth from low-income communities.

Currently Professor of Learning Research at the MIT Media Lab, he develops new technologies to engage people (especially children) in creative learning and design experiences.

Bran Ferren

AK—A Graphic Exposé

Let's see, what can I say about Alan Kay (*to save you fifteen years of getting to know him*)...

Great Friend

I'm not exactly sure why, but Alan and I seemed to hit it off the first day we met. It was at the second TED conference where we were both speakers, and we were collided by Richard Saul Wurman's desire to both have a really great extended dinner party and get someone else to pay for it. Our paths had never crossed prior to that fateful day. I had read about Alan, and we had many friends and acquaintances in common such as Marvin Minsky, Seymour Papert, Nicholas Negroponte, and Danny Hillis. Following his terrific presentation there in Monterey, we struck up a conversation that continued for a full four hours. For the life of me, I can't recall what we talked about. Come to think of it, I never can because whenever we're together, we seem to talk about everything—food, computers, acoustics, music, art, graphics, movies, books, friends, design, travel, talks, architecture, instruments, companies, games, entertainment, language, learning—you know, everything. Since that day, Alan, his remarkable wife Bonnie, my equally remarkable Robyn, and I have been great friends.



Valued Colleague

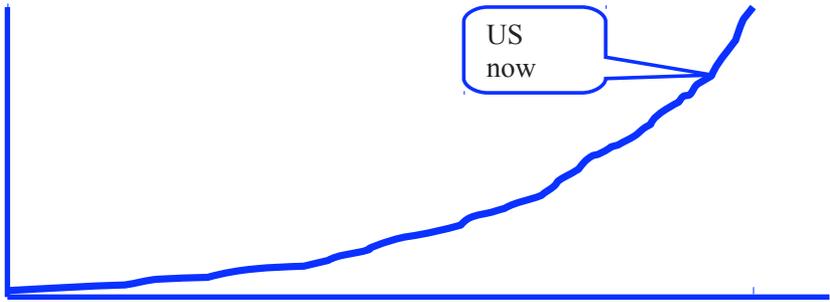
While I was running R&D at Disney, I persuaded Alan to leave a stint at Apple and come to Disney as one of our Senior Research Fellows. His job was to make us all smarter and to help find new ways to reach, inspire, and educate kids. His research group came along for the ride, and they extended their work on Squeak (a Mouse-friendly name for an OS if ever there was one, even though it started at Apple), which continues today at his Viewpoints Research Institute (housed at our AMI digs). Here's a picture from *The New Yorker*, definitively documenting that we fellows could all simultaneously stand together and be photographed.

National Treasure

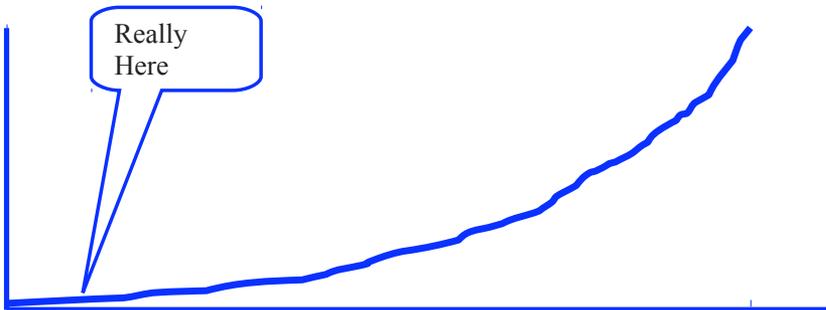
Alan is also one of those few people who says many smart things (and a few silly ones).

But rarer still, he's one of those incredibly scarce people who, every so often, says things that change the way we think about our world. Like Marvin Minsky, the Godfather of Artificial Intelligence who, when I was younger and less impressionable, said to me, "If as many people spent as much time and energy figuring out how to build an Intelligent Machine as they do saying it can't be done—we'd already have one." I won't exhaustively itemize here the numerous Alan Kay quotables which will undoubtedly infuse this tome as effectively as they have our culture.

But the Alanism that did it for me was the title of a class he taught for several years at UCLA—“The Computer Revolution Hasn’t Happened Yet.” Wow. For me, this was a seminal insight. You see, it’s easy to believe that given all of the computer-enabled technological wonders around us, our world has been forever changed by a mature technical and social revolution. The mere existence of 3.5 billion cell phones is evidence that it clearly has. But changed by how much? And how much change is yet to come? Our sensibilities tell us that, when you look at the graph of how far along we are in the life-changing, Computer Revolution hoo-ha, we must be about here. RIGHT?



WRONG. As Alan so acutely observed in the title of his class, we’re really way back here.

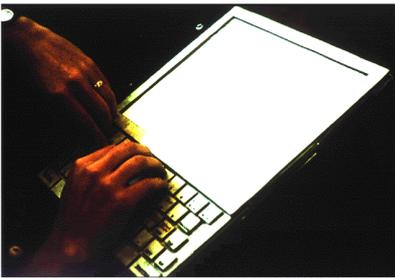


The revolutionary, world-changing fire that he and his friends helped ignite is truly just getting warmed up. Like the man said, the Computer Revolution

really hasn't started yet. All the fuss to date is only five minutes into the future festivities. Just wait until the kids, who (like my precious Kira) Alan's work will both empower and inspire, come into their own. It will be truly remarkable.

Authentic Computer Pioneer

Alan is so much more than a Computer Guy, it seems somehow wrong to pigeon-hole him with just a few notable techie examples, but what the hell.



The Dynabook

Now this was very cool—a prescient 1968 vision (I was 14) for what we would all be lugging around now. So did he predict it, or actually cause it to happen? Hard to know, but Alan both anticipated and included in his

vision the development of flat panel displays, low-power microprocessors, WYSIWYG, bit-mapped GUI screens, compact high-energy batteries, miniature high-density storage, and wireless communications. Even if he faked it, WOWSA! Tragic that he didn't also predict the invention of the hinge...

Smalltalk

Coding in machine language in 1970 when I got to MIT and getting really excited when I could get the computer I was working with to blink a light on its front panel, the revelation of OOP was a breath of fresh air. Like so many of the developments to emerge from within Alan's orbit, it seemed obvious that this was the only sensible way to do things. It's just that no one had ever done it that way before. Smalltalk was the first OOP language that I played with in school, having muddled with BASIC, LISP and FORTRAN. I was hooked.



Feeling empowered after MIT, OOP and I went our separate ways, I moved on to C and then FORTH, and while a few of my cute, little, and efficient programs actually worked, nobody could figure out why or how. Including me. Right around the time people were pushing me towards C and JOVIAL, I discovered that the ultimate solution to efficient programming was to acquire the necessary skills to hire other people to do it.



Overlapping Windows

Intuitive. Compelling. Efficient. Ground-breaking. And now pretty much ubiquitous across the realm. Overlapping windows changed the world.

BUT, I never did get the big deal with this one. The overlap makes it hard to open them for fresh air and to wash the panes, not to mention install bug screens.

Serious Book Dude

One can tell a lot about a man by what he reads, how much he has read, what he remembers, and which books he chooses to keep. In Alan's case the answers are:

- 1) Everything Alan cares about
- 2) Lots
- 3) Everything
- 4) Apparently, all of'em

And they're all cataloged!



Sci-Fi writers

Sci-Fi books

Sci-Fi movies

Sci-Fi conventions

Sci-Fi characters

Sci-Fi worlds

Sci-Fi themes

Sci-Fi aliens

Sci-Fi that's great

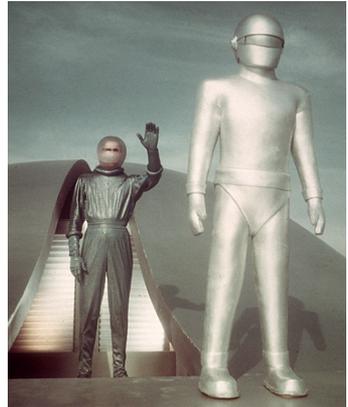
Sci-Fi that sucks

Sci-Fi Hi-Fi

Sci-Fi Wi-Fi.

Truly a remarkable event to observe. Every so often, I jump fearlessly into the conversation, only to be summarily corrected by one of them for some incorrectly resurrected memory or childish, superficial observation of an otherwise deep Sci-Fi construct that would have been self-evident if only I, too, had read “his” 17th unpublished short story. Suitably admonished, I glance at the other, and they nod knowingly with that “I tried to tell you” look. But, all in all, they humor we outsiders much like an alien might humor its human pet.

What’s noteworthy is that between them, they seem to have read every science fiction book ever written. (I’m not exaggerating.) Even more interesting to me is that throughout their lives, they have both independently reached out into the real world to meet and interact with many of their favorite Sci-Fi authors. The results of these interactions have influenced the lives of all parties



involved; clearly some of Alan's and his friends' inventions have affected the trajectory of Science Fiction.

And finally, let us not forget the influence Alan had on the making of the early CGI epic *Tron*—authored by Emmy Award-winning author (and actress... and musician... and producer... and artist) Bonnie MacBird, who was to become his bride.

And last, but certainly not least:

A Man with a *Really Big Organ*

What can I say—Alan is a talented and dedicated professional musician, with one terrific organ. Perhaps unsurprisingly, he caused this to happen over many years, by holding an organ builder captive in his library, not letting him leave until he completed a magnificent, two story high, North Germanic influenced baroque instrument.

Anyone who knows Alan is aware that the Big O is an emotionally, financially, and chronologically significant part of his life. So I thought it would be appropriate to share with you a picture of Alan playing (or at least co-located with) said organ.

I started Googling to find a nice snap or two of Alan + Organ. Seemed appropriate to do this using my Apple laptop, running Squeak, on that Internet Thingy that Alan helped create. So I fired up my trusty browser, set it to image search, entered:

Alan Kay's Organ

and clicked:

SEARCH.

This was the first image found.



Hmm... Now perhaps there's yet another something about Alan's life of which I'm unaware even after all of these years, but this just didn't look right. Perhaps Internet image searching is not yet perfect, and the Alan Kay / Kay Martin / Alan's Organ connection is weaker than one might otherwise hope.

So I persevered, and dug deeper into the returned results, and soon found this infinitely superior image:



Photo: Chris Ohyung

PERFECT!

I always knew Alan had chops, but didn't realize he was so well toned. Must be the makeup...

Cheers Alan!

To 70 more

Your pal and admirer,
Bran

(But I still don't get the big deal with the Overlapping Windows...)

Bran Ferren is a designer and technologist working in entertainment, development, engineering, architecture and the sciences. He was President of Research, Development and Creative Technology for the Walt Disney Company and, prior to that, President and Senior Designer for Associates & Ferren, the company that supplied the visual effects for Star Trek V.

Bran is an advisor to seven U.S. Government agencies and to the U.S. Senate. He has received numerous awards including Technical Achievement Awards and the Scientific and Engineering Award from the Academy of Motion Picture Arts and Sciences. He was nominated for an Oscar in 1987 for his work on Little Shop of Horrors. Bran has received awards from the New York Drama Desk, Los Angeles Critics Circle and the Maharam Foundation, and was the recipient of the Wally Russell Lifetime Achievement Award and the Kilby award for significant contributions to society.

Bran is currently a founding chairman at Applied Minds, Inc.

Betty Edwards

A Tribute to Alan Kay

I first met Alan Kay sometime around the early 1980's. Alan phoned me to tell me that he had read my book, *Drawing on the Right Side of the Brain*, which had been published in 1979, and he asked me to meet with him and some of his colleagues for lunch. We met at the old Synanon building in Venice near the beach. When I arrived, there was Alan waiting for me with that huge smile on his face. We talked through lunch, mostly about education, and that was the beginning of my long friendship with one of the few geniuses I have encountered in my lifetime.

The next thing I knew Alan had signed up his niece and his wife, Bonnie MacBird, to take my drawing class at California State University in Long Beach where I was teaching. That led to my friendship with Bonnie and many wonderful dinners, concerts, and just hanging out at the house in Brentwood.

Then Alan arranged for his whole group at Apple to take my intensive five-day drawing seminar at the old Presbyterian campgrounds in Pacific Palisades. Alan was there for every minute—never mind that he already had great drawing skills. He would sit there, intently listening to whatever I was trying to get across to the group, exploding with that great laugh at my often-lame little jokes. I finally asked him, “How can you sit through this stuff of mine when you’ve heard it all twice before and probably know it by heart?” His answer was quintessential Alan Kay: “Every time I listen, I learn something new.”

That is the mindset that Alan brings to everything. His mental fund of information is astounding. He devours books rapidly and voraciously, and unlike the rest of us, not only remembers what he reads but integrates it into a seemingly limitless, instantly accessible repertoire of expanding knowledge that includes many fields and is deep as well as broad.

Alan brings to mind the one other genius I have been fortunate to meet—Roger Sperry, who had the same quirky ability to be right there with you, listening intently, yet be off in some remote place at the same time. I cannot say that I *knew* Sperry, anymore than I can say I *know* Alan. One never truly knows a genius. But I recognize in them very similar qualities.

Alan is famous for many quotable quotes—for example, don't worry about what anybody else is going to do: "The best way to predict the future is to invent it."

Sperry was also famously quoted. Jerre Levy, who worked with him on the original split-brain studies, said that when Sperry came into the lab, he would ask three questions: "What are you doing?" That was fairly easy, Levy said. Second question: "Why are you doing it?" That was harder, she said, but answerable. Third question: "Why is it important?" That was the one that got them, she said. Everyone agrees that Alan, like Sperry, has been on a lifelong search for what is important—in computing, in music, in science, in literature, in art, and in education—perhaps especially in education, perhaps because inventing new ways of teaching important, deep ideas—as in *Squeak*—is right now predicting the future.

Another characteristic that Alan shares with Roger Sperry is generosity of spirit. Alan gives away his ideas, his time, and his knowledge, throwing it out with both hands whenever a receptive and worthwhile catcher is near. In similar fashion, Sperry met with me for weeks on end to make sure that the science was right in my then-unpublished manuscript. Generously, Alan invited me to his Learning Labs held at Apple Hill in New Hampshire, where he shared his thinking with our diverse crowd, and where I met Seymour Papert, Danny Hillis, Mitch Resnick and so many other remarkable people. Alan has truly

enriched my life with his friendship, and this project gives me a chance to say “Thank you.”

Betty was born in San Francisco and has lived most of her life in California, as near to the ocean as she has been able to manage. Of herself she says, “I am mainly a product (that’s what they call us) of UCLA, with a B.A., most of my M.A., and a doctorate in Psychology, Art, and Education.”

Her main field is Art. She has taught art at all levels—private classes for small children, some elementary and junior high public school teaching, five years at Venice high school, nine years at Los Angeles Trade Technical Community College, and thirteen years at California State University, Long Beach. Her publications include Drawing on the Right Side of the Brain, Drawing on the Artist Within, and Color.

Betty is currently retired and living in La Jolla. She’s thinking about another book, working on a new garden and looking forward to the next visit of her two granddaughters.

Bob Lucky

Portraits of Alan Kay

Every now and then I look back on my career and wonder: suppose I had gone into a different field? But I quickly dismiss the thought, and consider all the blessings that I have experienced as a technologist. The foremost thought on my mind is always the realization that I have had the experience of knowing and working with the best and the brightest that our society has to offer. The people I count as friends have created and shaped the world of technology that is such a part of everyday life today. Without their contributions the world would be a much less interesting and connected place. I'm proud of what they have done, and I feel a shared sense of participation in these accomplishments.

Among the friends I am so proud to have known, Alan Kay has a special place of honor. Long before I had met him in person, I knew his name as a legendary figure. I'm not even sure that I would have identified him as the creator of Smalltalk; in my mind he was the Apple guru, the source of their mystique, and one of the handful of geniuses that had made Xerox PARC the model of innovation that it had been.

I first met Alan through a series of meetings that took place in the late 1980s. I remember particularly a series of "Multimedia Roundtables" put together by Martin Greenberger of UCLA. It was a heady time, and the operative word was "multimedia." Martin brought together a unique blend of people to discuss the future ramifications of multimedia technology. As I look back

on it now, those early multimedia discussions seem to have been misguided, but at the time we were filled with a tremendous sense of excitement and promise.

Alan was one of the bright stars at the Greenberger roundtables, among the constellations of stars there. We all had the feeling of being special people at a special time. We had this marvelous new technology, and we were the chosen people who could decide where it went. I don't know if there are any similar meetings now; perhaps they were only possible at that particular juncture. Greenberger was able to put together a real who's who from cross-disciplinary fields surrounding the technology in multimedia. There were participants from the computer industry, software, telecom, human factors, education, art, content, and media, to just name some.

I remember one of these meetings was held in Santa Monica, where we took the occasion to visit the offices of The Voyager Company. It was Voyager, and its founder Bob Stein, that best epitomized the thrust of the Greenberger meetings. The Voyager office was on the beach in Santa Monica (they later moved to New York), where young people in blue jeans worked with Apple computers at large CRT displays creating interactive disks of Shakespeare plays. I felt a pang of jealousy—I thought that's what I'd like to be doing.

We discussed how interactive CD-ROMs would revolutionize media. No longer were we confined to sequential media, now we could navigate our own individual paths through content. What potential there was! But although I went away from the Greenberger meetings with sample disks of interactive content, I don't think I ever spent any substantial time exploring any of these sample disks. I should have known this wouldn't work.

We didn't realize at the time the extent to which the Internet would become the big "CD-ROM in the sky", or how the "ROM" (Read-Only Memory) part would become irrelevant. One thing we were right about, however, was the idea that content in the future could and would be created by amateurs. But at that time the Internet was a fledgling research network accessed by insiders with modems, and delivery of true multimedia was problematical.

Alan was a thoughtful, even philosophical, contributor to the Greenberger meetings. He was classically trained and passionate about education. That was apparent in these early meetings, and I have seen this passion and wisdom many times in the years since. It was 1989 or 1990 when I was sitting at my desk at Bell Labs and got a phone call from Alan. He had never called me before. It was a short conversation, and I got the impression he was in a car or a meeting. He said that he and someone named Rich Schroth from the Index Group in Boston were setting up a series of meetings, and that I would soon get a call from Rich about becoming part of this undertaking.

Shortly thereafter Rich Schroth called me and described a program they envisioned, called Vanguard, for keeping business managers informed about evolving technology. He invited me to join. I quickly decided that if Alan Kay was involved, I wanted to be part of this.

So it came to pass that we had our first Vanguard meeting. The idea was that Alan Kay, Nicholas Negroponte, David Reed, and I would give regular talks at these meetings to a set of paying clients. I think that at our first meeting there were seven such clients around a modest table. We dutifully put our viewgraphs on the machine and gave our standard talks. But it was immediately apparent to us that this wasn't going to work as an ongoing solution. After you've given your standard talk to a particular audience, what do you do for an encore to this same audience?

I remember a conversation that I'd had in jest with a friend in the early days of my career around just this question. We postulated that you could prepare a state-of-the-art presentation that you would give at MIT. Then you had two choices. If you were going to talk at MIT or a similar elite university again, you would have to upgrade your talk with substantial new material. Alternatively, you could keep the same talk, but you would have to progressively downgrade your audience. So you could take your existing MIT talk, give it later at a lesser university, then at a small community college, then the local Lion's Club, and then start working the developing nations, and so forth.

So the original conception of Vanguard was a non-starter. Well, it could start, but couldn't continue the same way. Instead we set up meetings with a slate of invited speakers, and Alan, Nicholas, David, and I became interlocutors, stimulating audience interaction with injections of wisdom, experience, or arguments into the presentations. Rich Schroth and his associate, Chunka Mui, arranged the meetings and, for the most part, invited the speakers. The program was officially named Vanguard, and was marketed by a team from CSC, the parent company of Index.

Alan was a crucial factor in the early days of Vanguard. His fame lent credibility to the program, his periodic talks were uplifting, his interjections in discussions were almost always insightful and provocative, and his network of friends was essential in bringing in speakers. I think the best talk that I have heard in all the years of Vanguard was at a meeting in Los Angeles when Alan brought in Quincy Jones as a speaker.

At one Vanguard meeting in London, Alan offered to set up a dinner for me with a couple of famous people from the local area. He gave me some choice, and I settled on Douglas Adams and Richard Dawkins. Douglas Adams was the author of *The Hitchhiker's Guide to the Galaxy*, among other best-selling books, while Richard Dawkins, the evolutionary biologist from Oxford, had written *The Selfish Gene*, *The Blind Watchmaker*, and other philosophical scientific books.

We met in Douglas' flat in Islington and walked to a local Italian restaurant. The evening was a memorable one, full of philosophy, wisdom, and wit. Every meal should be like that, I thought, but of course it can't be so. Through Alan's introduction I got to know Douglas Adams, and at other meetings went with him to Douglas' London club and to a new company that he had started, working on video games, where Alan and I were supposed to be advisors.

The point for this little essay is Alan's network connectivity—Quincy Jones, Douglas Adams, Richard Dawkins—a rather unlikely trio for any one person to know personally, and this is but a sample. If I could draw a social

network diagram of the world, Alan would be a major node. If you know Alan, you can skip through the six degrees of separation and leap right to the top layers.

Alan and I have been to many meetings together since that first tentative Index meeting many years ago, and though Alan continues to attend and contribute immensely, he is sometimes reluctant to travel. I remember a quip that he made about travel, that he would much prefer to be “FedExed” to these meetings!

A Portrait of Alan

How would I envision a portrait of Alan Kay? First there is the classically trained philosopher-scientist—perhaps, had they lived at the same time, a Holbein portrait of Isaac Newton. Then, never far from the surface, the musician and musicologist—here I think of the mathematical Bach, rendered enigmatically by René Magritte. Digging deeper, we have the software pioneer—and not merely a great programmer or hacker, but in the seldom-meant true sense of the word—computer scientist. I imagine something between Alan Turing and Fred Brooks. Here I would picture a blockish fusion done by Salvador Dalí, where you have to stand back and squint your eyes to see the forms emerge—just the sort of effect Alan likes.

All of these dimensions, and more, come out in any talk that Alan gives. Not only does Alan care passionately about issues like education, software structure, and open-source development, but he has a gift for words. He turns phrases effortlessly, and leaves me grasping for metaphors. I am trying to think of any other friend or acquaintance who has an actual quote that people know. “The best way to predict the future is to invent it,” is widely known and oft-quoted. I sometimes muse about how quotes like this come about. I think about all the talks and writing that I have done through so many years, and there is not a single quotation that has lasted. Whatever it is about quotations, I haven’t got it. Alas.

The best way that I can depict Alan with words is to use his own. I have my notes from a talk that he gave at a Vanguard conference in April 2001 in Los Angeles entitled *To Have a Future You Have to Get Past the Past*. This was a meeting focusing on software, and Alan was introducing the audience to the software he and his group were developing called Squeak, an open-source implementation of a Smalltalk programming environment.

Alan put up his opening graphic and called attention to the fact: “This talk is being given using Squeak.” Later on he returned to emphasize this point: “Notice that there’s no PowerPoint in this presentation.”

Alan challenged us with an optical illusion (a tabletop of apparently changing size) and then he put the illusion into context with a quote from the Talmud: “We see things not as they are, but as we are.” This snippet is so much like Alan—the use of illusions, metaphors, quotations, and philosophies from the classical literature.

Before delving into a deeper talk about software, Alan threw out some of his casual, yet memorable, one-liners:

We’re basically cave-people who now have atomic weapons.

How many people have had an epiphany? Most of them are mediocre, but they’re important—gifts from the heavens.

Learning is close to creativity.

Now we get down to Alan’s philosophies about software development, contrasting the development of Squeak with conventional program development:

- early binding vs. late binding
- data structures vs. only objects
- differences vs. similarities
- big vs. tiny
- system build vs. changes-per-second

- programmed by end-users vs. programmed by experts
- functionality outwards vs. user-interface inwards
- few platforms vs. as many as CPUs
- thousands of people vs. tens of people
- applications vs. gatherings
- waterfall development vs. ship the prototype
- free and open vs. money

In the course of discussion of these differences, Alan referred to some of his personal experience and biases about software development:

We had a saying at Xerox PARC: Error 33—putting someone else’s software in your critical path.

Most of software today is glue code to get around the big packages.

My problem with XML: don’t send something that has to be interpreted at the other end.

Zen and the art of computing: The music is not in the piano!

I thought about this Dynabook as a musical instrument, and its music is ideas.

The revolution in printing lagged the invention of the printing press by 150 years.

The computer revolution isn’t going to happen until you get people who are literate in this new medium.

All the rhetoric about silver bullets is irrelevant.

Is “software engineering” an oxymoron?

Squeak was done by a small team of about a half dozen people.

This was 2001, and Alan couldn't have known then that five or six years later he would take Squeak and get involved in Nicholas Negroponte's creation of One Laptop Per Child, which perfectly coalesced Alan's interests in education, philanthropy, and software development.

As I write about this particular talk, I can visualize Alan as I have known him through the years. It has been a pleasure, an education, and an honor.

Robert W. Lucky received his Ph.D. in Electrical Engineering from Purdue University in 1961.

Early in his career he invented the adaptive equalizer, the key enabler for all high speed modems today. Many engineers know him from the monthly columns he has written for Spectrum Magazine offering philosophical, and sometimes humorous, observations on engineering, life, and technology. He is a fellow of the IEEE, and a member of the National Academy of Engineering and both the American and European Academies of Arts & Sciences. He has led research laboratories at Bell Labs and Telcordia Technologies, where he was corporate vice president of applied research.

Bob retired from Telcordia in 2002 to devote time to advisory boards, studies, and consulting. He is currently chairman of the Fort Monmouth Revitalization Planning Authority and a member of the Defense Science Board.

Greg Harrold

Greg and Alan conspire to create a wonderful new pipe organ

There are times when a fortuitous confluence of events happens that results in enrichment of mind, body and heart. Meeting Alan Kay was one of those events in my life. When he asked if I would build an organ for him, it came at a point when I had no new work. Even before the organ was finished we had agreed that it had been a project well worth doing.

For three years, since 1984, I had been working on my Spanish, or Aragonese, organ—Opus 11 for the University of California at Berkeley, modeled after organs built in Zaragoza, Spain, around 1700. Lawrence Moe (professor and organist at UC Berkeley from 1957 to 1988) and I had travelled to Spain to do original research in preparation for Opus 11.

I was busy working in my shop one day in July 1987 when Alan paid me a surprise visit. Nicholas, my Yorkshire terrier, was my only shop companion during that time. Alan and I had never met, nor did we even know of each other. Tom Harmon (professor and organist at UCLA from 1967 to 2002), a long-time friend of mine, had given Alan some lessons and suggested he ought to visit my shop.

In his typical enthusiastic way, Alan volunteered to help me with my Spanish organ. It needed some forged iron parts in keeping with its period. Amazingly, Alan bought a horseshoe forge and an anvil at a nearby farrier and blacksmith's supply shop. Having never forged before, Alan taught himself in his

backyard and then continued forging at my workshop. He long had an interest in trying his hand at organ building but I doubt he thought forging iron would be his first task!

I built my Spanish organ virtually alone—not only the case but the pipes, bellows, keyboard, windchest, action, voicing and tuning—with help from Dennis Rowland (carving), Lawrence Moe (painting and gilding), Tom Harmon (working on reed blocks), Alan (forging the iron), and a few other volunteers along the way.

The premiere recital on the Spanish organ was given by Lawrence Moe on October 7th, 1988. Alan was at the recital and asked me then if I might be interested in building an organ for the house he and his wife Bonnie MacBird shared in Brentwood near Los Angeles. I immediately agreed, although with a mix of excitement and uncertainty, sensing already that it would be a project unlike any other I had done.

The idea of the new organ had just become reality, called affectionately Opus 14 after its place in my work list. It was stimulating for me to work with someone who is passionate and involved, like Alan. I think this made my work better. We started brainstorming ideas about the organ right away. He would show me a stoplist or drawing; I would counter with my own. We kept going back and forth this way, becoming more familiar with each other's way of thinking, listening to recordings in the shop and talking about all kinds of organs.

Alan told me how his interest in Baroque organs got fired up when he heard the 1957 Flentrop organ that E. Power Biggs bought and had installed at Harvard's Busch-Reisinger Museum. The Flentrop organ was one of the earliest of this type in the United States. Along with a few other new organs, it influenced a generation of American organbuilders to make instruments more closely based on antique organs.

It turned out that Alan and I shared a passion for seventeenth century organ music. We kept returning to the organ works of Dietrich Buxtehude (1637–1707), one of the greatest organists of his time and highly renowned for his improvisations—so much so that the twenty-year-old J. S. Bach (1685–

1750) walked 280 miles from Arnstadt to Lübeck to study with him for four months. On returning to his church job, Bach's organ playing demonstrated how well he had absorbed Buxtehude's teaching—to the consternation of his conservative congregation and elders. Alan truly loves playing the organ music of J. S. Bach, yet the organ music of Buxtehude and his peers continues to beckon and intrigue him on so many levels.

Improvisation and composition were traditionally required skills for any organist. The organ music of Buxtehude and his peers were ideal showpieces for this, particularly the *Prælude*. Alan, who has a lifelong love of jazz and rock-&-roll, explained to me that he found similarities among these improvisatory musical styles. The free and loose riffs that typically open and close a *Prælude*¹ would be familiar territory for a rock-&-roll or jazz musician.

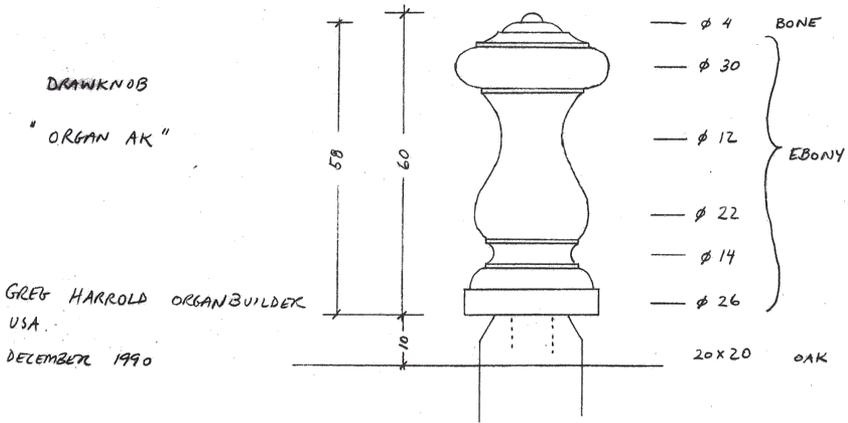
Early on it seemed obvious to us that the late seventeenth and early eighteenth century Hamburg school of organ building would serve as the best model for the stoplist, pipe construction and sound of Opus 14. These organs were a mature expression of the middle Baroque period.² They allowed the organist to display magnificently the emotional affects, so important in Baroque music. Most importantly, both Alan and I loved the sound.

The Hamburg school Baroque organ has influenced many contemporary organ builders.³ My organ building colleagues and I tried to get closer to the ancient pipe making techniques, while at the same time making organs with the cases, carvings, keyboards, pedalboards and wind systems according to our own preference. Very few have bravely tried to build everything faithfully in the old style.

¹ A *Prælude*, in this context, is a type of keyboard toccata in which free, improvisatory sections alternate with sections in imitative counterpoint.

² In music, the Baroque period is commonly divided into three parts: Early Baroque (1600–1654), Middle Baroque (1654–1707), and Late (or High) Baroque (1680–1750).

³ The early music revival began in the early twentieth century, a reconsideration of Renaissance and Baroque instruments and performance. At the same time the *Orgelbewegung* (or Organ Reform Movement) started, inspired by the great 1699 Hamburg *Jacobikirche* organ built by Arp Schnitger (1648–1719).



At first I did want to build the organ in the eighteenth century way, with a painted case, voluptuous moldings and florid carvings. Having enjoyed doing that on my Berkeley Spanish organ I figured it would be fun to try again. But this wasn't a good fit for Opus 14. I returned to my own personal heritage of freely adapted inspiration, a choice fully supported by Alan.

The stoplist for Opus 14 was based on the 1693–94 Arp Schnitger organ (built for the *Waisenhauskirche* in Hamburg) but *after* its move to Grasberg and the addition of a new 16' *Posaune*, along with other modifications, by Georg Wilhelmy in 1788. This was one of Schnitger's city organs, having an elegant sound and responsive key action suitable for virtuoso playing.

With the freedom to design a contemporary organ incorporating elements of the Hamburg school, I still wanted to keep everything harmonious. Alan and Bonnie had envisioned a serene and simple kind of casework, not active and dynamic like the Hamburg organs which had projecting central and side towers. They liked the sample of hand-waxed quarter-sawn spruce I showed them, so that's what I used for the case.

When planning the organ case I came across an excellent model in the North German organs of the middle Renaissance.⁴ Making another connec-

⁴In music, the Renaissance period is commonly divided into three parts: Early Renaissance (1400–1467), Middle Renaissance (1467–1534), and Late Renaissance (1534–1600).

tion to Buxtehude, I freely borrowed elements from the 1477 case of the *Totentanzorgel*,⁵ one of the organs he played at the Lübeck *Marienkirche*. The familiar five-part arrangement of pipes, modified so that the large pipes were central, the carved panels below the pipe feet, and the refined coved cornice molding were all used.

During the following years I drew up the complete designs for Opus 14. Alan and I agreed to let other specialized shops do the heavy, exhausting work of construction so, as each design was ready I would send it off to them. There was some time to spare while waiting for the finished parts, so I was able to fit another small organ project in.

Tom Harmon had bought my Opus 1 with the intention of having me enlarge it for his house. One of his friends, Paul Woudenberg, who led a small chapel in Pebble Beach, asked if he knew of an organ for sale. Tom and I agreed that Opus 1 would be perfect. I transformed my one-manual organ into a two-manual and pedal organ, renaming it Opus 12.

Furthermore, I used Opus 12 as a little laboratory for Alan's organ. The upper part of the case had a similar outline for testing visual proportion. The keydesk was identical to verify playing position and touch. Half of the pipes had already been voiced in Opus 1, making it easy to voice the rest of the pipes in my shop. Alan and I liked having a partial preview to study. One outcome was that Alan designed a custom concave-radiating pedalboard to swap with the flat pedalboard I like to use.

On a heroic instrument such as Schnitger's organ at Hamburg *Hauptkirche St. Jacobi*, the drawknobs extend too far from the player to be drawn while playing. Consequently, during major events involving the organ, someone would stand on each side carefully and quietly changing stops as needed. Alan asked me to include a motor to work each stop so that he could imitate the teenage apprentice changing stops. He planned to have a flat display that he

⁵ Located in the *Totentanzkapelle* of the Lübeck *Marienkirche*, but destroyed by bombs in WWII. The large organ at the *Marienkirche* was also extremely significant. So many important cultural artifacts have been lost to wars that it makes my heart sore.

could lift from behind the music rack. Above the pedalboard, on the left side, are two levers—one for *next*, the other for *last*. This was all to be directed by a HyperCard stack, which Alan would program for each piece, and the stops could still be pulled manually if desired. Once the organ was done Alan discovered that stop changes on Opus 14 were so easy that he didn't need his "registration assistants" after all.

Finding inspiration for the carvings took us in an unexpected direction. There is a very old, venerable and remarkable organ in Bologna, at the church of San Petronio. One of the most important and valuable organs in the world, it was built between April 1471 and August 1475 by Lorenzo di Giacomo da Prato. It has ten stops, which are played from one manual. The pedal keys are always attached to the lowest keys of the manual. The largest pipe in the front is a 24' F. As a credit to its builder and to the commune of Bologna, this organ has let fine musicians make music there for over five centuries! There is another organ directly across the choir from the Lorenzo da Prato organ, built by Baldassarre Malamini in 1596. Everyone calls it the "new organ." I have been to Bologna to examine, listen to and appreciate these organs, and greatly enjoyed describing my experience to Alan.

A man called Ercole di Francia (in the original documents) did the carvings on the Lorenzo da Prato organ. These carvings so appealed to us that, like many before us, we decided to use them as a model. The carving designs consist of roundels of various sizes, each with a different pattern. The bottom edges of the pipe-shades have half-circles above each front pipe. This design was a common theme in the fifteenth century. It allowed the carver or painter to show skill by making the roundels different but complementary. The *Totentanz* organ at Buxtehude's church had some carvings like this.

Dennis Rowland, an extremely gifted carver, agreed to make the carvings for Opus 14. He made the splendid carvings for my Spanish organ at UC Berkeley, and has done many magnificent carvings for organs made by Taylor and Boody.

After receiving the sample carvings from Dennis, Alan had a fascinating idea. He asked if Dennis would make some slight tweaks in the design. As a result, the carvings contain two iconic messages; one is profound, and one is very profound. Even Dennis doesn't know what these messages are, despite making the carvings, nor do I. Alan was clever about that. Alan promises a bottle of fine champagne to anyone who can discover what these messages are.

In time the organ took shape in my shop. I assembled as much as I could, considering my shop's tight quarters. Everyone was getting so jazzed about being able to see the organ in Brentwood!

On January 17th, 1994, at 4:31 in the morning, a strong earthquake jolted me awake. I dreaded that everything in my shop would be tossed about, including the organ. Alan, ever the optimist, went to the shop to check, and called me to tell me the organ had survived.

My shop ceiling, which sloped from 14 to 18 feet, was not high enough to set up the organ completely. I stood the *Posaune* 16' pipes upright on the floor, attached to their support racks. The lower part of the case was fully assembled up to the Great windchest with all of its pipes in their racks. The upper part of the case was on the floor, with the front pipes and carvings in place.

When I arrived at the shop I discovered that some of *Posaune* pipes had fallen against the staircase, but were easy for me to fix later. Most frightening, though, was that a tall shelf of my tools had fallen over. It had missed the upper part of organ case by only half an inch! Some of the tools had flown out and hit the organ front. There was a chip in the woodwork, which I was able to repair.

A few front pipes had minor dents. My recollection is that Alan asked if I could remove them. I said the repair would probably look worse than the dent, so we agreed to leave them.

Craftsmen, tradesmen and artisans have often left a "mark of meaning" on their work. Leaving these small marks on the organ was a way to acknowledge our imperfection and our humility to events beyond our control.



The quake gave us good reason to move the organ from my shop. Fortunately the new music room at Alan and Bonnie's house was almost done. I disassembled the organ, packed up all its parts, and, with friends volunteering, trucked it to Brentwood. We brought everything into the music room, now called Contrapuntal Performances Hall, nearly filling the entire floor.

I set up the organ during the next month. We attached steel and wood framing to the back and bolted it to the building as security against future quakes. In March I was able to put in the front pipes. Everyone involved was thrilled to see it, especially after the earthquake scare. Everything was coming together now after five years of work on the organ and the hall.

My voicing jack is a little organ without a case. I can stand at its keyboard and easily work on the pipes, which are set at eye-level. With it placed close to Opus 14, I was able to start voicing the pipes.

I started voicing the Octave 4' on March 17th, 1994. This stop helps me get my bearings on the room acoustics, relative loudness, and timbre. It was common for Schnitger and his contemporaries to include many older sets of



pipes when making a new organ, or when rebuilding an older organ that had fallen into disrepair. This link to the past adds an extra level of depth to the sound. I tried this by imagining the Octave 4' was from an earlier organ and was showing its age.

On April 5th I transferred the pre-voiced and rough-tuned Octave 4' pipes from the voicing jack to their place in the organ. That became one of the magical moments in the organ's history. Alan was surprised, I think, and thrilled to find he could actually play almost anything and make it meaningful. Words can't easily describe the experience. As Alan often says: "Talking about music is like dancing about architecture."⁶

Ironically during this period Alan was swamped with work—traveling, teaching and consulting—and missed being home as much as he wanted. He had hoped to be more involved at this stage, especially the voicing. When

⁶The often-quoted "writing about music is like dancing about architecture" has been attributed variously to Laurie Anderson, Elvis Costello, Clara Schumann, Frank Zappa, and others.

he was not away, though, he would test each newly voiced stop during the evening after I left, and in the morning before I arrived. It was a treat to find his handwritten notes on the organ bench or music rack telling me what he thought of the latest progress. I would answer back with notes of my own.

Alan thought it would be worthwhile to test the room's acoustics with an "in-progress" recital. Played on November 19th, 1994, by John Butt⁷ it was dedicated to Alan's mother, Katherine Kay. I always enjoyed seeing her whenever she visited Alan. She was a remarkable person, and an enthusiastic supporter of the organ project.

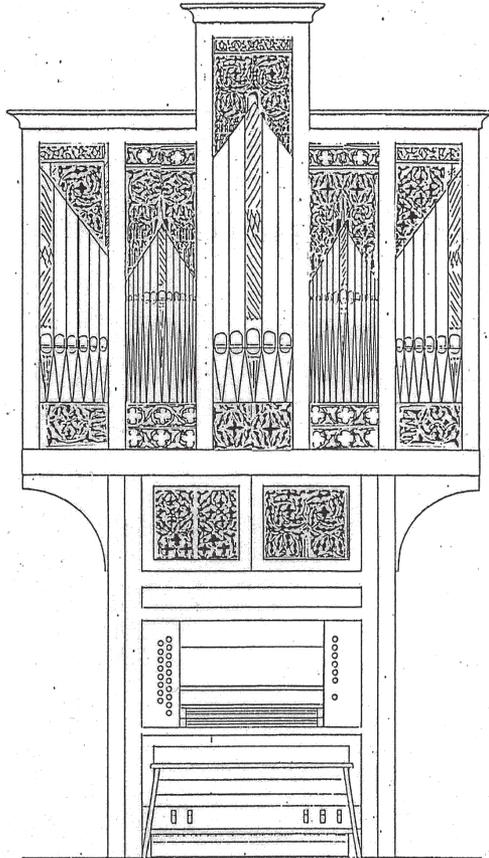
Eventually all the pipes were in the organ case, tonally refined and all in tune. When there is a particular combination of Opus 14, an organist who understands it, and certain music, the effect is profound. I had this experience with antique organs in Europe. In our brainstorming sessions, Alan and I had noticed something about the sonority of the Hamburg school organ that overwhelms the emotions.

When researching antique organs in Europe I have experienced so many amazing things. At times it felt as if a composer, or an organist, or an organ-builder, from the past was right there with me. There was an immediacy and intensity of sharing a connection across time. Outside of my dreams this comes closest to a form of time travel. Then, to be able to share even some of this experience with others listening to Opus 14, has been deeply satisfying—albeit a bit mysterious.

Even as a master organbuilder I still find it hard to describe to others certain aspects of my work, particularly why I make certain decisions in design, pipe scaling or voicing. A kind of "symphonic thinking," or maybe "baroque organ thinking," overtakes me when I work, guiding and prodding me forward. A very profound thing happens once in a rare while, resulting in an organ that seems to give back even more than I put into it. As much as I tend to prefer process as opposed to the completion, it always surprises me when something that was once a vision in my head becomes corporeal.

⁷Musicologist and organist at UC Berkeley and successor to Lawrence Moe.

Working with Alan was, and is, a pretty darn cool experience. He's really easy to be with, unpretentious, deeply engaged and focused. The differences between how he and I think, remember and communicate are stimulating. When I describe my strong ideas, passions, techniques and theories about organ building he absorbs it all easily, then applies his mental alchemy to come back with his own perspectives, which sometimes make me rethink my positions. We have been engaged in an ongoing dialog about and around Opus 14, what we learned during the project, what it teaches us now, and what questions it makes us ask.



Greg Harrold came to his profession almost by accident. Job seeking after high school he happened upon a listing for an organbuilder at Abbott & Sieker in Los Angeles. He was hired as their pipe maker in July, 1974. Within six months he had decided that he could make it his career. During the next three years, working in his spare time, he created his Opus 1 to learn how to build a complete organ.

Greg expanded his knowledge and experience in 1978, working with John Brombaugh in Eugene, Oregon, and taking his first trip to Europe. He became even more convinced that making new organs based on techniques used in antique organs would be his method.

Greg opened his own workshop in West Los Angeles in 1979, with a commission to repair one of the antique organs in the collection at UC Berkeley. His Opus 11 at UC Berkeley, finished in 1988, was built accurately in the style of organs from the Aragón region of Spain in the late seventeenth century. The rest of the organs he has built were strongly influenced by seventeenth and eighteenth century northern European organs.

Working mostly alone, Greg continued to design and build organs for twenty-four years. He was forced to close his shop in 2003, due to an ongoing illness.

Greg's unique combination of artistry, historical expertise, fine skills, and refined voicing talent shows in all his work. He maintains an active interest in organ building and is a regular attendee at the recitals held at Alan and Bonnie's home.

Quincy Jones

A three-sixty human being

Steve Ross called me one day. He was my mentor, my guru, a father figure—one of the greatest men I met in my whole life. He said, “I want you to go to Atari and see what’s going on over there.” I went. It was astounding. I met Alan and he showed us all around the place. That was the first time we’d met, about thirty-five years ago, or something like that. (The next thing I knew, Alan was an Apple Fellow.)

We used to spend a lot of time together. Back in those days they had organizations like E.A.T., a group for Experiments in Art and Technology. People from different genres were trading wild ideas, with people in areas they weren’t really familiar with. Paul Beaver was alive then too, and was very influential in my life. He was one of the high-tech guys for the musicians: he added touch sensitivity to the Clavinet, like Stevie used in *Superstition*, and the Novachord, and altered different things for us to use. He showed me the first Fender Rhodes piano and turned me on to the Moog synthesizers—the first one the people ever heard, which was on *Ironsides*. This was two years before Wendy Carlos, then-Walter Carlos, did *Switched-On Bach*.

In 1953 Leo Fender brought us the first Fender bass. Check back a little in USA TODAY and you’ll see on the cover a portrait with Bill Gates, Steve Case and myself. They asked each of us, “Which piece of technology changed your

field the most?” I said the Fender bass, because without the Fender bass there would be no rock and roll, there would be no Motown. The electric guitar was created in 1939 and it didn’t have a friend until it got the Fender bass, and the two of them together made an electric rhythm section that gave us all the basis of rock and roll.

I used to be Vice President of Mercury and Philips, and we used to do our executive training at Eindhoven (home of Philips Electronics). At Eindhoven they had 4,000 scientists working on experimental things like eight-track machines, the first audio cassettes, and the first laser videodiscs. This was the early 1960s, way before Atari.

They used fourteen of us as guinea pigs for all the synthesizers—the YC-30, the YC-45—to find out what we thought. I remember that Robert Moog asked me why the black musicians didn’t like to use the Moog synthesizer and I said, “Bob, it’s probably because it is electrical tone that is shaped by the instrument. You have an electrical signal and you can do one thing to it and make it a saw tooth, which is a little rougher sound, or a sine wave, which is smooth, but it doesn’t bend—and if doesn’t bend then black musicians aren’t going to deal with it because it can’t play any funk. It’s got to have *that* feeling.” So he immediately invented a pitch bender and a portamento attachment, and Stevie Wonder did four Grammy-winning albums after that. That’s when he did *Innervisions* and *Songs in the Key of Life*.

To have seen it is astounding: the way we rode the technology and music together, on a parallel path, all the way through to today with ProTools, Cubase and even Auto-Tune. I feel very fortunate to have traveled the road from monophonic discs to digital DAT.

Knowing Alan

I think of Alan as America’s Einstein, I really do, and a very *human* human being—a “three-sixty” human being. Our major bond I think is he played piano and pipe organ, so we had that in common. His brain is from another

planet: he was trying to tell me about the Internet, thirty years ago, and I thought he was smoking kool-aid!

I've heard that binary numbers started in Egypt around 3500 BC. So permutations of zeros and ones have been around a long time—and the north of Sudan and Egypt with its papyrus, etc., really was the cradle of civilization. I was fortunate to travel there when I was about twenty-three and I've been addicted to travel since then. I go to Cambodia, places like Angkor Wat, and all over the world; and I feel at home everywhere in the world. I try to learn the language everywhere I go.

As an Apple Fellow, Steve Jobs told Alan to come at him as if he were a competitor using all of his resources: technical, financial, scientific, everything. Alan went out and created the Mac I and the Mac II, with his overlapping windows and icons. I read the book *Organizing Genius* about how those guys at the Palo Alto Research Center always tried to find somebody better than they were—rather than find somebody not as good so they could continue to be the big-wig. Alan was like that. We'd go to executive conferences with Bill Gates and he'd say, "Alan will you please take over?" Alan talked me into doing a speech at MIT one time and I didn't get it. I'm looking at Ray Kurtzweil, Marvin Minsky, Nicholas Negroponte, and all these geniuses, and I thanked God I was talking on a subject where I really knew what I was talking about rather than trying to talk about something in their field.

Alan was always, and always will be, a great friend. I go to his house and listen to great bands like Phil Norman and his Tentet—it's just a miraculous and beautiful bond and friendship that I treasure very much. His wife, Bonnie, is great; they are incredible human beings.

Music and Science

Knowing Alan opens up your mind—opens it up so your mind is ready to accept any possibilities, and how constantly not to be afraid to think outside of the box. I've been like that all my life with jazz, but he did it on a scientific level

and made me just as curious about the science as I was about the music. Because music has a science to it too. It's the only thing that engages the left and right sides of the brain simultaneously. There's intellect and emotion, and emotion drives the intellect—that's the science. You have to have the science. You have to understand the science, backwards, to be a good composer, orchestrator, etc.

Mathematics and music are the only two absolutes. They found that people who are close to music learn other things a little faster because they are constantly using the left and right sides of their brain. Most people are making the choice between one or the other. Many computer people are also musicians. I noticed right away when Alan took me up to Silicon Valley how many of those guys had a musical background; the two fit like a glove. We have the same kind of mindset; even though he's a genius in his field, the mindset is the same. I've always loved being with Alan, as with Marvin Minsky and his wife Gloria. We'd meet in places like Dublin, Las Vegas or even Montreaux, and it just fits—we don't have to strain at all.

Managing Creativity

I didn't know what the word "producer" meant when I first heard it. I started very young—thirteen years old—and worked in night clubs. Later on I went to study with Nadia Boulanger and she used to kid me and say, "You jazz musicians shack-up with music first and then you court it and marry it later." And it's true. You go on to study counterpoint and retrograde inversion and the techniques of harmony and so forth and really get to understand orchestration. Frank Gehry always tells me, "If architecture is frozen music, then music must be liquid architecture." And it is. When you are writing for woodwinds, brass and strings, in a 120-piece symphony orchestra, it's architecture—emotional architecture.

And you have to *write* that. You are dealing with voicing and harmonic structures, and so forth, and you can't take a chance of going into the studio with ten or fifteen horns and tell the guys, "you play what you want to play." It

doesn't work like that. You have to sit down a few days before and figure all that out. That's what being an arranger and an orchestrator is, and that's what I've been doing since I was thirteen years old—and will continue to do until the day I die.

But I always leave room and solicit the musicians to add their own personalities in the parts that they can apply that to, so that you have their personality involved in it too. I learned a lot of that from Duke Ellington. I learned it from everybody—Ravel, Stravinsky, Rimsky-Korsakov. Ravel was influenced by Korsakov but he had a virtuoso orchestra, where everyone was a virtuoso soloist. It is very complex. I remember when I first started to do studio sessions after I left Lionel Hampton's band at a very young age, something like twenty-two. I was in New York and had to do all kinds of sessions with Clifford Brown, Art Blakey, and Big Maybelle too. We did *A Whole Lotta Shakin' Going On* four years before Jerry Lee Lewis even heard it. I worked with Count Basie and Tommy Dorsey, and I just wanted to write music more than anything in the world. It was a very eclectic group of people I was able to work with.

Back then it was not such a big thing to make a pop record. Little by little I would get the engineers and musicians together—all the things the producer does, but I didn't know that at the time. I found this sixteen-year-old girl and made *It's My Party* with her. *Sunshine, Lollipops and Rainbows*—that was Marvin Hamlisch's first song; he was fifteen years old. I had eighteen hits with Leslie Gore. We did all this stuff, and it turns out that's what a producer does, but I didn't know—and I didn't even get paid for Lesley Gore's hits! As I started to get serious at producing, everything I knew as an arranger just morphed itself into what a producer does. By the time I got to the Brothers Johnson and Michael Jackson it was second nature. It evolves.

To be a great producer takes love. You have to love the person you are working with and respect them. You observe what their limitations are, what you'd like to see them learn, and train them to react to certain things. Spielberg works like this too. When we first met it was like we fell in love. He first came

by my studio when I was doing *Thriller* and invited me over to his studio where he was doing *E.T.* I'd give him a synthesizer, he'd give me a light meter, and we found out we worked the same way. We would construct a foundation where we could be only so bad, and once we got that foundation then we'd take chances and improvise and jam a little bit and see where we could take it, see the range.

Computers and Education

I am on the advisory board at Viewpoints Research. I am very honored to be in that crowd of brilliant people. You *can* teach young people. People put limits on what young people can handle. Alan was showing me how he can teach them higher-plane math at three years old through the toys they play with, and teach them how to drive at four years old through computers. They are not even aware of it. It's astounding.

There was a time when doctors warned us against subjecting children to “good boy,” “bad boy”—the “stick and carrot” stuff—and let them form their own thoughts first. I am a big advocate for that and think that most parents, subconsciously or not, try to form the basis of a child to be like them. And that's not a good idea, they need to let children have their own personality.

With the computer kids have found their own world. My kids range from seventeen to fifty-six. I have six girls and one son, who is forty, but the girls are between seventeen and fifty-six. When the seventeen-year-old was three she was running the computer before she could read. My son is a genius with the computer. My nephew is a tester for Microsoft; he's a hacker. The influence of computers on kids has gone both ways—it can be positive or negative.

Faith Popcorn talks about something she describes as cocoonism, isolationism, or lack of social touch, which I think is a negative thing. But then, kids are also free to search out the things they are really interested in, and I think that's the positive side of it. They are free to go and challenge everything

that is “by the book.” Like my friend Shawn Fanning when he did Napster: he was eighteen years old and just playing around. It wasn’t just the peer-to-peer thing that was effective, it was the choice and the broad range of music that you could be exposed to—rather than just what was automatic, what Clear Channel might play. You could go in as deep as you want and find out about all kinds of music, and I think that’s healthy. I really do.

Marshall McLuhan told us fifty years ago that media would take over 50% of our parenting, and he was right.

Alan is a very special friend with a special mind and a special vision. We’re starting a consortium now to try to create a music curriculum for all of our schools. I am very concerned that American kids don’t know a damned thing about their music. It is so sad. I’ve said I would give \$2,000 to any rapper who knows Coltrane, Charlie Parker or Duke Ellington. They don’t know. It hurts. It’s very important they know the history.

We have a record business that is disappearing. It’s 95–99% piracy everywhere in the world. We have to fix that for the young kids. I experienced the biggest-selling record in the history of music, with Michael. We did three of them. It just hurts me to see the young kids—singers and songwriters—not get the opportunity. They have families to feed, rent to pay, a life to live. It’s not right to take their music.

Binary numbers go back thousands of years. We’re still using permutations of zeros and ones. That’s the power of our IT.

Quincy Jones is a conductor, record producer, musical arranger, film composer, television producer, and trumpeter. During five decades in the entertainment industry, he has earned a record seventy-nine Grammy Award nominations and twenty-seven Grammys. He is best known as the producer of the album Thriller, by pop icon Michael Jackson, which has sold over 110 million copies worldwide.

In 1968 Quincy was one of the first African-Americans to be nominated for an Academy Award in the “Best Original Song” category and the first to be nominated twice within the same year, the second nomination being for “Best Original Score” for his work on the music of the 1967 film In Cold Blood.

The number one item on Quincy’s list for the future is to save the record business. The next is to continue the work of the Quincy Jones Music Consortium, to make sure that kids understand what their own music is about. When not traveling the world he lives “down the road” from Alan in Los Angeles, California.

Gordon Bell

Dear Alan, Re: What about your digital afterlife?

Writing this, using my recollections and e-memory coming from the late 1980s, I'm convinced that having a surrogate e-memory that contains everything we've ever seen (i.e., mostly what was read and written) and heard is so useful. Having a real record of our interactions over these last forty years would be great. Each of us can claim we can recall many of these interactions. Thus, my comments here are a confirmation to at least me about the need to store everything and never delete it to aid our own recollections; and second, aid future archivists and others interested in long term preservation of the digital bits that constitute our varied lives. Thus, I present an appeal to you—and many of our readers are in a similar stage of their lives—to be able to collect and structure such personal-professional information for long term preservation to benefit others.

We first met at an ARPA Principle Investigators' Meeting at Alta, Utah in 1967. You always refer to this as the beginning of the “golden era” of computer science research, while lamenting its passing. You were working toward your 1969 Ph.D. at Utah. I had just come to CMU as an associate professor. Doug Engelbart from SRI presented the pioneering work that included new user interfaces. Having just introduced the PDP-6 timesharing system as a forerunner to the PDP-10 I was quite content with the use of direct commands, and it wasn't until I used the Apple Mac in 1984 that I gained an appreciation for the WIMP (windows, icons, menu, pointing device) interface. After all, what

could be better than the TECO or Emacs editors that we used, whereby any key could be defined to mean exactly what a user wanted, and the right key could indeed work miracles on a text string or whatever, including deleting it? Marvin Minsky was very direct about the lack of need for a new interface by a bunch of laid back folks on the west coast that sat around on bean bags talking about human augmentation and collaboration. So, it was a very interactive meeting and fun to hear your ideas and how the meeting influenced you. Your 1968 Master's Thesis on FLEX clearly shaped your thinking for the forty years that followed. Your FLEX Thesis Abstract stated a number of invariant principles that you have lived with:

The FLEX system consists of merged “hardware” and “software” that is optimized towards handling algorithmic operations in an interactive, man-machine dialog.

The basic form is that of a hardware implementation of a parametric compiler embedded in an environment that is well-suited for semantically describing and pragmatically executing a large class of languages. The semantic language is called FLEX, includes the compiler-compiler as a string operator and is used as the basic medium for carrying out processes. It is of a higher-level nature and may itself be used for describing many algorithmic processes.

The machine itself is designed to be of the desk-top variety and sell at a low price. Because of these design parameters, many compromises in time and space had to be made to save money. The software system is implemented in read-only memory. To allow any possibility at all of debugging such a scheme, the algorithms involved were distilled down to their essence so that the entire system for the machine can be displayed (in flow diagram form) on a small wall chart.

In many senses the described system is a “syntax-directed” computer.

During the spring of 1971, I ran an architecture seminar at CMU with Peter Freeman that created the top level specs of a large 16-processor, shared memory, multiprocessor that we felt would be useful for AI applications that we called C.ai [68]. The seminar stimulated the design and construction of a university size 16-processor computer, C.mmp at CMU that Bill Wulf led. Another member of the seminar, Tom McWilliams and Lowell Wood ultimately built the S-1 at Livermore along the lines we outlined for C.ai. Fortunately my e-memory was able to recall our connection on the design by noting from the report's acknowledgements: "Alan Kay of Stanford also interacted with the group frequently and in detail."

Our interaction about architecture naturally prompted us to invite you to be on a panel, "Past, Present, and Future Computer Structures," at the 1971 Fall Joint Computer Conference in Las Vegas that Allen Newell and I chaired with you, Fred Brooks and Dai Edwards of the University of Manchester. The notion of your Dynabook tablet was just radical—and is, even now, forty years later. Herb Grosch spoke critiquing the Manchester Machine, laid my multiprocessor architecture ideas to rest because they violated Grosch's Law, and finally complained that your Dynabook was just never, ever believable. Clearly for us, you had won Herb's endorsement and it was exactly where you like to be. Herb ended by saying that all these panels were bad, but this was the worst he had ever attended. Fred Brooks' retort: "Herb, you'd think you'd learn and just not come to them." Fifteen years later I was on a panel and Herb made the same claim about a panel.

Within a decade, John Ellenby had started GRiD to produce a portable personal computer that was almost recognizable as Dynabook, a tablet forerunner. John wrote me:

What Alan had in mind was something you used so you could show people what was on the screen and let them hear what sounds you made as you played it as a musical instrument ... strumming on the screen was a big thing for Alan (an accom-

plished organist, by the way). In that immediate sense it was a communication tool but not at all to the same extent as the GRiD System. What we envisaged at GRiD and what we built and fielded was ... as Steve Mann reports ... an RFC that could fit in my briefcase ... but something whose main job and whose design centre therefore was to be an electronic communicating device that could be stand-alone for long periods but whose primary role in life was to get you connected when you needed to be. ... The irony here is that Apple are the most likely people to produce a real Dynabook that can be an engaging musical instrument ... including strumming on the screen and that is the missing piece in Alan's Dynabook concept ... namely electronic communications and access to the world's knowledge riffs and all its music ... will be accomplished through iTunes and Google. In this sense both Alan and Steve Jobs back then both missed the main thing: it's not distributed/free-standing *or* shared/tethered that must be the design centre, it's the right set of compromises and the infrastructure that enables both distributed and shared at the right time and the right place. Connected when you need to be and free standing when you don't. The pendulum swings back and forth ... but we at GRiD got it swinging.

Although the Dynabook as such may not exist, circa 2005 tablet computers do much of what you envisioned except the singular programming environment. Our interaction led to an invitation to visit CMU and an offer to join the faculty. Fortunately, for history, you ended up at PARC. In 1972 I went back to DEC (Digital Equipment Corporation) to head R&D.

As a key inspirator at PARC, you might have been called the CIO or chief inspiration officer if "CxO" titles were handed out. The PARC Story and its contributions are pretty well documented and understood, claiming among other things object-oriented programming, personal computing and

Ethernet. The contributions and design of personal computing and Smalltalk are beautifully laid out in your article as exemplary scholarship and attribution.

In 1986, I had the pleasure of keynoting Adele Goldberg's ACM conference on the History of Personal Workstations. I tried to lay out this history outside of PARC and you gave an after dinner talk summarizing PARC's work and influence, e.g., on Apple. Your talk started with your video of a two-year-old girl working on a Mac. The video enabled me to understand the power and attraction of the pointing interface metaphor of "being able to be a two-year-old" that I still use. I had long abandoned command lines whose command names and modes I forget. So I, too, treat my computer as a two-year-old: I point, I click, and then wait impatiently, like a two-year-old, for the computer to do what I hope for. If it doesn't then I undo if the scene has changed and then repeat the point and click process until I get what I wanted or get tired of pointing and uttering two-year-old and other, sometimes obscene, phrases. If all else fails, I go into RTFM-mode hoping a manual will help.

In 1994 you, Chunka Mui and Nicholas Negroponte recruited me to the TTI/Vanguard Board where we meet five times a year to bring technologists together for two days to listen to talks about science and nearer-term technology that will result in future computational tools. For the last few years we also come to the meetings to hear about your progress at Viewpoints Research to rewrite operating systems that are a factor of a thousand smaller!

The Organ Recital at your home was especially memorable for me and the other Vanguardians. It was seeing your impressive library to feed your love of books that I now can wonder about your own immortality. Preserving digital lives of you and other people, both exceptional and ordinary, is one of my concerns now. You have 11,000 books, and no doubt many recordings of your own as a Jazz Musician and Organist. I wonder about the photos, videos, and hard drive space for all of this valuable content that needs to be preserved. Have you considered what your digital life will look and feel like? And, especially for others, your digital afterlife?

I have been on an experimental course for ten years with the MyLifeBits project to encode as much of life as possible onto a Terabyte-or-so e-memory with the goal of eliminating everything physical that can be reasonably (by my filter) encoded digitally including memorabilia [69]. I have about a million items, several hundred thousand pages of communication, web pages, presentations, and articles about my own professional life plus a hundred thousand photos including many of physical artifacts or e-stuff (e.g., certificates, plaques, and ephemera) that I have eliminated from life, a few videos that take up most of the space, ten thousand songs, but just a few hundred e-books. However, what I have is just bits and I have been unable to project what my digital afterlife should look like.

When Stanford University obtained the Buckminster Fuller archive, they heralded it as: “one of the most extensive known personal archives in existence.” Taking up 2,000 linear feet of shelf space, including hundreds of thousands of pages and over 4,000 hours of audio/video, Fuller’s archive is indeed impressive because “Bucky” was meticulous in caring for his legacy. But such an archive will be commonplace in a decade—even minimal—for a twenty-first century digital person [73]. While most of us will not be bothered by fame and hence having archivists and others fighting for our bits, it is likely that our progeny will want at least some of our bits. For example, having the health data of one’s ancestors, including their DNA, can go a ways in improving our own health and that of our progeny.

We may already see a glimpse of the future of biography and digital immortality. The National Library of Medicine’s Profiles in Science project encoded thirty twentieth-century scientists for twenty-first century scholarly study, containing over 180,000 files (mostly scanned from paper) [74]. Striking towards digital immortality, Carnegie Mellon created a virtual Albert Einstein that one can question and receive answers from, and similar effort has been made for Darwin [70]. Ed Feigenbaum has been leading an effort at Stanford to provide tools for Stanford faculty and scholars to archive their own e-memories using

SALT (Self Archiving Legacy Tool) lest they be forgotten and fail to become a part of history [76].

Faced with the demand for digital preservation, the British Library held its first Digital Lives conference in February 2009 to explore “how modern digital collections are being created, managed, and made accessible now and in the future.” [71, 72] What, they asked Emory University’s Woodruff Library, would they do with Salman Rushdie’s archives, which included five MacBooks and a sixty gigabyte disk [75]? The library grappled with the obvious need to back up all the contents. However, in order to protect the value of the intellectual property of Emory’s unique digital asset it must be scarce, i.e., no copies and limited access. Thus, the content can only be viewed at a single protected physical site, just like the special archive rooms in every large library.

A research community has formed around such digital life-logging, and work in the field has demonstrated the practicality and benefits of a wide array of e-memory creation. The next decade will see the commonplace recording and retention of every web page one ever reads, one’s exact location, one’s health data from a variety of wearable and even in-body devices, all one’s electronic communications, and all sorts of data from a world in which virtually everything that can be instrumented will be.

I believe that, as technologists, we must accept the challenge of retaining all digital data for a minimum of one hundred years: long enough to discover the biographical highlights of both one’s ancestors and historic figures, to examine scientific data, and to settle on the important parts that should be preserved even longer. The first step is ensuring that documents, photos, audio, and videos are stored in gold-standard formats that are unlikely to become obsolete—that is, that are used by millions of people, ensuring a future market for software that will either view the content or at least upgrade it to a more modern format. This is the digital equivalent of keeping paper, printed photos, audio tapes and video tapes. But computers hold much more, particularly interactive formats such as spreadsheets that calculate what-if scenarios, databases with billions of records,

and three-dimensional visualizations. The hardware/software combination required to render these interactive formats cannot last forever unless we make the appropriate changes for universal emulation. Preserving this data presents a demanding challenge to the computer science community: the use of virtual machines that can run all the software from past generations. Even given the right formats and virtual machines, the big question for an archivist or library regarding acquisition and long term storage has simply to do with capacity. Who will make it into the digital lifeboat and have their story preserved? Will it be just the famous? Should one life be preserved at full resolution, or two lives at half-resolution?

Having the information is critical, but the most important challenge is making the tools so that others can view and gain insight from all these digital persons that approach immortality.

Gordon Bell has been a colleague of Alan for more than four decades. He led the development of the DEC (Digital Equipment Corporation) VAX architecture, served as Professor of Computer Science and Engineering at Carnegie Mellon University, and was the first Assistant Director of the National Science Foundation's Computing Directorate.

Educated at MIT and a Fulbright Scholar to the University of New South Wales, Gordon is a member of the National Academy of Engineering and the National Academy of Sciences, and a Fellow of several institutions including the ACM, the IEEE, the American Academy of Arts & Sciences, and the Australian Academy of Technological Sciences and Engineering. Gordon was awarded the National Medal of Technology in 1991. He has served alongside Alan on the board of the TTI/Vanguard Conferences since 1994.

Gordon is currently a Principal Researcher at Microsoft's Silicon Valley Research Center.

Danny Hillis

The Power of Conviction

Many years ago, when I was an undergraduate at MIT, I attended a talk by a visiting distinguished lecturer named Alan Kay. He spoke about his vision of a personal portable computer, the Dynabook, and he seemed to be actually trying to build it. Unlike other distinguished lecturers I had seen before, he wore sneakers and a t-shirt and he was not a university professor. I immediately knew that I wanted to do what he did, but it took me decades to understand what that was.

I now understand that Alan Kay is an architect, by which I mean someone who deliberately designs the world in which we will live. In Alan's own words, he is someone who "invents the future." To do this successfully, an architect must perceive the possibilities of reality and guide them into the service of practicable human goals. This is not easy. I have learned from Alan that it requires tapping into three sources of power, which I will call Knowledge, Imagination, and Conviction. The successful architect must possess the knowledge to know what is possible, the imagination to see what is desirable, and the conviction to build a connection between them. Any two of these traits can be useless, or even dangerous, without the third. The greatest architects possess all three.

I will not dwell on knowledge, except to point out that a successful architect requires a lot of it. Great architects have a voracious appetite for all kinds of knowledge and their libraries are filled with thousands of volumes ranging

from Astronomy to Zoology, Anthropology to Zoroastrianism. They are interested in the details, especially about the more practical subjects like agriculture and engineering. They are also interested in understanding things from first principles. Since mathematics is the language of science, a serious architect will be mathematically fluent. (Studying science without mathematics would be like studying French poetry without learning French.) Yet, the architect's appetite for knowledge is not limited to science. It includes art, music, literature, and most especially, history. Great architects continue to learn for their entire lives, which may account for their serious interest in education.

True knowledge guides conviction and inspires imagination. Ignorant people often claim that knowledge is a constraint on imagination, but the truth is just the opposite. Just as the rules of the chess pieces create the possibilities of the game, so too the rules of reality create the possibilities for imagination. The world is full of surprising examples of how these rules play out, providing an inexhaustible source of inspiration. A great architect is steeped in metaphors, matching every problem against a great reservoir of known patterns. (The computer is a musical instrument. The screen is a desktop.) These are the ingredients of imagination.

The final point that I would like to make about imagination is that, to an architect, imagination is mostly about the future. To invent the future, one must live in it, which means living (at least partly) in a world that does not yet exist. Just as a driver whizzing along a highway pays more attention to the front window than the rear, the architect steers by looking ahead. This can sometimes make them seem aloof or absent-minded, as if they are someplace else. In fact, they are. For them, the past is a lesson, the present is fleeting; but the future is real. It is infinite and malleable, brimming with possibility.

This brings us to Conviction. Conviction without knowledge or imagination is just pig-headedness, but informed, imaginative conviction is the force that creates our future. Faced with the infinite possibilities, an architect must make judgments about where to focus their energy and attention. They must decide what is important and commit to a point of view. Conviction gives

purpose to knowledge and clarity to imagination. I think this is what Alan is talking about when he says that, “A point of view is worth 80 IQ points.” A point of view guides you to what is salient. Without it, knowledge and imagination would be directionless.

Few philosophers have the practical skills to be architects, but every great architect must be a philosopher. Like Thomas Jefferson, they must hold certain truths to be self-evident, and they must have the determination to act upon those truths. Conviction is required because the world will resist change. The world has a lot of inertia and it is only moved by people who are willing to keep pushing in the same direction for a long period of time.

When I heard Alan’s talk at MIT a quarter century ago, he said a few things that I still remember. He said that the good is the enemy of the great. He said that systems can only scale if they are constructed according to carefully designed principles of modularity. He advised picking a metaphor that you like and running with it. He said to be sure that you are having fun. He was absolutely certain that computers would one day be so intuitive and natural, so personal, that they would feel like an extension of our minds. Even then, I understood that Alan had the knowledge and imagination to build his Dynabook. That part was obvious, but it took me many years to appreciate the depth and wisdom of his conviction. I now appreciate its importance. While I have met many people with knowledge and imagination, I have met very few who, like Alan, have the deep conviction required to change the world.

Danny Hillis received a Ph.D. in Electrical Engineering and Computer Science from MIT in 1988 for his work on the 64,000-processor parallel computer called the Connection Machine, and was later appointed adjunct professor at the MIT Media Lab.

He was co-founder of Thinking Machines Corporation before starting his own consulting firm in 1995. In 1996 Danny and Alan both became Fellows at Walt Disney Imagineering.

He has received numerous awards, is a member of the National Academy of Engineering, and is a fellow of several institutions including the Association for Computing Machinery and the American Academy of Arts & Sciences.

Danny is currently co-chairman and Chief Technology Officer at Applied Minds, Inc., the company he co-founded in 2000. He also serves as co-chair of The Long Now Foundation and is the designer of its 10,000 Year Clock.

“Cheshire Puss,” Alice began, rather timidly, “Would you tell me, please, which way I ought to go from here?”

“That depends a good deal on where you want to get to,” said the Cat.

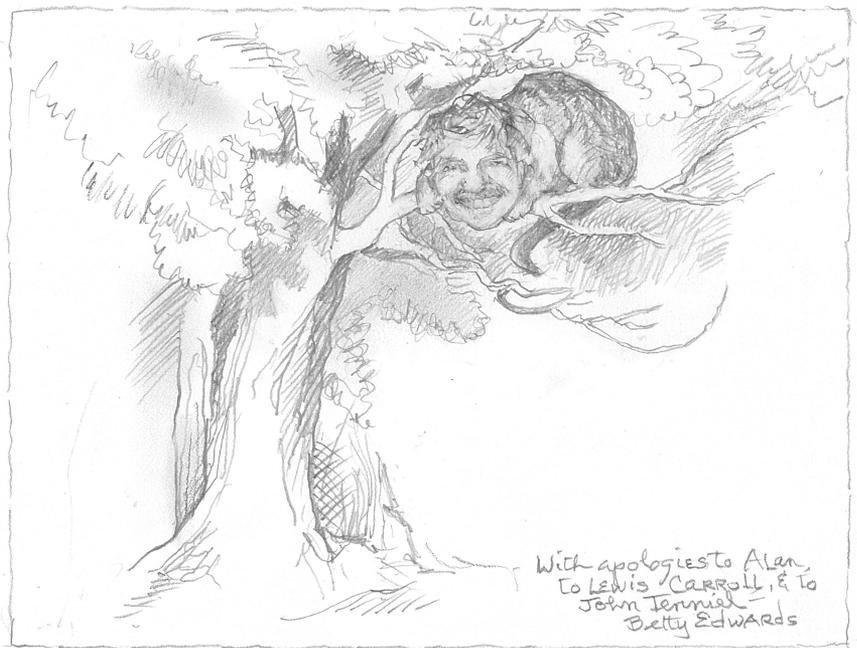
“I don’t much care where—” said Alice.

“Then it doesn’t matter which way you go,” said the Cat.

“—so long as I get *somewhere*,” Alice added as an explanation.

“Oh, you’re sure to do that,” said the Cat, “if you only walk long enough.”

Lewis Carroll



With apologies to Alan,
to Lewis Carroll, & to
John Tenniel -
Betty Edwards

Afterword

Many people donated time and energy to make this book possible. Of those who are not listed in the Contents, two deserve particular mention.

Kathryn Hewitt, administrative assistant at Viewpoints Research, not only managed to keep this book a secret from Alan and our other colleagues for ten months, but also proofread most of the chapters in the book. She found many errors that had escaped our notice during extensive revision and re-reading.

David Mayes, our “insider” at Typecraft Wood & Jones, generously donated his time, extensive knowledge of print production, materials for prototyping, and culinary recommendations in Pasadena, throughout the process of bringing this book to press. His willingness to discuss at length and without reservation the myriad small details surrounding the production of this book, a tiny project by his standards, contributed greatly to the quality of the final artifact. It is always a pleasure to visit him, talk about about printing and typesetting with him, and marvel at the many examples of exquisitely beautiful books that he has helped to produce.

This book was designed by the editors and typeset using Donald Knuth’s T_EX document preparation system. The body is set in 11 pt Adobe Garamond Premier Pro on a 16 pt baseline.¹ The sans-serif and monospaced types are from the Bitstream Vera family. It is printed on 70# Classic Crest Text, Classic Natural White, and bound in Arlington #64400 Navy cloth. The paper is acid-free and of archival quality.

¹The typeface was modified slightly for use in this book.

Choosing the typeface was not difficult. Among the dozen or so types that we considered, Garamond was the clear winner for its simple elegance and readability. Its creator was a remarkable man—in some ways an Alan Kay of his time—and so I think it appropriate to describe a little of his story, achievements and the marks he left (literally) on the world.

Claude Garamond was born in 1490, just forty years after the European re-invention of movable type by Johannes Gutenberg. Garamond was an avid reader from an early age, fascinated by the process of bookmaking and in particular by the structure of the letters themselves. Publishing was the obvious career for him and he set up a print shop in Paris where he began to design and cut his own typefaces.

Garamond almost certainly based his romans on the types cut in 1495 by Francesco Griffo for the Venetian printer Aldus Manutius. Several adaptations of Garamond exist, all exhibiting a remarkable balance and grace. The letterforms are fluid and consistent, with several distinguishing features, in particular the wide, concave serifs and the very small counters (bowl of the “a” and eye of the “e”).

Complementing the elegance and readability of his types, Garamond the publisher insisted on clean design, generous page margins, careful composition, and the highest quality of paper, printing and binding.

Garamond designed his types specifically to be cut in metal, unlike his contemporaries who designed types to mimic handwriting, and was one of the principal forces behind the adoption of roman letters as standard.² He was the first person to design, cut and cast type specifically for sale to other printers, establishing the industry of typefounding. His italic types were designed specifically to complement his roman types, rather than as independent cursive

debant, (iis enim in huiusmodi casu nihil contra vim potentiorum tuti est) tum verò non inimici modò aut diuites, sed etiam amicissimi præter opinionem peribant. Quippe & si perpauci priuatis de causis triumuiris inuisi erant, vt iis ea res necem afferret, tamen publicæ res, & dominationum permutationes in causâ vehementibus istis cum amicitiiis tum odiis erant. quicumque enim vnum ex his vel studio vel factò suo adiuuisset, statim inimici loco reliquis habebatur. ita, idem homo alicui ipforum a-

Amici etiam triumvirorum quare profecti sunt.

²The time was thirty years after Aldus Manutius, the father of what is now the paperback, had invented italic type style as a means to save space in his small-format “portable” books.

scripts, giving birth to the notion of type families. We have him to thank for the introduction of the apostrophe, and for the accent and cedilla in the French language.

Garamond's punches were used for many decades after his death, ultimately finding their way to the Plantin-Moretus museum in Antwerp and to the *Imprimerie Nationale* in Paris.

It is fitting that a book written for an outlier, and largely by outliers, should be set in a typeface designed by one of the most influential outliers in the world of typography and printing, to whom many of today's typographic conventions and contemporary letterforms can be traced. In Garamond's day type sizes were named, not numbered, and it is also appropriate that he would have known 11 pt type as *Philosophie*.

A typesetter's task is to deliver the meaning of a writer's words to an attentive reader as clearly as possible, just as a musician's task is to deliver the meaning of a composer's music to an attentive listener as clearly as possible. As an amateur typesetter I can hope only that the words in these pages are delivered to you undamaged—without your having to pause to puzzle over the form in which they are presented.

Ian Piumarta
Los Angeles
March 2010

Bibliography

Preface

- [1] *Realtime: A Tribute to Hasso Plattner*, Wiley Publishing, Inc., Indianapolis, IN, USA, 2004. ISBN 0-7645-7108-7

Adele Goldberg

- [2] Graham Birtwistle, *Introduction to Demos*, Proceedings of the 13th Conference on Winter Simulation, pp. 559–572, 9–11 December 1981, Atlanta, Georgia.
- [3] Alan H. Borning, *ThingLab, A Constraint-Oriented Simulation Laboratory*, Technical Report SSL-79-3, Xerox Palo Alto Research Center, Palo Alto, CA, July 1979.
- [4] Laura Gould and William Finzer, *Programming by Rehearsal*, Technical Report SCL-84-1, Xerox Palo Alto Research Center, Palo Alto, CA, May 1984.
- [5] Ted Kaehler, *Virtual Memory for an Object-Oriented Language*, Byte Magazine, August 1981.
- [6] Ted Kaehler and Glenn Krasner, *LOOM—Large Object-Oriented Memory for Smalltalk-80 Systems*, in Glenn Krasner (ed), *Smalltalk-80, Bits of History, Words of Advice*, Addison-Wesley, 1983. ISBN 0-201-11669-3
- [7] Alan Kay, *The Early History of Smalltalk*, ACM SIGPLAN Notices, Vol. 28, No. 3, pp. 69–95, March 1993. Also in Thomas Bergin and Richard Gibson (eds), *History of Programming Languages*, Vol. 2, 1996.

- [8] Alan Kay and Adele Goldberg, *Personal dynamic media*, IEEE Computer, Vol. 10, pp. 31–41, March 1977. Reprinted in A. Goldberg (ed), *A History of Personal Workstations*, Academic Press, New York, 1988.
- [9] Kenneth Rubin and Adele Goldberg, *Object Behavior Analysis*, Communications of the ACM, Vol. 35, No. 9, pp. 48–62, January 1992.
- [10] David Smith, *Pygmalion: A Creative Programming Environment*, Stanford University Computer Science Technical Report No. STAN-CS-75-499, June 1975.
- [11] David Smith, Alan Cypher and Jim Spohrer, *KidSim: Programming Agents Without a Programming Language*, Communications of the ACM, Vol. 37, No. 7, July 1994.
- [12] Randy Smith, *Experiences with the alternate reality kit: an example of the tension between literalism and magic*, ACM SIGCHI Bulletin, Vol. 17, 1986. Reprinted in *Computer Graphics and Applications*, Vol. 7, No. 9, September 1987.
- [13] Robert Stults, *Media Space, After 20 Years*, in S. Harrison (ed), *Media Space: 20+ Years of Mediated Life*, Chapter 14, p. 293, Springer-Verlag, London, 2009.
- [14] Robert Stults, Steven Harrison and Scott Minneman, *The Media Space—experience with video support of design activity*, in A. E. Samuel (ed), *Engineering Design and Manufacturing Management*, pp. 164–176, Elsevier, Amsterdam, 1989.
- [15] Stephen Weyer, *Searching for information in a dynamic book*, Ph.D. dissertation, School of Education, Stanford University, 1982.

Larry Smarr

- [16] Annie Chabert, Ed Grossman, Larry Jackson and Stephen Pietrovicz, *NCSA Habanero—Synchronous collaborative framework and environment*, Software Development Division at the National Center for Supercomputing Applications.

<http://www.isrl.illinois.edu/isaac/Habanero/Whitepapers/ecscw-habanero.html>

- [17] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, *Surround-screen projection-based virtual reality: the design and implementation of the CAVE*, Proceedings of the 20th annual conference on Computer Graphics and Interactive Techniques, 1993, pp. 135–142.
- [18] M. Czemuszenko, D. Pape, D. Sandin, T. A. DeFanti, G. Dawe, M. D. Brown, *The ImmersaDesk and Infinity Wall projection-based virtual reality displays*, SIGGRAPH Computer Graphics, Vol. 31, No. 2, pp. 46–49, 1997.
- [19] T. DeFanti, G. Dawe, D. Sandin et al, *StarCAVE, a Third-Generation CAVE and Virtual Reality OptIPortal*, Future Generation Computer Systems, Vol. 25, Issue 2, pp. 169–178, 2009.
- [20] T. DeFanti, J. Leigh, L. Renambot et al, *The OptIPortal: a Scalable Visualization, Storage, and Computing Interface Device for the OptIPuter*, Future Generation Computer Systems, Vol. 25, Issue 2, pp. 114–123, 2009.
- [21] Future Generation Computer Systems, *Special issue*, Vol. 22, Issue 8, October 2006.
- [22] Future Generation Computer Systems, *Special issue*, Vol. 25, Issue 2, February 2009.
- [23] B. Jeong, L. Renambot, R. Jagodic et al, *High-performance dynamic graphics streaming for scalable adaptive graphics environment*, Proceedings of the ACM/IEEE Conference on Supercomputing, Tampa, Florida, November 11–17, 2006.
- [24] Volodymyr Kindratenko and Berthold Kirsch, *Sharing virtual environments over a transatlantic ATM network in support of distant collaboration in vehicle design*, Proceedings of the Eurographics Workshop on Virtual Environments, 1998, Springer Computer Science Series, Springer-Verlag, Berlin, Germany, pp. 151–161.
- [25] C. Lascara, G. Wheless, D. Cox et al, *TeleImmersive Virtual Environments for Collaborative Knowledge Discovery*, Proceedings of the Advanced Simulation Technologies Conference, San Diego, CA, April 1999, pp. 11–15.

- [26] J. Leigh, A. E. Johnson, T. A. DeFanti, *Issues in the Design of a Flexible Distributed Architecture for Supporting Persistence and Interoperability in Collaborative Virtual Environments*, Proceedings of the ACM/IEEE Conference on Supercomputing, 1997, p. 21.
- [27] D. Pape, *A Hardware-Independent Virtual Reality Development System* IEEE Computer Graphics and Applications, Vol. 16, Issue 4, pp. 44–47, 1996.
- [28] B. St. Arnaud, L. Smarr, J. Sheehan and T. DeFanti, *Campuses as Living Laboratories of the Greener Future* EDUCAUSE Review, Vol. 44, No. 6, pp. 14–32, 2009.
- [29] L. Smarr, *Riding the Light Towards New Science*, Nature Photonics, Vol. 1, pp. 133–135, 2007.
- [30] L. Smarr, *The OptIPuter and Its Applications*, IEEE LEOS Summer Topical Meeting on Future Global Networks, July 22, 2009, pp. 151–152.
- [31] L. Smarr and C. Catlett, *Metacomputing*, Communications of the ACM, Vol. 35, No. 6, pp. 44–52, 1992.
- [32] L. Smarr, P. Gilna, P. Papadopoulos et al, *Building an OptIPlanet Collaboratory to Support Microbial Metagenomics*, Future Generation Computer Systems, Vol. 25, Issue 2, pp. 124–131, 2009.
- [33] L. Smarr, L. Herr, T. DeFanti et al, *CineGrid: A New Cyberinfrastructure for High Resolution Media Streaming*, Cyberinfrastructure Technology Watch Quarterly, Vol. 3, No. 2, 2007.

David Reed

- [34] Douglas Crockford, *Javascript: The Good Parts*, O'Reilly, 2008.
ISBN 978-0596517748
- [35] Richard Dawkins, *The Selfish Gene*, Oxford University Press, 1976.
ISBN 978-0192860927
- [36] Richard C. Lewontin, *The Triple Helix: Gene, Organism, Environment*, Harvard University Press, 2000. ISBN 978-0674001596

- [37] John McCarthy et al., *The LISP 1.5 Programmer's Manual*, MIT Press, 1965.
ISBN 978-0262130110

Doug Lenat

- [38] James Cimino et al., *HL7 Good Vocabulary Practices*, Radiological Society of North America, RadLex Committee, November 20, 2001.
- [39] Edward Feigenbaum and Bruce Buchanan, *Dendral and Meta-Dendral: roots of knowledge systems and expert system applications*, Artificial Intelligence, Vol. 59, pp. 233–240.
- [40] Adele Goldberg and Alan Kay, *Smalltalk-72 instruction manual*, Technical Report SSL 76–6, Learning Research Group, Xerox Palo Alto Research Center, March 1976.
- [41] Paul Grice, *Studies in the Way of Words*, Harvard University Press, Cambridge, MA, 1991. ISBN 978-0674852716
- [42] Frederick Hayes-Roth, Douglas B. Lenat and Donald Arthur Waterman, *Building Expert Systems*, Addison-Wesley, 1983. ISBN 0201106868
- [43] Carl Hewitt, Peter Bishop and Richard Steiger, *A Universal Modular Actor Formalism for Artificial Intelligence*, Proceedings of the third International Joint Conference on Artificial Intelligence, Stanford, California, 1973.
- [44] Alan Kay, *speaking* at the Creative Think seminar, Apple, July 20, 1982.
http://folklore.org/StoryView.py?project=Macintosh&story=Creative_Think.txt
- [45] Alan Kay, *Predicting the Future*, Stanford Engineering, Vol. 1, No. 1, Autumn 1989, pp. 1–6.
- [46] Douglas B. Lenat, *BEINGS: knowledge as interacting experts*, Proceedings of the fourth International Joint Conference on Artificial Intelligence, Vol. 1, 1975, pp. 126–133.

- [47] Douglas B. Lenat, *AM: An artificial intelligence approach to bdiscovery in mathematics as heuristic search*, Stanford AI Lab technical Report AIM-286, Computer Science Technical Report STAN-CS-76-570, and Heuristic Programming Project Report HPP-76-8, 1976. Published in *Knowledge-based systems in artificial intelligence*, McGraw-Hill, 1982. ISBN 0070155577
- [48] Douglas B. Lenat and R. V. Guha, *Enabling Agents to Work Together*, Communications of the ACM, Vol. 37, No. 7, July 1994, pp. 126–142.
- [49] James Grier Miller, *Living Systems*, University Press of Colorado, 1995. ISBN 978-0870813634
- [50] Allen Newell, *Human problem solving*, Prentice-Hall, 1972. ISBN 978-0134454030
- [51] Seymour Papert, *Teaching Children to Be Mathematicians vs. Teaching About Mathematics*, MIT AI Memo 249, LOGO Memo 4, July, 1971. Published in the International Journal of Mathematical Education in Science and Technology, Vol. 3, No. 3, 1972, pp. 249–262.
- [52] Daniel Weinreb and David Moon, *Flavors: Message Passing in the Lisp Machine*, MIT AI Memo 602, November, 1980.

Butler Lampson

- [53] A. Borning, *The programming language aspects of ThingLab, a constraint-oriented simulation laboratory*, ACM Transactions on Programming Languages and Systems, Vol. 3, No. 4, October 1981, pp. 353–387.
- [54] S. Brin and L. Page, *The Anatomy of a Large-Scale Hypertextual Web Search Engine*, Computer Networks and ISDN Systems, Vol. 30, No. 1–7, 1998, pp. 107–117.
- [55] F. Chang, *Generation of Policy-rich Websites from Declarative Models*, Ph.D. Thesis, MIT, February 2009.
http://sdg.csail.mit.edu/pubs/theses/felix_phd.pdf

- [56] L. de Moura and N. Bjørner, *Z3: An efficient SMT solver*, Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Budapest, Hungary, 2008.
- [57] C. Gomes et al., *Satisfiability solvers*, Handbook of Knowledge Representation, Elsevier, 2008, pp. 89–134.
- [58] J. Gray and A. Szalay, *Virtual observatory: The World Wide Telescope*, Science, Vol. 293, September 2001, pp. 2037–2040.
- [59] G. Little and R. Miller, *Keyword Programming in Java*, Automated Software Engineering, ACM/IEEE, 2007, pp. 84–93.
- [60] D. Lowe, *Object recognition from local scale-invariant features*, Proceedings of the International Conference on Computer Vision, 1999, pp. 1150–1157.
- [61] N. Snavely et al., *Modeling the world from Internet photo collections*, International Journal of Computer Vision, Vol. 80, No. 2, November 2008, pp. 189–210.
- [62] C. Upson et al., *The Application Visualization System: A computational environment for scientific visualization*, IEEE Computer Graphics and Applications, Vol. 9, No. 4, July 1989, pp. 30–42.
<http://www.avs.com/>
- [63] B. Williams et al., *Model-based programming of intelligent embedded systems and robotic space explorers*, IEEE Proceedings: Special Issue on Modeling and Design of Embedded Software, Vol. 9, No. 1, January 2003, pp. 212–237.
- [64] Y. Yu et al., *DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language*, Proceedings of the International Conference on Operating System Design and Implementation, USENIX, December 2008, pp. 1–14.

Vishal Sikka

- [65] M. G. Anlauff, V. Sikka and R. Yaseen, *Language-level integration of programming models*, United States Patent Application No. 20090241091.

- [66] A. Kay, *The Early History of Smalltalk*, ACM SIGPLAN Notices, Volume 28, No. 3, pp. 69–95, March 1993. Also in Thomas Bergin and Richard Gibson (eds), *History of Programming Languages*, Volume 2, 1996.
- [67] V. Sikka, *Integrating specialized procedures into proof systems*, Ph.D. Thesis, Stanford University, 1996.

Gordon Bell

- [68] Gordon Bell and Peter Freeman, *C.ai—A Computer for AI Research*, AFIPS Fall Joint Computer Conference, 1972, pp. 779–790.
- [69] Gordon Bell and Jim Gemmell, *Total Recall*, Dutton, New York, 2009. ISBN 978-0525951346
- [70] Gordon Bell and Jim Gray, *Digital Immortality*, Communications of the ACM, January 2001.
- [71] British Libraries, *First Digital Lives Research Conference: Personal Digital Archives for the 21st Century*, February 2009.
<http://www.bl.uk/digital-lives/conference.html>
- [72] British Libraries, *Digital Lives: A Digital Lives Research Paper*, October 2009.
<http://britishlibrary.typepad.co.uk/files/digital-lives-legal-ethical.pdf>
- [73] Jim Gemmell and Gordon Bell, *The E-Memory Revolution*, Library Journal, 15 September 2009.
- [74] National Library of Medicine's *Profiles in Science* project.
<http://profiles.nlm.nih.gov>
- [75] Emory University Distinguished Writer in Residence archived at Emory's Woodruff Library, 2007.
<http://marbl.library.emory.edu/fc-salman-rushdie-papers.html>
- [76] Will Snow, *Self Archiving Legacy Tool*.
<http://www.diglib.org/forums/spring2008/presentations/Snow.pdf>

