CampSmalltalk Supreme keynote
Adele Goldberg
June 11, 2022

Smalltalk research through the eyes of an educational mission

Transcript of the keynote presentation by Adele Goldberg

The critical idea behind the invention of Smalltalk was that it would be the software solution to a personal, accessible device called the Dynabook. This handheld device would be a new medium in which people could share both their understanding of how something works, and, where appropriate, be challenged as to whether that understanding approximated reality. The Dynabook would offer children a thinking partner with the modeling capacity to test ideas and upend how we educate the next generations. The mission is not complete, but how far did we get? And is the mission still critical? In this presentation, I will attempt to answer these questions.

Introduction (by Richard Kenneth Eng)

Adele Goldberg. Ms Goldberg is one of the co-creators of Smalltalk, a past president of the Association for Computing Machinery, and with Alan Kay and Dan Ingalls, a recipient of the ACM software systems award in 1987. In 1994 she was inducted as a fellow of the Association for Computing Machinery. This year Adele Goldberg and Dan Ingalls were honored by the Computer History Museum for their contributions to Smalltalk programming and the use of computers in education. As it happens Ms Goldberg was the first person to introduce Smalltalk to Canadians back in the 1980s. At the risk of embarrassing Ms Goldberg, i'd like to show a video from that time.

An Educational Mission

Good morning. I've been out of the tech industry for a while now. I've been mostly enjoying my roots in the art world and mostly working on photography as an art and a craft. I thought about who would show up for 50 years later for Camp Smalltalk and decided you were all real

techies right? But that's not what i'm going to do. Instead I decided to take this opportunity to revisit the Smalltalk educational mission.

Specifically, if I were offered the money that we had back in the 70s, to create a research team such as the one that Alan Kay put together, and we could start over again, but with today's technology, what would we do? I was there for an educational mission and I don't feel that that mission has yet been met. So what I wanted to do was, I set a task for myself. First, to reconstruct the idea of the Dynabook. The Dynabook is more than the handheld computer that we have today and we're all familiar with. And second, to revisit some of the research projects that we did back in the 70s and 80s and 90s and see what we learned. See what the themes are and what the questions around those themes would be. So to a large extent, what I'm doing is extracting questions that would need to be answered if we actually were going to have a Dynabook, which you can tell from what I'm saying, I don't think we yet have.

It is my assumption that, with the exception perhaps of the programming interface to the Simulation Kit, you already know the details of these projects. But since I made an assumption, and I'm not sure, I'm going to give very brief introduction to those projects, that you can look up and some of them you can actually play with, because Dan has started a thing which I love, he calls the Smalltalk Zoo where you can actually play with some of those.

The Dynabook

So let's start with what is the Dynabook. Whenever you look over the years of what was driving the Xerox Smalltalk research, you can see the desire for technology to enable creativity, and you can see an educational mission. The critical idea was that the Dynabook be the medium in which people can share their understanding of how things work, and be challenged to whether this understanding reflects an approximation to reality. So I see the Dynabook, computationally as a modeling environment, and ultimately I see it as a thinking partner with whom the user can rely on, to find data or use cases that do or do not fit a proposed model. In other words it's not purely a general purpose programming system, nor purely a communications device. This is an early sketch from Alan Kay. Of course, it has to provide resources for communication, for collaboration, and visualization. It should draw on the human senses for touching, and seeing, and hearing, for input and output as forms of communications including or especially for self

reflection. Educationally it should support the learner's ability to create models, working individually or collaboratively, and those models should be about both the physical and social worlds, both real and imaginary, to test the learners thinking about those worlds. Now today and over the last years there's been some interesting modeling examples for educational use. Mostly they're presented as gaming applications. They're multimedia applications written in or extendable by an object-oriented programming language. An interesting example that I like is Blender 3d which has a Python API. There are Internet sites that give access to multi-person gaming, and there are multi-person construction sites such as Simcity, Minecraft, and Roblox. There are also purely educational sites, such as that supported by the Khan Academy where the tutoring was originally a collection of short videos, and is now organized by standard school curricula.

The question is, is it time to burn the disk again? Alan, remember what he said: children of all ages are growing up thinking that technology, good technology matters. The Dynabook educational mission as I said is far from completed. Given the extent to which, our returning to the educational mission is important. For today's talk I wanted to look at the Smalltalk based project history, to see what we learned, in order to accomplish our educational mission. Including as I said, to revisit whether the objects and messages paradigm is well suited to the task. In the past Alan Kay would call the activity that I set out to do as "burning the disk", starting over given what we have learned, with the eye on how both technology and society have changed.

Of course each time a new Smalltalk was released, we were in essence burning the disk. Since the mission did not change, neither the name: still Smalltalk. In the interest of checking out the latest disks before burning them, I explored various application level projects that were done in our group (which with time was called the Learning Research Group), in other Xerox groups, and elsewhere. Projects that are not always discussed from the education mission perspective when we look back at the Smalltalk history.

I wanted to say one thing about the Computer History Museum which is in Mountain View California, which is in September the 1st, there will be another anniversary celebration, a formal one from the museum, and I believe it's open to the public.

Please note that i'm not going to present these projects in chronological order, but rather as how I went searching for a small set of themes and related research questions, to use in evaluating future Dynabook solutions.

The need for Curriculum

I think the only surprise in what I'm saying might be in the need for curriculum. When the mission is to upend how kids are educated, so they can be intelligent participants in a growingly complex global community, then we need to have the content of such an education, not just the context and tools. There's no doubt that it is a major undertaking to create tested curriculum content for at least the first nine years of formal schooling. But there's plenty of evidence that the difference in acceptance or popularity of one programming language over another, is often the existence of curriculum. For professionals, students, and teachers alike, they rely on books, which now are mostly electronic. For self learning, we rely on Internet help sites and online video tutorials. And for kids in school, curriculum is a critical guide for their teachers.

As history reports, the Xerox Learning Research Group started its work by creating Smalltalk 72, both the programming language and its environment, and testing these by creating creativity tools. When I joined, which was in 1973, my first step was to worry about how to teach the language, and to advise the team of what may make the teaching task easier, without losing the programming power of the language. Others helped with hardware and communication parts, and later our group expanded and worked on collaboration tools. My background was in Artificial Intelligence and its use in education, to create intelligent computer-based tutors. But at Xerox, creating any such tutors was set aside.

Dynabook as Modeling Environment

So what does it mean to say the Dynabook is a modeling environment? A modeling environment is more than a programming system. It is a medium in which to explore how things work, so it needs to be a library of things that work. It is a medium that challenges the user to consider whether the way things work reflect a useful approximation to reality, which implies that it provides access to testing data and use cases. And it is a medium for creating models that can be accessed and viewed at different levels of sophistication, and that can be shared with human mentors or even a computer tutor.

Back in 1972. Are you all too young to be there? No! Okay, good. Back in 1972, there were a number of projects worldwide that were founded on the belief that computer programming afforded a new and useful way to think about teaching problem solving skills. Notably, the pedagogy emphasized an iterative approach to developing alternative solutions, rather than button holding students into thinking there are only two results: a right one and a wrong one, or that you either get it right or wrong. In reality what was being taught is that you write something, experiment with it, and search for and fix apparent or differences in what your goal was. These projects set out to demonstrate how to teach programming, often by having the learners do physical activities, that could be directly mapped to executable computer algorithms. Probably the most famous such projects were done with the language Logo at both BBN and MIT, better remembered as Turtle Geometry, and recreated in most other programming languages targeting young kids, including Smalltalk-72. Creating geometric designs as though a robot were moving forward or backward or turning left and right, has morphed into creating simple games and animations and line drawings.

Cultural surroundings or designed curricula?

It was Squeak, Etoys and Scratch in the Smalltalk family, and I do think that Scratch was likely the more successful, because of its focus on curriculum and partnering schools worldwide, building internet community and crowdsource support. I think if you want to understand the level of detail it would take to create a multi-year curriculum, based on doing physical activities, that inform kids about possible models that they can program, you can take a look at the Squeak book from 2003. It is shown here on the slide. It is entitled "Powerful ideas in the classroom", it was written by B.J. Allen-Conn and Kim Rose. In it, they unfold how their students are to compare whether objects of different weights take the same or different time to fall to the earth from say a roof. Thinking back about the curriculum problem, and its complementary problem of teacher training, bring back memories of Seymour Papert, jumping around on a table pretending to be a turtle robot, in order to teach geometry. As most classroom teachers bulked at jumping around on the top of the table the curriculum delivery approach definitely needed more universal appeal. And of course reaching back in history, when we think about the goal of appending how children learn, we should be reminded of the 1980 book "Mindstorms, children

computers and powerful ideas" by Seymour Papert, an effort to shine the light on computers to be used for computation. Programming a computer was in contrast to having the computer turn the pages of online programming instruction, or drill and practice, that was the rage in the late 60s and early 70s, from such practitioners as Patrick Suppes at Stanford university with whom I worked as a graduate student. Papert called his world's Microworlds but at the core, the ideas of identifying restricted libraries and visual user interfaces for introducing computer concepts, is similar to some of what we tried with Smalltalk. Papert would say that in Microworlds the student is programming the computer, whereas in computer-assisted instruction the student is being programmed by the computer. Papert would want us to have the students think about thinking, which is another way to talk about building models for understanding, and of course since he was a student of the psychologist Piaget, Papert's hypothesis was, that the barrier that takes a child from concrete thinking to formal thinking, is overcome by the ability to program. To making learning more active, more self-directed, and I would add more exploratory, as an approach to create concrete representations of formal explanations.

I do have to note at this point, that Papert and I diverged on the importance of curriculum. As a student of Piaget, Papert preferred the kind of learning that happens without deliberate teaching, although he did try to equate curriculum to the Piagean idea of creating specific cultural surroundings, that provide materials from which children learn. Thus turtle geometry defined as a microworld consisting of a particular set of objects and their behaviors provided to the learners, delivers a kind of curriculum. What we do agree on is that a well-selected set of interacting objects, what Papert called Microworlds and we called Kits, is likely a more engaging approach to the educational use of computer technology, than drill and practice.

Although we did work with kids, at our lab and in on their schools, and we did look to find ways to teach them how to program by creating interesting interacting objects, doing so was really informing us about the teachability of whatever was then the current version of Smalltalk. We were not researching whether students became better problem solvers or thinkers solely through their use of a computer programming language, and I would add that although many people have anecdotal evidence that there is some effect on kids thinking and approach to thinking, it is not scaled up, which means it's not a worldwide experiment.

Our initial pedagogical ideas for teaching Smalltalk originated with our teaching Smalltalk-72. Specifically we were asking whether the elements of an executable software

solution, explicit in the form of object definitions and message passing among objects, offers something special in the way of creating interactive models. The issue we ran into, in Smalltalk-72, was that the receiver of the message had control of how to parse the remaining message stream. This means that the only way learners could understand and talk to one another about a Smalltalk 72 method, was to mentally or verbally execute the method associated with the message. Besides being very hard for beginners to understand, such a situation ran counter to the pedagogical idea that you learn to write by reading, and introduce what turned out to be unnecessary learning barriers. It certainly restricted the complexity of what could be taught. So remember the context of learning environment based on modeling, is as much about understanding how an existing model works, as it is constructing a new model. As a consequence, the ability to read an implementation matters.

I'm going to say a little side note here. The thing that always, always annoyed me about the C++ world, was that they had obfuscation contests to see if you could write something you could not figure out what it did. Right? The read to write approach to the use of Smalltalk was critical to our educational efforts, and most importantly our ability to attract users who were not programmers by profession. Smalltalk as the modeling language evolved as we explored the pedagogical idea that the students start with models they can read and then refine those models or reuse them to create new ones.

NoteTaker

Let me start looking at projects from the past with the stories some of you may might have already heard. How many of you knew about the NoteTaker from 1978. One, okay. In 1978 we worked with a group of hardware designers at Xerox led by Doug Fairbairn, in order to build a portable, carry it with you computer. In those days they were called luggables. By 1978 we had published Smalltalk-72, and we had released a major new language named Smalltalk-76. In 1978 Doug's team at Xerox built a portable physical device based on Intel 8086 processors, that included a mouse a small display screen and an Ethernet board. The resulting computer was indeed luggable, but it fit under an airplane economy seat, not first class, I don't know why not, and ran Smalltalk-76.

But why call it a Note Taker? So i'm going to take responsibility for what might seem an unlikely name. I suggested the following scenario: the student is tasked with research on how socializing among the servants in the household of the English and French royalty might have influenced alliances and disputes. Since it was 1978, 16 years prior to the introduction of the Internet, the research challenge talked about going to a brick and mortar library, and browsing the books in their collections, but the idea of course applies to in today's Internet age. Generally the educational plan was first for the student to formulate a hypothesis, consisting of with whom, with what, and where each socializing among the servants might have taken place. Then gather data on significant historical events occurring in the targeted time frame. Third, identify any way in which information flowing among the servants could have affected the instigation or outcome of these events. And finally, create a story that you can poke at, visualize, show to others, get their critique. That is, learn by sharing a model and the data you use to formulate that model, where the model represents your understanding of what would have taken place. The core focus was clearly to be on how the model could be represented; and using identified data, executed and displayed in an interactive format; with the ability to access the implementation details of the model.

The simplistic idea was to think of this knowledge gathering in, as creating active notes hence the name NoteTaker. The not so simplistic notion, was to broaden the modeling conversation we were having, to include topics outside the physical sciences. Anyways that's my story of how the 1978 Smalltalk machine can be called the NoteTaker, and is my introduction to you of the educational goal for the next iteration of the Dynabook as a modeling environment.

I looked around in preparing this visit with you, to see if there is now software that could provide the modeling solution I was thinking about, when I stated the software challenge to the NoteTaker. Maybe you know, and if you do please let me know, but at first nothing popped up, except to be reminded of the interesting visualization ideas from Silicon Graphics and the Xerox PARC NoteCards project in the early 1980s.

NoteCards was hypertext based, and an early contributor to the notion of personal knowledge databases. When doing a search for NoteCards, I mistakenly started by thinking Stuart Card was the responsible researcher, rather than his longtime collaborators Randy Trigg, Frank Halasz, and Tom Moran.

Lambda.Books

As a consequence of one of those chance encounters that browsing the Internet should be famous for, I found a 2012 ACM SIGCHI talk by Stuart Card in which he's pitching the idea of a Lambda.Book. In this kind of book any text is computational so that the text itself can be activated, potentially to help maintain a detailed chain of causation. And thinking about the NoteTaker challenge, there's a similar idea, that the student capturing information in the library search, does so in the form of actionable text, that can itself become data used by the student, to identify patterns of relationships and information flow. Moreover this actionable text becomes part of a chain of causation, so that a reader seeing a pattern emerge, can capture why the pattern makes sense. Of course expecting any system to be able to explain itself, is likely the reason why intelligent computer-based tutoring programs are both hard and tedious to create. And why embracing the machine learning ideas of teaching the computer how to learn with statistics and probability theory, basically analyzing even crowd-sourced examples is so attractive. Stu Card also pitches something I mentioned earlier having learned the idea from Stanford Computer Science professor Donald Knuth which is that you learn to write by reading. Stu states this idea by saying that a Lambda.Book has two domain specific languages: one is the user interface language for readers, and the other is a builder language for authors. This duality is often seen in many of today's construction or gaming software, but it introduces the question as to whether the duality is actually necessary, as it is not in for example, the Squeak family of languages. I think we can agree the NoteTaker challenge remains an unsolved problem, at least, as I said, I have not found an existing solution.

Modeling and Kits

In the 1970s we began to work out the idea of a Kit, somewhat similar to the Microworlds concept. I'm going to use the term Kit when I'm talking about a specific constrained set of interacting objects, and the ways in which those objects relate to one another, and how they can be accessed and refined. I also use the word "modeling" rather than "programming" since as you now know, creating sharing and challenging veracity of models comprises a particular approach

to learning that interests me. An approach often referred to as inquiry-based learning. This approach is best understood as having the learners do experiments and observations. Learning expectation is that the students would extract some understanding of what kinds of elements are present in the observed activities, and some knowledge of how they think these elements relate to one another. This approach is the basis for the museum-based learning activities and exhibits at the San Francisco Exploratorium, where I volunteered for a large number of years.

Happily, incorporating modeling activities in schools is not new. As a short list consider what you may have done in school to learn: how your natural language is composed, how mechanical things interact, how electrical things transmit power, how social and political networks operate. How the Earth and its inhabitants evolve, how our planet exists in a larger galaxy of space. How our body works as a system of organs and connectivity. How we acquire and spend money. How human history, culture evolution, especially cultural differences, landed us where we are today. And schools do teach modeling tools: such as number lines in early math, timelines for history, mapping, graphing, and flow diagramming to name a few.

All of these existing modeling tools and modeling examples provide us with background material we need to fulfill school expectations, and to check that a proposed Dynabook modeling environment is in alignment with standard curriculum expectations for learning outcomes. But the Dynabook mission is to change the process of learning, so as to deliver the inquiry approach, and to improve the quality of education.

In my research for this talk I did encounter some papers written about how to teach beginners, concepts of object-oriented programming, which were based on specific micro worlds. But none that I thought offered new thinking on what are the critical properties of an environment for general purpose modeling, or its libraries of kits and tools.

Next I'm going to point to six different Smalltalk based kits. Where I could find video I'll show a short clip. I'm summarizing these kits, because they represent models explored in the pursuit of the Dynabook, and they inform the categories of questions with which I will end this talk.

The Dance Kit

As I said, not chronologically, so very early kit we called the Dance Kit was designed specifically for young kids. It was documented in an article in Byte magazine in August 1981. How many of you have seen that article? Ok, a little more hands around. Yes Dan I know you did. As part of what was a special issue about Smalltalk that we organized, the Dance Kit idea was originally conceived by Bill Finzer to teach the Basic programming language, but in the context of choreography. The Smalltalk-80 version was likely one of the earliest user interfaces to programming by manipulating visual blocks. Here positions composed of steps and bridges for iteration, commands, collected into routines such as the kit and jump that the learner creates. Dance acts then make use of these routines. Since the dancer object could take on any shape made up of parts, we could easily extend this dance kit into turtles boxes, stick figures and so on. It's not much believe to also represent the dance stages a maze, and direct the dances accordingly. Similar use of manipulating visual blocks representing language elements was adopted in Squeak Etoys and Scratch amongst other places.

ThingLab

Okay. How many of you know ThingLab? Oh, so I misjudged you. I thought you all knew all this stuff, so I'm kind of glad we're reviewing it. The next kit to look at then will be ThingLab. It was created by Alan Borning around 1978 or 1979, although there was an initial Smalltalk-72 project. It was his PhD dissertation at Stanford University and work that he continued when he went to the University of Washington as a computer science professor. It is a visual object-oriented simulation system. ThingLab was initially an exploration of designing a constraint oriented system in which the user provides arbitrary inputs or outputs, then the system solves for whatever is unknown. The ThingLab browser references only those objects that are specifically in the modeling kit.

So i'm going to show you three minutes three and a half minutes of Alan Borning explaining ThingLab.

[VIDEO PLAYS NOW]

This is a video clip from a keynote talk that Alan Borning gave in 2016. So if you search the Internet you'll find like an hour and a half talk and discussion. But it's running on a Xerox Alto with 64k of memory shared with the display. It's running in a browser in Smalltalk-76, and one

of the things I want you to notice about that browser, for those of you familiar with Smalltalk which I assume you are, is it's not collections, it's not all the stuff that you're used to, all right? It's only about ThingLab, so it's constrained about ThingLab. But it's huge. It has construction kit parts and it has constraints. But when you look at this video, and you see that library with such a vast warehouse of parts, you have to realize that what he's doing is scrolling to find parts. And most likely, if we're trying to scale, start with beginning learners and scale to more and more and more knowledgeable, there needs to be better filtering visualizations and search mechanisms based on pedagogical needs.

And I'm quite sure that those things now exist, because after all, we're all searching a world of information on the Internet and a lot of those techniques apply. But I am reminded, when I see this, of the question I used to ask Unix gurus, as to how they know what is available for reuse. And the answer was always the same: you just know. Right? Funny, but not going to be good enough, given our target audience. Fortunately, Internet search systems, as I said, will tell us how we can shop for parts, and get a filter onto an otherwise enormous library. Using a system like ThingLab, a teacher could have an interactive animation to demonstrate ideas in the physical sciences. There are already education companies that make animations for this classroom purpose, but if teachers can make their own demonstrations and students could modify the model, I think we can give teachers and students more input into their collaborative learning experience. What also makes ThingLab so interesting is its view that constraints, whether equations to be solved or relations to be maintained, can express partial information. They can be combined with other constraints. They can be general purpose in the sense of their properties are declarative. And they can be searched and experimented with. So these represent an important class of base objects we would need in a modeling environment.

Constraint programming is obviously not new, nor unique to ThingLab. Indeed Alan Kay, whenever explained the ideas of objects constraint based programming and direct manipulation interfaces, always showed Ivan Sutherland's Sketchpad as an important system that solved complicated non-linear systems of constraints. And of course, Bertrand Meyer has long pushed the object-oriented community to understand the importance of being explicit about constraints. Although his focus and the vocabulary of Eiffel speaks about contracts among objects in order to emphasize the need to build systems more reliably. I do think that as we consider how to lower the entry level for specifying models, we will need to make defining constraints a core capability

of the modeling tools, whether in how objects communicate or whether they can communicate at all. We're going to see an example of this in the next kit. But as we move there I want you to know one other thing, which is you're staring at Smalltalk-76 computer graphics. So this is pretty pretty much graphics of four decades ago, right?

Alternate Reality Kit

So we're going to move now to the Alternate Reality Kit. Anybody here know about the Alternate Reality Kit? So the next kit is the alternate reality kit it was written by Randy Smith around 1986, so it was implemented in Smalltalk 80. Ideas we gleam from what we call ARK. Effects are a class of constraints called Interactors or interaction laws. It's how you specify very carefully which objects can communicate with one another, and how you constrain what that communication can be. In a simulated world, Interactors can be changed as a way to modify and explore how laws may affect behavior amongst objects.

Now i'm going to run a shorter video, just for those of you who don't know ARK.

[VIDEO PLAYS NOW]

I just remind you again that this was 1986 so 35 years ago. And I think it is extremely clever work. Randy's initial work was clearly about exploring user interfaces for accessing objects in an alternate reality. When I talked about Stuart Card's ideas earlier, the object space or domain, and the domain's appropriate user interface for accessing those objects, he thought of them as a duality: it is two different interfaces. And I said, and I repeat, they're not necessarily independent pursuits. And much of what Randy accomplished was an example of how to make them more the same. And of course it would be a mistake to assume there's just one virtual world of simulation, as generality often proves to make things harder for beginners. But each example begins to teach us what is the general nature of the modeling tools and base libraries that will need to be supported in any successful solution, that will span early school years through what in the United States would be high school. Just as in learning a natural language, depending on the age of the learner, the vocabulary and sentence structures grow in quantity and complexity and span different domains. I remember distinctly one day being at a meeting, where the vocabulary was oops and such, and thinking oh my god I know what they're talking about. It just seemed weird it wasn't English but it was English, right, in terms of grammar structure. Preferably we

learn to constrain the vocabulary we use depending on what we know about the others in the conversation.

Program By Rehearsal

The next example would be Program By Rehearsal also called the Rehearsal World and unfortunately I haven't found video of this. I hope there is some, on some of the tapes that I didn't dig out of storage. But you can look in Byte magazine, june 1984 and it starts on page… I found it on page 187. A thing about Byte magazine on the internet, is that it is it's a beautiful interface to seeing the whole history of Byte magazine, and some wonderful articles are all there, and easy easy to access. This particular project was done by Laura Gould and Bill Finzer, two in our research laboratory, who created a number of interesting educational tools. Bill Finzer's interest was data centered, to look where data shows up in every subject study. Bill's early interactive applications heavily used literally point and change the data, to ask what if questions. Obviously you need to be discouraged from using professional databases. We randomly can't go changing data. He would also embed games in the context of such applications to encourage the users to analyze the games they played in order to improve their strategies.

In the mid-80s Bill and Laura collaborated on a Smalltalk-80 visual programming kit called Programming by Rehearsal that was intended for creating educational software. There were two kinds of users, there were authors of curriculum, and then there were the learners. To accommodate non-programmers, the metaphor of this kit is about performers on a stage: what is visible; and actors backstage supporting the performance: what is not visible. The stage is acting as a filter, it is what the intended user can see. The emphasis is on programming visually. Only things that can be seen can be directly manipulated by the learner, while the author has access to the backstage. The design and programming process consists of moving performers around on stages, and teaching them how to interact, by sending cues to one another. The system relies almost completely on interactive graphics, and allows designers to react immediately to their emerging products, by showing at all stages of development exactly what potential users will see. So an important quality of this kit, is that it is always working, always displaying what the model represents.

Pygmalion and Programming by Demonstration

There are many other examples of these kinds of efforts to create end user programming, or programming by demonstration, or by example, or by interaction. Pygmalion by David Canfield Smith was his Phd dissertation at Stanford in the same time frame, and others are nicely documented in a book that Allen Cypher wrote in 1992, "Watch What I Do: Programming by Demonstration". There are probably a lot of reasons to be interested in these approaches to programming. One is that the approach is more playful given the instant feedback and likely to encourage experimentation.

The Simulation Kit

Let's move on to the Simulation Kit. I didn't do a video of the Simulation Kit, but you can access a running version in Dan's Smalltalk zoo that is being hosted at the Computer History Museum in California. So an important point made by our teaching experience is that because we expect the learner to extend a model through programming new methods, not just by parameter modification, it's critical that some outcome that the learner mistakenly creates does not get explained using vocabulary outside of what has already been introduced. The obvious example from the Smalltalk system itself is its class browser and debugger. So imagine you're creating a weather forecasting model and an error occurs in execution, but instead of seeing the error at the level of weather activity, and the buttons and dials of the weather station, you see a bug in how text is displayed on the screen. Wouldn't bother you, but there are learners that would bother quite a bit, because it's not something they know anything about, or maybe don't ever want to know anything about. Certainly dealing with an error outside of a kit is a distraction, potentially defocusing the learner's modeling intentions. We had two opportunities to explore the problem of constraining content and context. In 1978 through 1981 we did work on the Simulation Kit and then in the early 90's on a system called LearningWorks. Do any of you know any of those systems? Maybe it's all new. Given this audience I thought you would know it, if you looked at the Smalltalk-80 books.

The underlying model of statistics and discrete event driven simulation is in in the original language book. But not this particular embodiment of it not the user interface for it. So a quick reminder of the features relevant to our discussion today, the Simulation Kit provides the basics

for implementing discrete event-driven simulations. Actions to happen in parallel are synced by a clock. The browser for the Simulation Kit was called the Playground. It was constrained to show only the classes in the kit library: simulation objects such as worker, customer job, and station, as well as all the classes that define a statistical package. An instance of the class Simulation manages the layout arrival scheduling, reporting, schedule and a clock to register elapsed time. Neither clock nor the reporting windows are modifiable by the programmer, except to provide implementations to a fixed interface. Within a playground the user could create subclasses of those simulation classes, as well as schedule the arrival and work processes of workers and customers. In addition the sub-views in the browser invoke their own editors. For example browsing to a view of the worker invoked a bit editor. There was of course no semantic testing as to whether what the learner writes into these methods complied with expectations. The debugger for the Simulation Kit was also constrained so that only the implementation of classes specific to the kit would be made visible in the execution stack, if an error occurs. Traversing deeper into the system, for example using control-c, to interrupt the running system, to see how display the text display was handled, a favorite demo of all of ours, was not accessible.

Deployment was very interesting. So this would be was developed in 1977, deployed in early 1978. It was to 10 top executives of the Xerox corporation. That would be the chairman of the board, the president, the chief financial officer, the head of corporate research, and so on, who in early 1978 were invited to Parc for a two-day seminar on components-based software. We were asked by the inviters, which would be the computer science lab, to prepare a curriculum and give some hands-on experience with components-based software development. The curriculum consisted of models of Xerox businesses, so that's what you see here. One of which is Versatech, there's some manufacturing simulation. The other one is the copier duplicator center.

If you looked around the room, everyone started with the same thing, and then everyone had something different. We were worried that these execs, back in those days, wouldn't wouldn't want to drive a mouse or type on a typewriter, so they were paired with somebody from the computer science lab or the systems division, just so they wouldn't wouldn't have problems. And to my delight and surprise the then president of the Xerox corporation had been in IBM and had been a programmer. So we lucked out quite a quite a bit on that. And I think Dan it was you, who turned the screens into color and hooked it up to Gary Starkweather's color printer, so that they can print things out. And we had a whole binder of explanations that they went home with.

LearningWorks

LearningWorks was an attempt to make developing curriculum, embedded into the modeling environment. The basic idea was to create a software implementation of a kind of smart book, that we called the Learning Book. Book was a metaphor for packaging a set of class definitions, to find, read, and use. The capabilities had been more typically seen visually in a Smalltalk browser or debugger, along with the user interface accessing constraints we identified in doing the Simulation Kit. LearningWorks was first reported on the talk I gave at OOPSLA in 1990, while I was still involved at ParcPlace Systems. It was then part of a contract to Neometron, a consultancy reform, and the contract was from Mitsubishi Electric Corporation. The work was carried out by myself, David Leaves, Vladimir Kubowski, Tammy Lee, Jason Mint and Stephen Abel.

LearningWorks as I said, was comprised of a learning book framework seen as windows on display screen, to emulate the structure of a book, with sections and pages. The pages contain application activities with which the user can interact, and the objects needed to realize these applications. Learning books reference other learning books as a way to specify which object definitions are to be included and which should be visible to the user. An author uses these packaging features to build a set of coordinated exercises that can limit access to objects in the larger system. Such constraint can be done at the book and also the book section levels. So there were several kinds of books: an activity book which contains the context in which the learner explores curriculum, an authoring capability consists of creating a course binder that contains the overall curriculum plan, and one or more learning books that carry out the plan. There was usually a book supporting team communications framework. It was included so students can form teams and work together on the Internet to do a project. So you see us embedding, into what otherwise was a general purpose programming environment, not just modeling capabilities in the libraries, but also curriculum delivery support. You're seeing a peer-to-peer communications framework, that supports real-time sharing of learning books.

It was based on Spline, that was developed at the Mitsubishi Electric research laboratory to enable social virtual reality in 3d. So we essentially recast the Simulation Kit into doing simulations or models executing in 3d space. This essentially required a slightly different

approach to graphical user interface editing, because of the special micro world elements: dynamic creatures that can be animated, places or cells in which in which those animations creatures live, and events that trigger creature and cell behavior. There's also a bit of complexity to enable what is constructed to be seen and what is in the background, somewhat reminiscent of the Rehearsal World.

LearningWorks was published in the 90s in the communications the ACM and other places and you could find it. But the most interesting use of LearningWorks was from the Open University. Are you all familiar with the Open University? Well you should be. It's really a very interesting place! So, what happened to us while we were working on LearningWorks, we were approached by Mark Woodman, who was then a professor from the Open University in Milton Keynes in the UK. He was starting the creation of a new Open University first course in Computer Science. The instructional format for the Open University was to use the postal system to deploy assignments. I mean they would send entire Chemistry labs to people in their homes and then teach a Chemistry lab through the mail. They had tutoring centers, that they students could go to in nearby communities. One of the things that happened here was that we there was a transition and timing, because it was the early 90s, from the postal system to using the Internet for distribution. The problem posed by Mark was that he'd be able to unfold his new CS curriculum, delivering the curriculum content to his remote students in a form in which they can minimize any perceived confusion when a student working alone is given access to a very rich programming environment. So this would have been the version of Smalltalk-80 done by ParcPlace Systems called VisualWorks. In 1994 when the OU was planning its new first course in computer science, the decision to "embrace an objects first approach", and choose Smalltalk as the primary teaching vehicle, was seen as a radical and controversial step. It probably still is.

Nonetheless, the project we started together in 1994, when released at the time of its 1999 publication, had already attracted over 10 000 students in the UK and Europe. And they were prepared to go to Singapore, and back to the US, which are all countries in which the Open University is incorporated. As Mark said this course was the largest such course in the world. The average age of a student was 37 years old, which was fitting given the Open University model to support people pursuing their education regardless of age and situation, to learn at their own pace, and to earn undergraduate as well as graduate level degrees. I had the privilege of attending a graduation exercise that was held in Ely cathedral in Oxford, and I met a lot of these

students, who were as old as 80, which Dan and I decided isn't old anymore, that's what happens! And it was fascinating because the president of the university in shaking hands and giving a degree to people who had worked probably as much as two decades to get their bachelor's degree, would ask them okay what's next, and some of them would actually say they were going to do a Master's or Phd. It's at your own pace right, and I kind of wish that through the pandemic, the the kind of fun of this and the motivation these students had, could have somehow been invoked with our younger students.

What LearningWorks offered was a way to progressively disclose details, and thereby control disclosure of an otherwise complex programming environment. Such disclosure meant the student would gradually learn to use a complex commercially available programming system. Note that in the same time frame as Mark's work, this particular issue how to support learners of a sophisticated programming environment working remotely, is dealt with an interesting paper by Thomas Kühne in about the same time frame. While Kühne was comparing Smalltalk to other choices, specifically Eiffel and Java, he added some diagramming visualizations to Smalltalk, to handle some student confusions which is in fact the same thing the OU decided to do. Our Open University terminology was that each learning book contained a microworld to be explored and modified and added. This becomes interesting to those of you into software engineering.

Remember they're teaching software engineering, but they had some software engineering issues. Some simple changes to user interface to constrain the size and numbers of windows, and to pull out learning book pages into separate windows, separate but related, so that instructions and inspections could then be done simultaneously, thereby treating them more like a notebook, or as Mark would say a project book, in which the relationships were explicitly maintained. The OU team added new tools: an html browser, special visualization tools for arguments and parsing expressions, workspaces for controlling microworld behavior textually in addition to the existing button clicking controls, and a class reporter that showed a complete view of a class in a single read-only textual view. I found that especially interesting because at one point Ted Kaheler and I had come up with such a layout for the earlier executive class session using the Simulation Kit. We chose not to use it, but for beginners it was nice to see it all laid out.

Students could create their own user interface to a micro world, made possible by the careful separation of the microworld modeling from its interaction interface. The graphical user interface tool checks to make sure it is complete and consistent, that it fulfills the expected

protocol for interacting with the microworld objects. The Open University team created a curriculum as a set of learning books to teach software engineering concepts. They have an air traffic control simulation that I found really interesting because they were teaching about professional ethics.

And they added the restriction that only one learning book would be loaded at a time, so the unfolding curriculum could include modifications or replacements to the objects defined in a prior learning book. So this raised the red flag. It was a problem for them, it was a problem for us. It's something that if you try to do, these kind of book-like or constrained browser or kit-like libraries, that are independently constructed, you have to worry about your namespace and any conflicts in that namespace. Especially if there's a group of students who are sharing their results and negotiating what to do. Unless there's some automatic way to provide some user specific naming changes, you know like prefixes or something, there's going to be a conflict. And what they ran into was you give an assignment, you get your results, each student gets working. They probably name things similarly not necessarily differently. Now you're going to give them the next layer of the curriculum and what the OU developers curriculum developers wanted to do, is give them the starting point so they all started with the same solution. Not sure that's pedagogically necessary, but anyways they did, and so then they had to replace what the students did and there's where pedagogically there's an open question. But because we were doing a whole new system, and we were working together at the same time, the OU needed a way to update the underlying LearningWorks itself in a way that new and remote learners could handle. Remember they're updating to 10 000 users. To that end the system was shipped in two parts. An unchangeable and safe image containing the underlying ParPlace VisualWorls, the definitions for the LearningWorks framework, and the core changes to LearningWorks that they made. And secondly, the learning books themselves. That comprise the unfolding curriculum. The unchangeable image offered a problem. How to patch any bugs given a large scale distribution worldwide. And it was eventually resolved by including the patches in an individual patch learning book. So lots of things to think about changing LearningWorks to do a better job of supporting the pedagogical plan of an ambitious course.

Both the development of LearningWorks and the OU version overlapped, and timing, so to some extent we had a typical coordination issue. I have to say that one of the fun parts of preparing this talk was rereading all of Mark and his team's papers and I do commend you to do

so as well. I was able to find them in the digital library from the ACM, that has some of them is open for free now so that was very helpful.

Themes and Questions

Now i'm going to turn to the final part of the talk which is: What are the themes and questions that I've raised this morning? What would I be asking if you came and said you've invented the Dynabook modeling environment?

1. Can all textual information be actionable? How work with the youngest learners?

What is the role of text? Especially where the youngest learners are involved, is text always actionable? For example clicking on a single token, does that raise an explanation, or shows me a reaction? I like that idea, mostly because when I'm in Kindle reading a book that has a tough vocabulary that I'm not familiar with, that's what I do. I touch it and up comes the dictionary explanation, So it's helpful. Do buttons with icons properly replace text for the youngest learners?

2. What are patterns?

- What contributes to the student's finding patterns of model behaviors or relations?
- Is it critical to be able to say why a pattern exists?
- Are the patterns mostly to be found in the model visualization tools, such as graphs, timelines and flow diagrams?

What precisely do we mean by pattern of behavior, or pattern of relations? Remember the NoteTaker challenge. Is identifying patterns and understanding what raises this behavior, a critical aspect, coming up with an abstract representation of a set of models?

3. How do we support the idea of learning to write by reading?

- Is the interface for how the user reads necessarily different that how the user writes?
- Is this idea the sam as designing for reuse?

We explore the idea of learning to write by reading. Is doing so a requirement for effective reuse?

4. Thinking Partners - Where will users get help?

- Is building intelligent tutoring into the Dynabook necessary?

- Or can learners rely on an Internet community?
- And do we need a base UI like that of LearningWorks?
- How critical is curriculum, alignment to school standards, and teacher training?

Where do users get help? Is help necessarily about building intelligent tutors, or built-in curriculum, or is the current climate of building Internet communities something to be leveraged? I shouldn't say or I should say? And if we start by teaching users how to effectively use debuggers, do the users have the tool they need to get help?

5. The Objects and Messages Paradigm

- Offers and easy way to talk about elements of a model and how they interact
- Do we have to explicitly teach object-oriented programming? If so, when?
- Provides an understandable way to create visualizations of, and interactions with, related objects. Is there good support for defining views and interactions?
- Is there ever a need for a hybrid approach to the underlying programming metaphor?

It seems clear, but i'm biased, that the objects and message paradigm serves the educational inquiry-based mission quite well. Is there a limit? Do we have to specifically teach object-oriented programming as how we talk to a computer in order to use the modeling environment? And if so at what point do we start that parallel educational track? Oftentimes, the way we support pairing user interfaces views and interactions with underlying models can get quite complicated. Certainly an initial encounter with VisualWorks, which was designed for professional developers might testify to this concern. For pedagogical purposes, should some simplifications, to progressively move towards the full system capability be considered?

6. What role does the ability to declare constraints play in model creation?

Should the base classes of the environment support constraint specification?

Is the notion of constraints just another set of class of objects, or some new first class category of base class definitions? There are many ways in which constraints likely need first class support in the modeling environment as gleaned from past projects. I list some here but probably missed a few:

- Maintain how specific properties of one object constrains the behavior of another

- Maintain constrained relationships between classes of objects or between specific instances
- Declare whether particular objects are visible to other objects, can communicate with those objects, and are restricted in behavior by the properties of other objects
- Hide user access to implementation details outside that which pertains the elements of the kit itself when creating, looking to reuse, or to debug
- How do you declare your constraints? What is the nature of that change to how you program?

7. How should we provide explicit support for Reuse and Programming by Refinement?

- Help in finding and reading (including playing with) existing models
- Help in expanding knowledge of modeling vocabulary and grammar
- The object and message paradigm excels when writing is paired with reading
  - Write by using existing, tested objects
  - Write by adding to object behavior
  - Write by subclassing/overriding an implementation
  - Write by adopting existing behavior (message sets) and creating new implementation

Much of the Smalltalk system was about supporting reuse and programming by refinement. Will the approaches that work for professionals suffice, or need to be supplanted for younger learners? Do the common Internet search capabilities offer new ways to teach how to find existing models and views that compare with these models?

8. Modeling Kits as Curriculum

- What are the specific ways in which kits are created, packaged, explored, modified and shared?
- How do we support kit creators in deploying curriculum that adhere to curriculum design principles?
  - Principle of Constrained Disclosure

    Outcomes that the learner mistakenly creates through modifications, inappropriate data input, or original creation, cannot be explained using vocabulary outside of what has been defined as part of the kit.

- Principle of Progressive Disclosure

    Hide user access to implementation details outside that which pertains to the elements of the kit itself when creating, looking to reuse, or to debug.

    Be able to dynamically change the filter on what is hidden and what is visible.

I brought up two curriculum design principles during this during our discussion. Should the modeling environment provide a curriculum deployment framework for explicitly adopting the principles of Constrained Disclosure and the Principle of Progressive disclosure? This may look sort of obvious: what you show in a browser. But as you're opening it up and more and more, the complexity of just quantity of objects can be overwhelming to new learners. It's not compartmentalized. Students get compartmentalized. They go from a math class, to a science class, to an English class, right, but in this world that kind of compartmentalizing doesn't happen.

Some Additional Thoughts

Frankly we would only know for sure if what's being proposed for the new Dynabook works, if we actually build a multi-year modeling curriculum. This is modeling projects for children of all ages. And see what's needed to support the kind of modeling that should be a part of the way we educate our children. The key to future success is by taking on this curriculum challenge iteratively, building the modeling environment, the libraries, and the tools, until we have something worth criticizing.

But I do want to throw out some things that come from modern computing existing ability to use sensors and remote controls. Again for pedagogical purposes, creating models that drive physical devices like robots, can be a lot of fun, and the world of school is full of these kinds of activities and contests worldwide .

Perhaps the big, the obvious we need to remember, is that information used by these models will most likely come from direct access to internet information knowledge bases, and how do we make that easier, how do we facilitate that. It appears that most things needed for a good first modeling environment have been experimented with, and also broadly distributed. Maybe it's just the packaging that's missing, or testing the uses that are missing. But the challenge is that this packaging involves coming up with a new user interface, that can progressively unfold from something simple and manipulable enough for young kids, repeated in multiple model kits, yet

evolve gracefully for more experienced modelers. And it does seem that ways to connect a model to Internet data gathering, still seems something to be determined. And of course, success means solving the NoteTaker modeling challenge!

Remember:

The critical idea is that the Dynabook is the medium in which people can share their understanding of how things work, and be challenged as to whether that understanding reflects an approximation to reality.

When the mission is to upend how kids are educated so they can be intelligent participants in a growingly complex global community, then we need to have the full content of such an education, not just the context and tools. And notice that I'm not saying they have to go to a building called school.

So I'm going to end with that, and I understood that Richard announced that there'd be a Q&A, so I wrote "maybe A", because maybe I don't have the answer so we'll find out. Thank you.

Questions and Answers

Q: (Inaudible question)

A: Did everybody hear him ok? So you've asked a very complicated question. As you know, a very complicated issue, and you didn't say that the social consequences were good or bad. I mean they're going to be good and they're going to be bad, right? The classic one that's being argued by the media right now, the media is never the best place to look for how to think about things, but nonetheless, has pointed out that things, as some of the AI machine learning capabilities of face recognition, have positive and negative. The positive is instead of fussing like we did last night at the airport, kind of after customs. In the U. S. we walk up to a screen and does face recognition if you have global entry, and boom you're done. And you're out of there because they recognize you. Now there's terrible consequences of that kind of recognition. The downside is that facial recognition works primarily for white white males, right? And so when the police use that face recognition they're picking up people who are innocent, so there's a negative side of that. So I'm assuming you're talking about how do you educate people on how to think about the various ways in which technology is impacting our lives, and how do you educate

people about that, and bridge a gap to people who don't care, but don't understand that they should care. Is that was that part of your question?

Q: (Inaudible question)

A: What if you had a model of how people interact worldwide, which doesn't exist. I mean, an incorporation of multiple cultures, and how they interact with one another. And someone makes a proposal that is potentially disruptive. Before you could deploy that, would you match some way in which they'd have to put that as a change in the model and see what happens, and show people the consequences of the good and bad? I mean those are possible things.

But I do want to tell you a story, of the first time understood the consequences of going around giving talks on youth. When we would early explain object-oriented technology, and this would be early 80s, one of the things we would point out is that if you want to know how something works, you hit control-c and the objects interacting are there, and you could start finding those and browsing around. There's a division of Xerox that did a lot of government contract work. And they wandered through our research lab, this was early earliest graphical user interface, and they had Xerox machines. And they pitched to the people in the government they worked with: Hey let's try and build what it what your ideas are. But there was a big communication step. It's a big communications gap when somebody has in their mind a vision of how something could work, but it's not something you can verbalize. And they got an idea by looking at what they were seeing, which was a programming environment, but they had an idea of how to model their own work. And I found out that this division was doing this by chance. At the time we moved from a research group to a full laboratory, we had a lot of stuff going on at once. But they really did have this division an obligation to tell the research lab manager, which was me at the time, what they were thinking of doing, because this is a corporation selling something with no support no maintenance, right? So that caused me to go see what their customer was thinking of. And I had the most amazing experience, which was consequences of giving talks that I hadn't really understood. A man, who was a by trade a photographer, had an idea for how the government could analyze incoming constant data, that he can only express by building. And, correctly as it turned out, he believed that he could reuse all these different components and went searching around, and made a new user interface. Different than ours, but reusing everything. And I was kind of surprised that a non-programmer could get as far as he got. And I said can you show me how you implemented that what you did? And of course the

code was childlike, right? And quite inefficient. It wasn't going to scale for deployment. He said it doesn't matter. They're never going to deploy what I do. I'm just showing them what I want. And those guys will be contracted to build it properly, to rewrite it properly. It which would take a bunch of texts and make it this way, right? It was just not good knowledge of control structure. And I went "oh!". And that's what happened. And that system was very publicly shown and shown worldwide. And had a very high impact on people's thinking about not just what's doable with this new technology, because now remember we're talking early 80s. It caused a mild explosion, from the government back to Xerox, because they wanted to know how this was going to be supported as a real product. And as a result was one of the big motivators for spinning out and creating powerful assistance. But it taught me a lesson: Be careful what you say without testing and finding out what was going to happen. And I was impressed, and it was funny that it was like that, because it was exactly what we did with the kids in Smalltalk-72 and 76, when we went to the schools. I didn't think everyone should program. But I did think people should have ideas, that some programmer they can work and collaborate with. And that's how the kids work. Someone was on the keyboard, and some were saying "yes but what about this and what about that". They were doing a lot of animation systems that way. So you learn that the team is not all programmers, right? It's a mixture. And what i'm trying to think about is, how to turn these two sides of the same problem into a collaboration. How to not just do check and balances, but evolve, evolve together. I don't know the answer. It's a hard thing to think about.

Q: You mentioned searching the Internet for information. The Internet is a great source of good information. It's also a great source of really bad information. How can we teach kids to distinguish that this is good information versus bad information?

A: So you know I don't have an answer to that one. And what you don't know is that I live now in Montana, not in Silicon Valley anymore. So the people I live around are not like me. And we were talking at breakfast about this. And to tell you the truth, I can't figure out how they think. I can't understand how they could look at clearly scientifically nonsense, and actually buy into it. And we need to do that, you know. As technologists, and especially if we think we're on an educational mission. See, I think that if the kids are taught to question, and build a model of what's going on, and have a way to be challenged on the model they build, and that's their lifelong educational model, not rote learning, we might have a way to grab on to exactly that problem, right? We'd have a way to grab on to that problem. And it is a lifeline.

Dennis Bartels used to be the executive director of the San Francisco Exploratorium at a time when, as a member of the board, I also chaired the program committee. We were doing pretty interesting and now published research on how you design exhibits that allow people to say what's going on, and understand what's going on. About the same time Alan Kay used to have these educational discussion sessions that he'd organize. And I went to one. And there were all these brilliant people there, Marvin Minsky and others, and I said: okay well, how did you become the thinker you are? How did you learn? And the agreement in the room was, that whatever they did, they were building a model of how they thought things worked. And then a parent would say "You forgot this data". "Does this data work in your model?" And the parent would know whether the answer was yes or no. Especially if it was a no, it causes you to think about it a little bit.

And I know, one of my daughters is a physicist and had done some very surprising work, that had gone to publication to Nature Materials. And they said: "But your data, your experimental result, rocks the theories. It changes the theories, so your paper has to have the theorists comment on that." It was the best peer review I've ever seen. So she turned around, and there were two contending theories. She got each of them to look at her data and they all came back with the change to their models, because of the data.

So the reason I emphasize this, is that I have this private hope. That with building a model of how you think how things work, forcing people to be explicit about how they think things work. If you then could take the conspiracy theories that are so dominating the United States right now, and try to fit it in the model it, would break, all right?

So you can only hope and maybe not in my lifetime. But I do think we have an obligation, as technologists, to worry about these things. And to worry about those uses. And to break what has become of a very problematic way in which at least, I won't say worldwide, at least in the United States has become so problematic.

I remember at Stanford, when I was a student from University of Chicago visiting, and then working at Stanford before I went to Xerox. I can still picture this one guy who was doing a math education degree. And somebody else in his program saying "but what you're saying makes no sense". And going through the most logical: Okay here's your assumption you, know, where does that lead us? And of course it led logically to contradiction. And this young man going: "I don't care. It's what I believe." So even though if we could do all this, you're still going to have

that “I don't care, it's what I believe”. And that's sad, but it is. If the whole world where like you and me want to be, wonderful. It doesn't work. It doesn't work.

Q: (Inaudible … tool to teach my children to learn about grammar … do you have a recommended tool that make up…)

A: Everyone heard the question? No? So what I understood the question was: Today, given where we are, is there some recommendation of what we would use to start with kids?

So in the light of story, the answer that is not necessarily. It always depends on who the child is, where they're coming from, who's going to tutor, what's going to happen. So I'm going to give an example. And it's as always, it's a example of cautionary tale. So at some point, my grandkids were all sent Scratch books. And they all had computers at home, and they all had parents who were skilled enough to make sure that they got everything set up. And my eldest granddaughter showed up to visit in Montana, computer in hand, to show me that she had done all the projects in the book, and very proudly said they all work except the last one. And I asked her, you're going to appreciate this Cat, I asked her, “well why doesn't it work?” and she said “I don't know”, because she didn't have anyone to help her. I don't know why, her parents were busy. And I said “do you want to know?” It's always the first step. And she said she did. And I said “okay, so let's let's read what you wrote”. And obviously, that version of Scratch, this was 10 or 11 years ago, but it was based on the book that the MIT crowd had done. She said “well you know it does work some of the time, but not all the time”. Which of course was the hint. Because computers are not thinking. They're doing whatever you tell them to do, or you don't tell them to do. And so we looked and we just started reading it, at which point I put it down and I went, oh I understand the problem. Because this whole world was Mitch Resnick's idea of modeling parallelism. So all the objects exist at once, and if if you tell them all to initialize, they all initialize at once. But there's no clock, there was no simulated time, right, so it was random. So sometimes the cars and the people were synced and sometimes not. Now that's all I had to say to her and she went and fixed it, right? The problem with that is and what it always says to me, is how important the teacher is in this story.

You want to let the kids just go on YouTube and get tutorials. Which I do all the time. I design sweaters and knit them, and you know looking for stitches. Or I do basket weaving, you know a lot of craft stuff. And I get it all from YouTube. And I have a grandson who built himself a pretty amazing gaming computer. And I said, how did he talked us into letting him buy the

components he wanted, which was an investment. And it's sure enough it was all YouTube videos. He started showing me all these videos, at which point I realized I could teach myself how to change the oil and oil filter in my ATV using YouTube, and I was very proud of myself for doing that. And it wasn't that long ago.

But that teacher, in this case it was me, knew enough to be able to think about the model in which the language they were using, the kit they were using was implemented, to understand that there was a constraint in thinking, right?

So whatever tool, I'm not answering your question directly, but I am saying you have to be prepared to let the kids explore and say what they want to build. But at the same time you have to be capable of understanding it, so that you can help them. And I do know I remember we used to have kids come to Xerox all the time. The biggest challenge is if I would say to someone "what would you like to program? what would you like to make?", the kids weren't very good at that, because in school they're told "make this". The adults were full of ideas, and so the problem was saying to an adult: " You know you're you're asking to do something that's quite large. And in order to feel successful we're going to have to break this down and do things in pieces." And you're going to have to trust the teacher to help you pick those pieces. And adults most of the time will trust you to do that, and if you pick right you can get pieces that are interesting enough to keep them engaged. But it's why I think having a foundation of thought through tested curriculum matters. And not the same one all the time, all the time. Not turtle geometry every time you turn around. Even though it's very engaging and my favorite was the Button Box. You know what the Button Box is? It was all that turtle stuff but in buttons for little kids that Radia Perlman did. Just loved it.

But you just you have to be careful the order in which things go. And if you if you as a parent want to encourage your kids. If you have curriculum that you know people have done in it. And if we could prove it actually helps them do something. I mean there's so many claims, from anecdotal evidence, that teaching programming helps kids be better thinkers. I wish that were true. But you know, the skeptic of me says where's the proof? And where's the proof?

Q: … we learn classification also learn composition, but with the development of programming technology and also we try to teach kids more advanced topics, need to first provide first-class concepts like the dynamic of autonomous objects. My question: Should we

introduce new concepts or some secondary or first-class concepts into Smalltalk, into our library too, to help kids to master more advanced computational thinking?

A: The answer is obviously yes. I mean, I've already pitched that I think that if we're going to do modeling and simulation, that the whole idea that the behavior of an object is constrained in some way, is itself a first-class idea. And if we're serious about a modeling environment and not just embodying general purpose programming, sure! I mean I could write all that, but then it isn't at the core. And I think it's important to understand well, when you do that, what does that really mean. When we first were engineering for supportable Smalltalk-80 at ParcPlace, one of the first things that popped up was the need for exception handling to be first class. As an example of what needed to be folded in. And that work was non-trivial. I mean, you just don't leave that to somebody else to then distribute it. And of course we have a beautiful world right now, of everybody sharing everything, you know. And having communities of people saying "look, this needs to be first class". But this Smalltalk world no longer has, like Python has, you know this guy, who's the ruler of the universe, right? You know, who makes makes all those decisions. So we don't have that kind of an organization, which came later, and even conceptually came later, to make things first class. But that's why I think of it: let's not call it a Smalltalk programming environment. It's a modeling environment. Something new. And then doing the experiments, and the curriculum, and people can do curriculum examples.

David Leebs always would try to teach me to start with the simplest. Can we do the simplest thing? You know and then what's the next simplest thing? So it's the, you know, prove 1, prove n, prove n+1, right, type thing.

Q: …but in fact till now no specific language can really support the programing of an agent. So that's a big deal… There are some simulation tools to make dynamic graphics or animations, help to implement agents… I'm advocating my model it's called the economy model and we include environments with classes, agents, roles, and groups and of this… based on the roles, agents play roles to access objects.

A: Yes it's a little bit like Randy's Interactors right? That's a particular role, which is finding objects, and giving them permission to interact with one another. Which is a very interesting idea. But one of the things, you know I'm a believer, that for people to be interested in a technical solution they need to see why. And again I'm driven by teaching certain ideas, so that it's in kids dna that as they grow up they're always asking "wait a minute, what does this really mean?"

“what is the description of this as a model?” “and what are its implications?” But there are other obviously more professional or, I don't want to seem more grown up, but more professional ideas about how you motivate.

Closing

And we're going to end, because I think we have to, with one last story. You know motivating just object technology was also non-trivial. I would get phone calls from desperate students at universities. I remember one from the university in Florida: Can you please call my professors and get them to teach us an arbitrary other language? You know, like C++ or Java, instead of Smalltalk. Because we can't get a job with Smalltalk. And I would say to them: I don't understand are you at school for vocational training, or for education? And I would never get an answer back, right. Because you know at some level it's all the same.

But my story is, there was a conference in Washington state, in which the keynote speaker was Steve Jobs when he was at Next. And he gave a whole speech explaining why object technology was interesting and important, because it was his business proposition at the time. But he very carefully told the audience: I'm not telling you that you should redo your business accounting system, you know. You have an accounting system, it works. You don't need to redo all that.

And the next speaker told me this story. This was a man running a big project, at one of the large car automotive companies in the U. S. And he said we're rebuilding our accounting system. And his reason was fascinating. And he said because what we'd have doesn't work. Our customers are our workers, and yes they are unionized but that doesn't matter. I mean it's a clear voice that compensation plans need to change over time, and we are always in a fight with them, saying no. Not because we don't think their ideas are good, and not because we don't want to give them that compensation, but our accounting system isn't changeable, right. Not predictably, not reliably. So it's a case where the system works. The spec is there. It’s a running system. But they wanted to get to yes with the unions. And I just love that story. I just thought, Good!

And another reason why I loved it is Jim Davis from Sun Microsystems was their very first salesman, and he said to me you can't pitch change and changeability and predictability of change as the benefit of object technology, because people just can't build their systems in the

first place. And you know, and he says: It's not going to work. And it did you know. Obviously it did work. But he had his point, which is the the old style of building large systems was to never reuse. Especially in very large expensive government projects, where the so-called compartmentalization of apis existed. He said so the business model for typical consultant pitching to the government was four years 400 people to get to something that you can criticize, and you know is wrong, and so then we'll get an extension to our contract. And I had one of those companies literally screaming at me in the hallway of a meeting saying you're ruining our business. And I asked them, was that their business? And he said yes. And I said yes, we're going to ruin it. Mission accomplished! Yes, okay. We're done. Thank you very much.