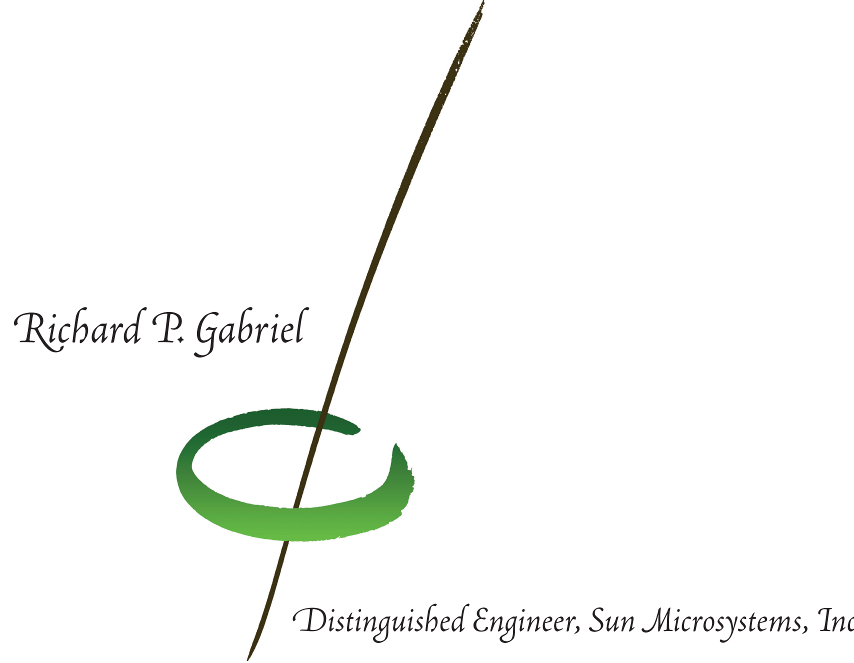
The Road Not Taken



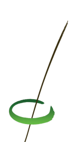
Richard P. Gabriel

Distinguished Engineer, Sun Microsystems, Inc

Whither Software

Significant changes are on the way because:

- The nature of software we need to build will be different
- The `.com` collapse is partially a failure of software, especially programming language design, and we should fix it
- Security failures in Operating Systems are mostly failures of programming languages, and we should fix it
- The way we "teach" programming is an embarassment, and we should fix it
- A growing lack of confidence in the underpinnings of current computing models is forcing rethinking



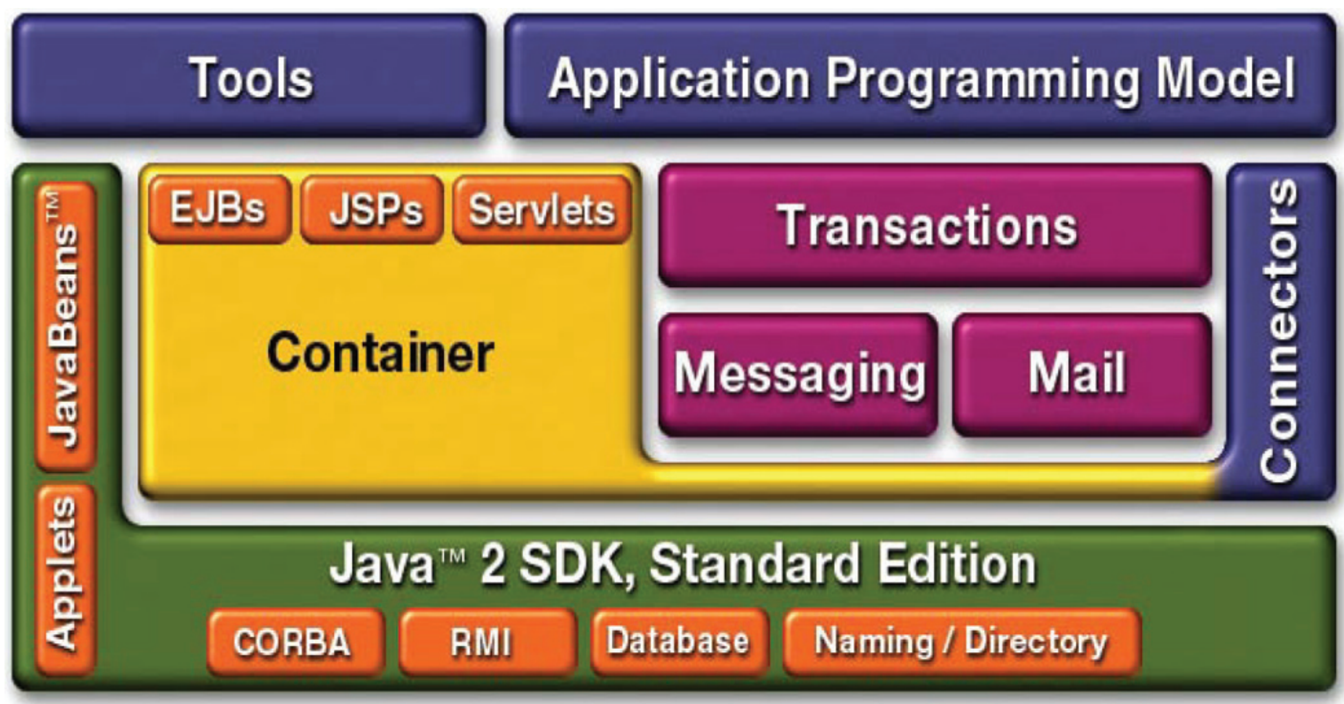
The Nature of Software

Nature of Future Software

Software will move from:

- Vertical to Horizontal Systems
- Programming Interface to Protocol/Language
- Monolithic to Distributed Architecture
- Monolithic to Mob Development

Vertical Systems

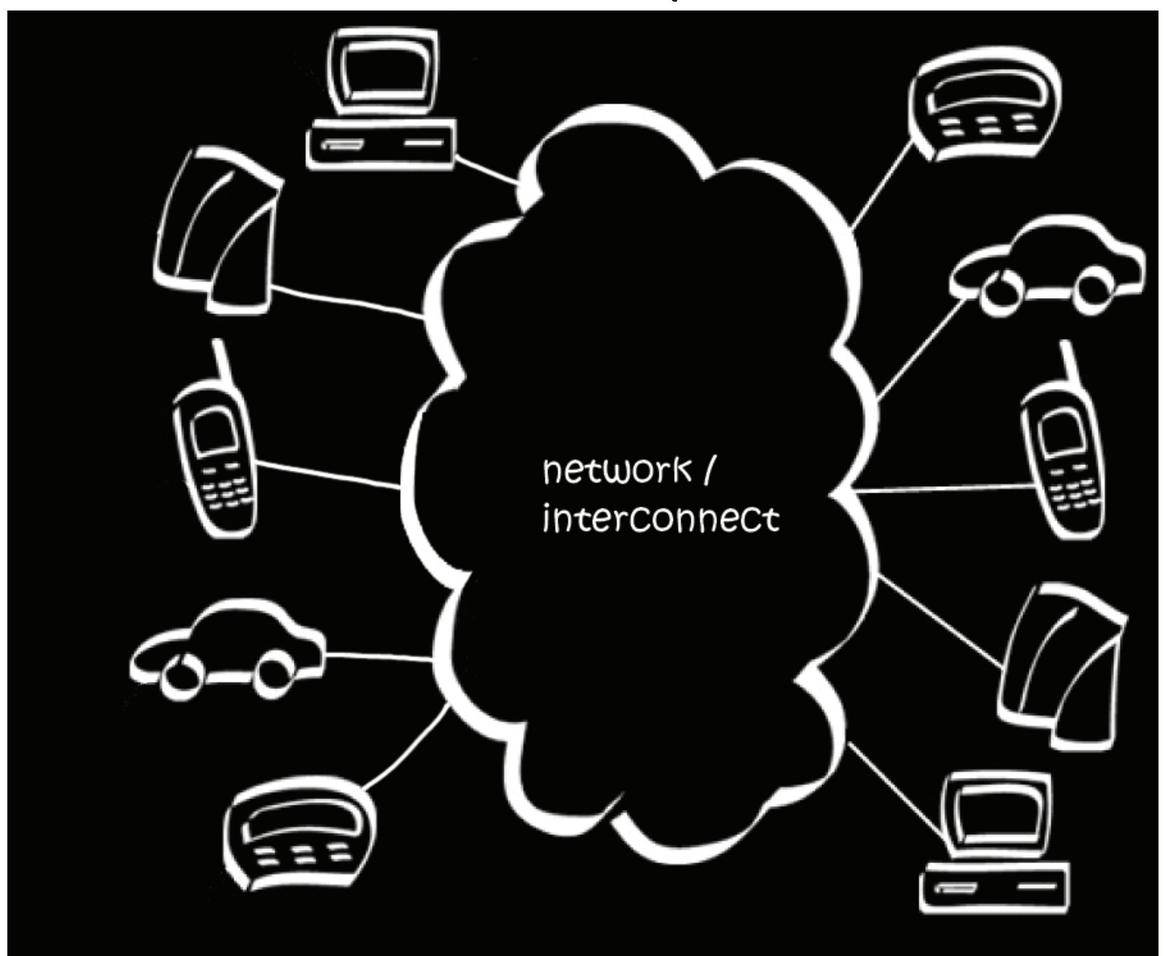


- Monolithic systems
- Client/Server
- N-Tier

Vertical Systems

- Hierarchical
- Deep interconnections based on programming interfaces (APIs)
- Deep interdependencies based on common assumptions, design, and architecture
- Deep dependency on underlying programming language and programming model
- Brittle because programming interfaces must match exactly, by position and type with no coercion or intelligent intermediaries
- Requires extensive testing or proof of correctness
- Uniform software development methodology is required to minimize human error (chosen from a plethora of such methodologies)
- Jigsaw puzzle

Horizontal Systems



- Distributed, peer-to-peer
- Edge computing, end-to-end

Horizontal Systems

- Heterarchical, polyarchical, fractal
- Shallow interconnections based on protocols
- Shallow interdependencies based on simple interconnect
- No dependency on underlying programming language and programming model
- Resilient because partial understanding of protocols is possible
- Requires only isolated testing or proof of correctness
- Variety of software development methodologies can be chosen according to local needs
- Collage, postmodern

Programming Interface versus Protocol/Language

Programming Interface:

- Usually fixed, positional arguments
- Function-calling metaphor, though perhaps no value(s) returned
- Types must match exactly or by subsumption
- All communication is through programming language types—INTs, strings, arrays, etc.
- Full understanding or no understanding—all or nothing
- Usually not flexible or extensible
- Mathematically sound

Programming Interface versus Protocol/Language

Protocols or Languages:

- Parsing and meta-data are possible—in fact, any amount of automated reasoning can be applied to understand interactions
- Speech or interaction metaphor
- Types are manifest or constructed from descriptions
- All communication is through common representations—text for example
- Partial understanding through defaulting, deferral, or partial implementation
- Almost always flexible and extensible
- Goofy

Monolithic versus Distributed Architecture

Monolithic Architecture:

- Tight integration—requires planning to put parts together, and all changes move forward, rarely backward
- Programming language-based control and communication
- Requires a coördinated development methodology to get timing down ***or*** a highly incremental development methodology to minimize divergence—but the latter is generally despised by management
- Everyone uses the same language—uniformity
- Tempts people into master planning
- Dominated by resource concerns

Monolithic versus Distributed Architecture

Distributed Architecture:

- Loose coupling—parts can be added when ready and changes can be undone
- Protocol or language-based communication and, perhaps, some synchronization mechanisms
- No coördinated methodology required for the entire system—in fact, it would be nonsensical
- Anyone can use any language—diversity
- Forces people into piecemeal growth
- Can be dominated by timing and coördination concerns

Eight Fallacies of Distributed Computing

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

Monolithic versus Mob Development

Monolithic Development:

- Master planning and control
- Chokepoints
- Integration planning
- Shared culture required
- Planned releases force misfits of time and effort—for some modules and components, developers are squeezed for time, for others, developers languish or move on prematurely
- Tends to exclude users from design
- Correctness
- No room for artistry

Monolithic versus Mob Development

Mob Development:

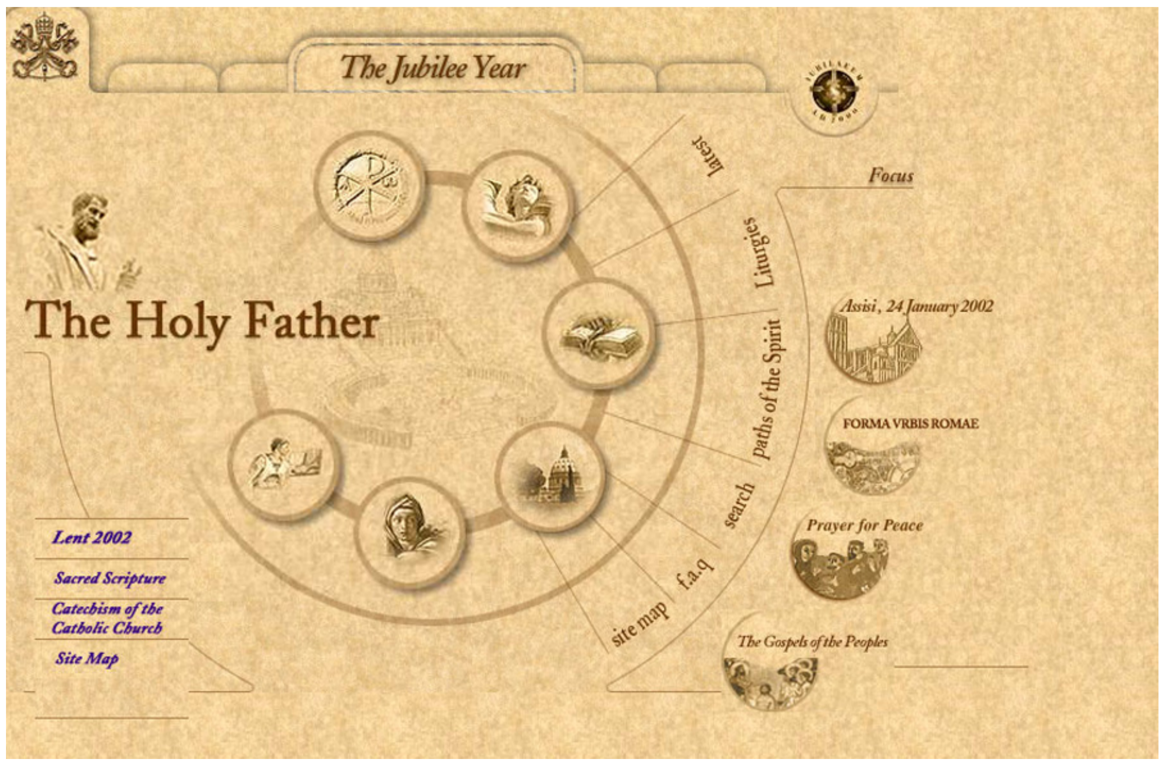
- Piecemeal growth and leadership
- Shared vision is desirable
- Few coördination points
- Can include users
- Comfort
- Can allow for artistry

Diversity, Distribution, Artistry, Users, Protocols, Piecemeal Growth

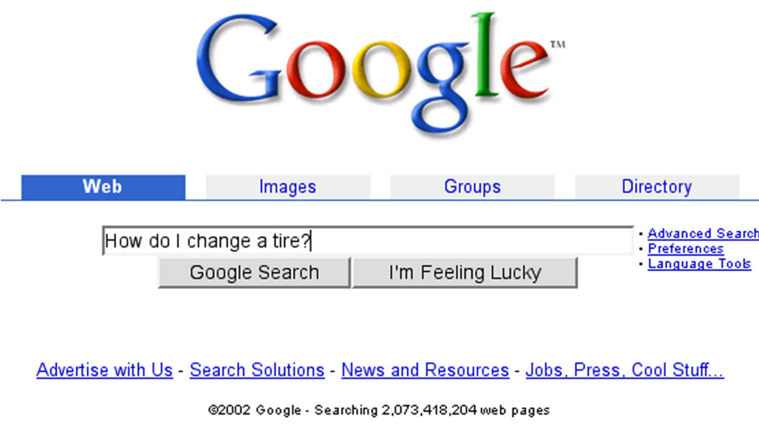
```
<html>
<head>
<title>The Holy See</title> ...
<script language="JavaScript">
<!--
function MM_reloadPage(init) {  //reloads the window if Nav4 resized
  if (init==true) with (navigator) {if ((appName=="Netscape")&&(parseInt(appVersion)==4)) {
    document.MM_pgW=innerWidth; document.MM_pgH=innerHeight; onresize=MM_reloadPage; }}
  else if (innerWidth!=document.MM_pgW || innerHeight!=document.MM_pgH) location.reload();
}
MM_reloadPage(true);
// -->
</script>
</head>
<BODY background="img/sfondo.jpg" alink="#000000" vlink="#000000" link="#663300" TOPMARGIN="0"
   LEFTMARGIN="0" MARGINWIDTH="0" MARGINHEIGHT="0">
<table border="0" cellpadding="0" cellspacing="0" width="641">
  <tr>
    <td align=left valign=top colspan=6><a href="index.htm"><img src="img/chiavi.jpg" border="0" hspace=0
   vspace=0 align=left alt="Vatican - The Holy See" width="69" height="64"></a>
      <map name="FPMap0">
      <!area href="liturgy_seasons/christmas/index.htm" shape="rect" coords="164, 33, 317, 60">
      <area href="jubilee_2000/jubilee_year/novomillennio_en.htm" shape="rect" coords="160, 32, 326, 62"></
   map><img src="img/annogiubilare_en.jpg" border="0" alt="The Jubilee Year" hspace=0 vspace=0
   usemap="#FPMap0" width="495" height="64"></td>
    <td align=left valign=top rowspan=12 width="100">
      <p align="left"><a href="jubilee_2000/index.htm"> <img src="img/giubileo.jpg" border="0" alt="The Holy
   See - Jubilee 2000" width="81" height="96"></a><a href="jubilee_2000/index_en.htm"> <br>
        </a><img src="img/ddx.jpg" border="0"><br>
        <img src="img/vuoto.gif" width="3" height="125" border="0"> <br>
        <img src="img/fil_brown.jpg" width="175" height="1"> </p>
      <!--
  <p align="center">
  <img src="img/vuoto.gif" width="17" height="220" border="0">

  <a href="multimedia/wydphoto/open_en.htm"><img border="0" src="img/multim/gmgfoto.jpg" width="139"
   height="43" alt="galleria fotografica"></a> ....
```

Diversity, Distribution, Artistry, Users, Protocols, Piecemeal Growth



What is the Effect of Mob Development?



- Billions of Web pages, each a small program, written using a simple programming language, and relying on simple text-based protocols
- Does it solve the Turing Test?

> *I propose to consider the question "Can machines think?" This should begin with definitions of the meaning of the terms "machine" and "think."*
>
> —Alan M. Turing

What is the Effect of Mob Development?

Learn2.com

The new way to learn.

What do you want to learn2? Go! Select a Channel Go!

Ask a Question | Home | Free Learning | Join Us | Online Courses | 2torials | Learnlets | Help

Automotive

The Steps:

Intro: Before You Begin
Step 1: Find the right spot
Step 2: Get the spare and the jack
Step 3: Loosen the lug nuts
Step 4: Jack up the car
Step 5: Remove the old wheel
Step 6: Put on the new wheel
Step 7: Lower the car and pack it all up

The Necessities:

The jack in your car is in there,isn't it? Check before the need arises)

A properly inflated spare tire (Again, check to see4if it's4there)

lug wrench that fits your wheel

lug bolts (see Before You Begin)

Optional

A plastic tarp

2torial #0621:

Learn2 Change a Flat Tire

Fix that sinking feeling!

Few people like to change a tire. But if you can follow simple directions, this is a good opportunity to revoke your "mechanically incompetent" status. You'll also save time, money and stress, and your passengers will consider you the greatest thing since sliced bread. With this 2torial you'll be back on the road in a jiffy.

Before you begin...

To change a tire (or more accurately, a wheel with a tire on it), you need another to replace it. Many car owners haven't checked the spare since they bought the car.

- **Take the time to look for your spare tire.** Under the floor of the trunk and under the rear of a truck are the usual spots. Whether it's a full-size tire or one of those small, low-quality, high-pressure ones, make sure that it's properly inflated and easily accessible.
- **While you're exploring the car,** check the car's jack as well, especially if you bought the car used. It's not uncommon for the jack to be missing or incomplete. Find that out now, before you need to use it.

Community
Forums
Shop Now
Free Web Pages
Talk 2 Us
Join Us

Click here to send to a friend.

Related 2torials:

#0508 Jump-Start a Car
#0518 Understand Tire Care
#0601 Change Your Oil
#0696 Install and Remove Snow Chains

The .com Collapse

.com Collapse

- eCommerce requires adaptable software to handle changing business conditions and models
- From January 2000–May 2001:
  - 374 companies were delisted from NASDAQ
  - on average $5–$10m was spent on computer infrastructure
  - of that, $1–$5m was spent on software development
  - in many cases, the software was not suitable and not adaptable enough for real business situations

.com Collapse

ZoZa.com is typical: the first proprietary apparel brand launched online selling high-fashion sportswear—hop off your mountain bike, pop into the Porsche, and off to the Pops.

- ATG Dynamo running on Solaris—eBusiness platform
- Oracle 8i
- Verity search tools
- A lot of custom, stand-alone Java glued everything together
- The bulk of the ATG work was outsourced to Xuma—an application infrastructure provider
- The production site:
  - 2 Sun Netras doing web services via Apache/Stronghold (web server/secure web server)
  - 4 Sun Netras providing an application layer and running Dynamo
  - 1 Verity server
  - 1 Sun Netra providing gateway services to fulfillment partners
  - 1 Sun Enterprise 250 running Oracle as a production database
  - Staging: 2 web boxes, 2 logic boxes, 1 Oracle box, 1 Verity box
  - Development: one big Sun Ultra 2

.com Collapse

The CTO of ZoZa says:

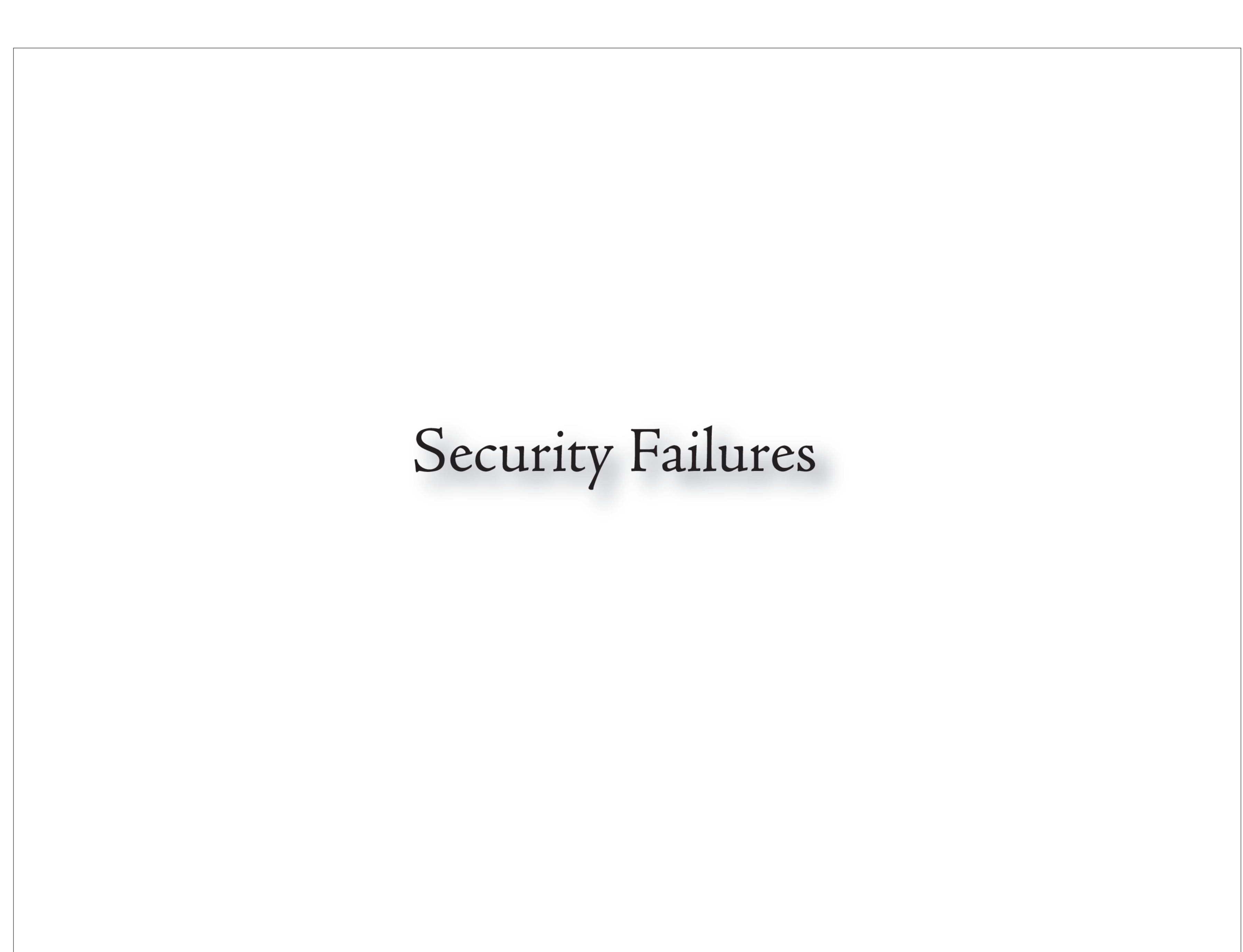
> *We built a good e-commerce platform, but unfortunately sales were slowly building just as the dot com economy collapsed. We had built a company to handle the promised phenomenal sales based on the Ziegler's self-promoted public profile. That never happened. The* ***costs of building our sales and fulfillment capabilities****, combined with ZoZa's lack of credit in the apparel manufacturing world caused us to go through SoftBank's $17 million quite quickly.*
>
> *Sizing was a terrible problem for ZoZa. Not only did the clothing get designed for ever smaller people, but even then the sizing was highly variable. Sometimes only a specific color of a product would be whacked out.*
>
> ***We ended up building a separate database table for sizing anomalies.***

.com Collapse

[The Zieglers were] *disappointed by the Web. Initially, they planned to use virtual-reality technology so customers could mix-and-match items and feel like they were trying on clothes. They also wanted to provide a personal assistant to shoppers who could recommend items based on an individual's coloring. But the Zieglers scrapped all that when they found that* ***the technology ruined the shopping experience*** *because it took too long to download. "****The medium is far more rigid than we imagined****," says Mel* [Ziegler].

Patricia [Ziegler], *a former newspaper illustrator who designs many of the clothes, was* ***put off by the poor quality of colors on the Web****. And* ***she was really bummed to learn how complex it was to swap out items that weren't selling well****. "We have to change 27 to 32 different links to swap out just one style—from the fabric to sizing to color," she says.*

So the Zieglers have gone back to basics. ***Although it cost roughly $7 million to build, their Web site is, well, stark****—a picture of Zen minimalism. Navigation is simple and uncluttered. Nothing exists on the site that can't be optimally used with a standard 56k modem. The one concession to flash comes in the form of so-called mind crackers, which are Zen sayings about life hidden behind little snowflakes that have been sprinkled throughout the site.*

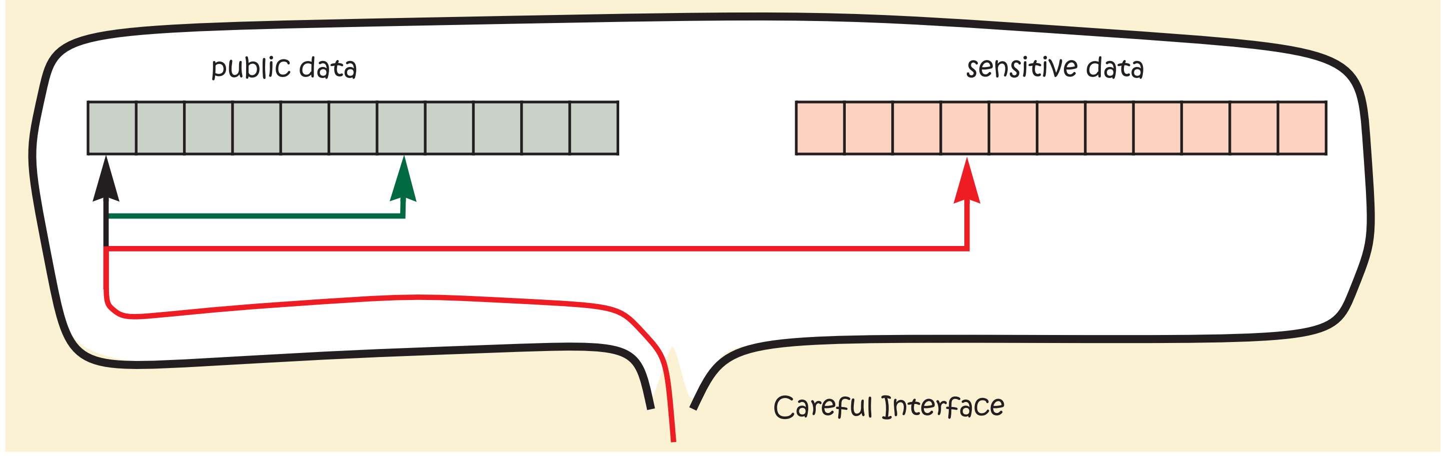

Security Failures

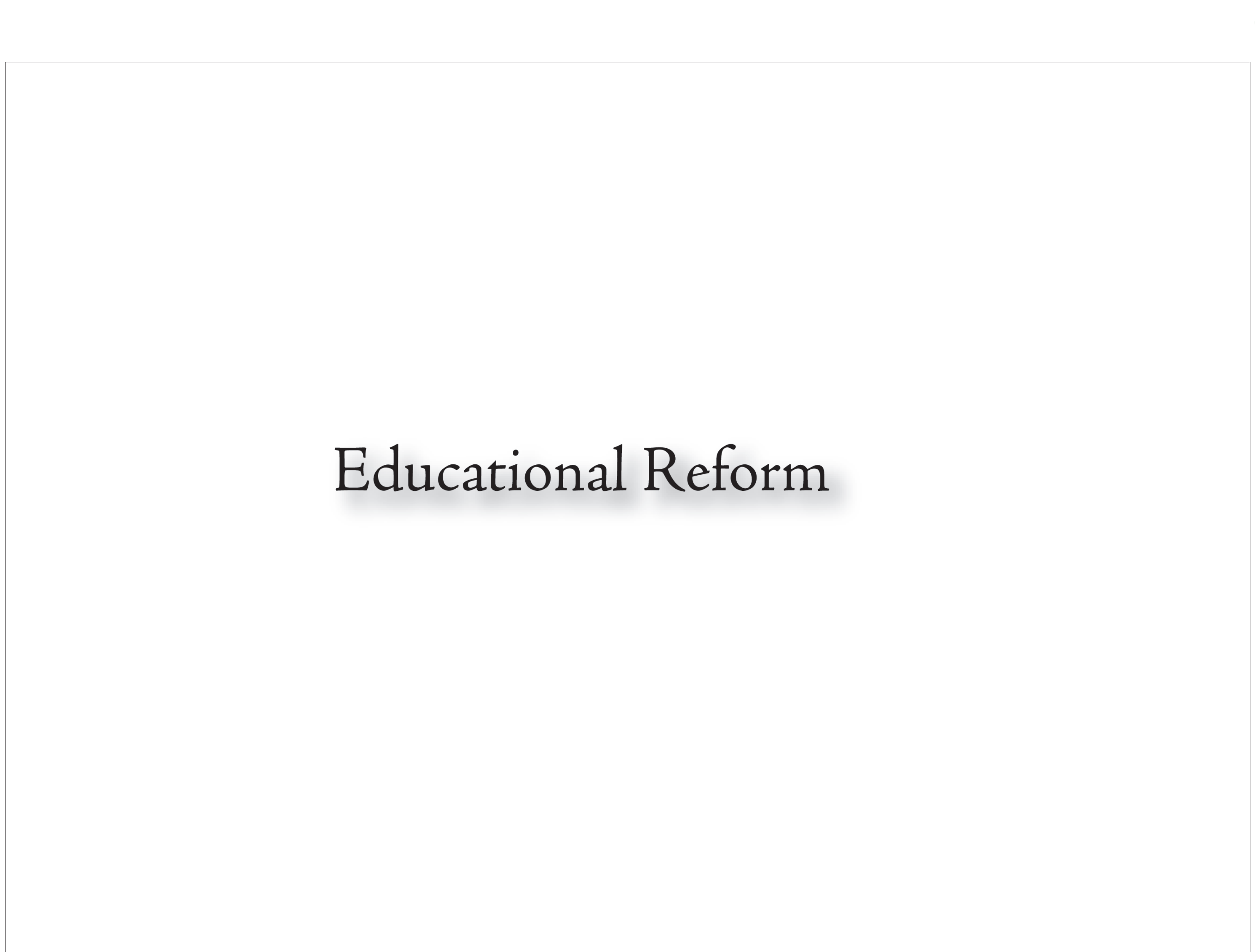
Security Failures

Security Expert$_1$: *Your typical buffer overrun costs a vendor on the order of $2,000. A system immune to buffer overruns would save the vendor about $20,000 a year. The cost to make a system immune to buffer overruns, using a technology such as Immunex™ would greatly exceed that.*

Security Expert$_2$: *Why not just funnel all array writes through a routine that checks the bounds? If there are a few for which this results in too great an impact on performance, those can be analyzed individually.*

Security Expert$_1$: *This is non-trivial to do in C (because you rely on the programmer correctly determining the bounds for the funneled writes **and** you have to modify large amounts of libc).*




Educational Reform

Learning to Write

The MFA process:

- A series of mentors who are writers
- Revision under direction of the mentors
- Learning a process of writing
- Craft-oriented, critical writing on work by other writers
- Learning what and how to "steal"
- Learning how to write by writing while reflecting on it

Then:

- practice like hell
- read like hell
- writers' workshops
- revise like hell

Creative Writing and Software

Working Writer	Working Developer	Working Team
practice & reflection	?	read a book??
education		
practice		
individual	individual	individual

Master of Software Engineering vs MFA

	CMU MSE	Warren Wilson MFA
Semesters	3	5
Classes	10 courses	75 classes/seminars
Systems/Manuscripts	1	1
Programs/Pieces	?	85
Workshops	0	5 (150 hours)
Essays on Craft	?	64
Dissertation	0	1
Judged System/Manuscript	1	1
Classes Taught	0	1
Systems Presented	0	1
Mentored Projects	1	5
Books Read	?	112

Redefining Computing

Redefining Computing

While it is perhaps natural and inevitable that languages like Fortran and its successors should have developed out of the concept of the von Neumann computer as they did, the fact that such languages have dominated our thinking for twenty years is unfortunate. It is unfortunate because their long-standing familiarity will make it hard for us to understand and adopt new programming styles which one day will offer far greater intellectual and computational power.

—John Backus, 1981

Programming Languages

Millions for compilers but hardly a penny for understanding human programming language use. Now, programming languages are obviously symmetrical, the computer on one side, the programmer on the other. In an appropriate science of computer languages, one would expect that half the effort would be on the computer side, understanding how to translate the languages into executable form, and half on the human side, understanding how to design languages that are easy or productive to use.... The human and computer parts of programming languages have developed in radical asymmetry.

—Alan Newell & Stu Card, 1985

Computing Paradigms

...the current paradigm is so thoroughly established that the only way to change is to start over again.

—Donald Norman, The Invisible Computer

Deep Trouble

Computer Science is in deep trouble. Structured design is a failure. Systems, as currently engineered, are brittle and fragile. They cannot be easily adapted to new situations. Small changes in requirements entail large changes in the structure and configuration. Small errors in the programs that prescribe the behavior of the system can lead to large errors in the desired behavior. Indeed, current computational systems are unreasonably dependent on the correctness of the implementation, and they cannot be easily modified to account for errors in the design, errors in the specifications, or the inevitable evolution of the requirements for which the design was commissioned. (Just imagine what happens if you cut a random wire in your computer!) This problem is structural. This is not a complexity problem. It will not be solved by some form of modularity. We need new ideas. We need a new set of engineering principles that can be applied to effectively build flexible, robust, evolvable, and efficient systems.

Gerald Jay Sussman, MIT

Amorphous Computing Project, MIT

A colony of cells cooperates to form a multicellular organism under the direction of a genetic program shared by the members of the colony. A swarm of bees cooperates to construct a hive. Humans group together to build towns, cities, and nations. These examples raise fundamental questions for the organization of computing systems:

- *How do we obtain coherent behavior from the cooperation of large numbers of unreliable parts that are interconnected in unknown, irregular, and time-varying ways?*
- *What are the methods for instructing myriads of programmable entities to cooperate to achieve particular goals?*

These questions have been recognized as fundamental for generations. Now is an opportune time to tackle the engineering of emergent order: to identify the engineering principles and languages that can be used to observe, control, organize, and exploit the behavior of programmable multitudes.

Amorphous Computing Project

The objective of this research is to create the system-architectural, algorithmic, and technological foundations for exploiting programmable materials. These are materials that incorporate vast numbers of programmable elements that react to each other and to their environment. Such materials can be fabricated economically, provided that the computing elements are amassed in bulk without arranging for precision interconnect and testing. In order to exploit programmable materials we must identify engineering principles for organizing and instructing myriad programmable entities to cooperate to achieve pre-established goals, even though the individual entities are unreliable and interconnected in unknown, irregular, and time-varying ways.

Autonomic Computing Project, IBM

Civilization advances by extending the number of important operations which we can perform without thinking about them.

–Alfred North Whitehead

. . . we need to create more complex systems. How will this possibly help? By embedding the complexity in the system infrastructure itself—both hardware and software —then automating its management. For this approach we find inspiration in the massively complex systems of the human body. Think for a moment about one such system at work in our bodies, one so seamlessly embedded we barely notice it: the autonomic nervous system.

It tells your heart how fast to beat, checks your blood's sugar and oxygen levels, and controls your pupils so the right amount of light reaches your eyes as you read these words. It monitors your temperature and adjusts your blood flow and skin functions to keep it at 98.6° F. It controls the digestion of your food and your reaction to stress—it can even make your hair stand on end if you're sufficiently frightened. It carries out these functions across a wide range of external conditions, always maintaining a steady internal state called homeostasis while readying your body for the task at hand.

Autonomic Computing Project

But most significantly, it does all this without any conscious recognition or effort on your part. This allows you to think about what you want to do, and not how you'll do it: you can make a mad dash for the train without having to calculate how much faster to breathe and pump your heart, or if you'll need that little dose of adrenaline to make it through the doors before they close.

It's as if the autonomic nervous system says to you, Don't think about it—no need to. I've got it all covered. That's precisely how we need to build computing systems—an approach we propose as autonomic computing.

Paul Feyerabend

...one of the most striking features of recent discussions in the history and philosophy of science is the realization that events and developments ... occurred only because some thinkers either decided not to be bound by certain 'obvious' methodological rules, or because they unwittingly broke them.

This liberal practice, I repeat, is not just a fact of the history of science. It is both reasonable and absolutely necessary for the growth of knowledge. More specifically, one can show the following: given any rule, however 'fundamental' or 'necessary' for science, there are always circumstances when it is advisable not only to ignore the rule, but to adopt its opposite.

—Paul Feyerabend, Against Method

Feyerabend Project

Computing theory was first developed as a branch of computability theory based on a particularly inexpressive set of mathematical constructs.

The proofs of their universality under certain assumptions and equivalences to other mathematical constructs under others have irreparably boxed us into a realm of almost blinding inexpressiveness.

Feyerabend Project

Early computing practices evolved under the assumption that the only uses for computers were military, scientific, and engineering computation—along with a small need for building tools to support such activities.

We can characterize these computational loads as numeric, and the resulting computational requirements as Fortran-like. Early attempts to expand the realm of computing—such as John McCarthy's valiant attempt with Lisp and artificial intelligence, Kristen Nygaard's similarly pioneering attempt with Simula, Doug Englebart's intriguing attempt to expand our usage options, Alan Kay's later attempt with Smalltalk, and Alain Colmerauer's attempt with Prolog—were rebuked.

Feyerabend Project

The architecture of almost every computer today is designed to optimize the performance of Fortran programs and its operating-system-level sister, C.

Further, attempts to consider neuron-based, genetic, or other types of nonstandard computational models were soundly rejected, snubbed, and even ridiculed until the late 1980s.

Feyerabend Project

Programming languages have hardly shown one scintilla of difference from the designs made in the first few years of computing.

All significant programming languages are expressively, conceptually, and aesthetically equivalent to Fortran and assembly language—the direct descendants of Turing computability based on Von Neumann machines. Programming today is the intellectual equivalent of picking cotton, with the same overtones of class structure. For example, it is currently not permitted for a developer to refuse to implement a faulty or unsafe program unless he or she is willing to be fired. Lawyers are provided more avenues for ethical and moral behavior than programmers.

Feyerabend Project

Software development methodologies evolved under a mythical belief in master planning.

Such beliefs were rooted either in an elementary-school-level fiction that all great masterpieces in every discipline were planned, or as a by-product of physicists shovelling menial and rote coding tasks to their inferiors in the computing department. Christopher Alexander has run into the same set of beliefs in the building trades, and this uniquely Cartesian view of the world is currently setting back progress at an alarming pace.

Feyerabend Project

High-octane capitalism has created a nightmare scenario in which it is literally impossible to teach and develop extraordinary software designers, architects, and builders.

The effect of ownership imperatives has caused there to be absolutely no body of software as literature. It is as if all writers had their own private "companies" and only people in the Melville company could read **Moby-Dick** and those in Hemingway's could read **The Sun Also Rises**. Can you imagine developing a rich literature under these circumstances? There could be neither a curriculum in literature nor a way of teaching writing under such conditions. And we expect people to learn to program in exactly this context?

Feyerabend Project

Distributed systems—the gold standard of computing today—requires that vastly diverse, dispersed, and different-minded people—and massive numbers of them—contribute to each large software system.

Each such system never goes down, it can't be recompiled from scratch, it can never be in total version coherence, it must have parts that depend on everything that has ever been developed. Even in a monolithic system we find the same thing. For every major operating system, there are people who absolutely demand and rationally require the newest backwardly incompatible features, and others who absolutely and rationally require the oldest features to remain as they are forever. Every single programming language we have is predicated on the physicists' model of figure it out, code it up, compile it, run it, throw it away. All our computing education is based on this, all our methodologies, all our languages, environments, tools, attitudes, mathematics, prejudices, and principles are based as solidly and firmly on this rock as Gibraltar is based on the stone it rests on. Many if not most development headaches come from precisely this problem, one that programming language theorists ridicule. Every attempt to define a language to handle this or a computing device to make it simpler is met with laughter and rejection. And all this in the face of the fact that the reality of programming must be based on precisely the opposite assumptions: Everything changes, every version is necessary, evolution happens.

Feyerabend Project

Computing practice and theory is based on the very hidden and hard-to-reveal assumption that engineers, mathematicians, and computer scientists are the only ones who will write a program or contribute to a software system.

When computing started, no one but scientists and the military even remotely conceived of the utility of programs and programming. Today, almost every business and human pursuit is built on computing and digital technology. Artists, craftspeople, writers, fishermen, farmers, tightrope walkers, bankers, children, carpenters, singers, dentists, and even some animals depend on computing, and most of the people I mentioned want to have a say in how such software works, looks, and behaves. Many of them would program if it were possible. The current situation might feel fine to some of you, but suppose all computing were based on the needs of tightrope walkers? Hard to imagine. What we've created is hard to imagine for them.

Feyerabend Project

Computing is based on utility, performance, efficiency, and cleverness.

Where are beauty, compassion, humanity, morality, the human spirit, and creativity?

Feyerabend Project

- Understand the limitations of our current computing paradigm
- Understand the limitations of our current development methodologies
- Bring users—that is, people—into the design process
- Make programming easier by making computers do more of the work
- Use deconstruction to uncover marginalized issues and concepts
- Looking to other metaphors
- Three workshops so far, four more planned—using a tipping-point approach

Feyerabend Project

- Homeostasis, immune systems, self-repair, and other biological framings
- Physical-world-like constraints—laws, contiguity
- Blackboards, Linda, and rule-systems—use compute-power
- Additive systems—functionality by accretion not by modification
- Non-linear system-definition entry—instead of linear text
- Non-mathematical programming languages
- Sharing customizations
- Language co-mingling and sustained interaction instead of one-shot procedure invocation in the form of questions/answers or commands
- Piecemeal growth, version skews, random failures
- Artists' understanding, ambiguous truth

Biological Framings of Problems in Computing

- Goal is to come up with "Hilbert Problems" for computing
- Need for new metaphors both for computing and for biology

> *Every living organism is the outward physical manifestation of internally coded, inheritable, information.*
>
> –http://www.brooklyn.cuny.edu/bc/ahp/BioInfo/GP/Definition.html

Biological Characteristics

- Robustness
  - Error correction
  - Failure tolerance
  - Massive redundancy
  - Scalability
  - Stability
  - Tolerance of uncertainty
  - Self-sustaining
  - Regeneration/turnover
- Dynamics
  - Bloom and pruning
  - Far from equilibrium
  - Dynamic communities
- Identity
  - Self-containment
  - Ongoing organization—this is what biological systems do
  - Motivation/emotion
  - "Thermodynamically" self-sufficient

Biological Characteristics

- Development
  - Self-development/growing
  - Genotype/phenotype distinction
  - Not constructed - grown instead
- General systems
  - We need to do total system thinking
  - Inter-connectedness

Phenotype: the "outward, physical manifestation" of the organism. These are the physical parts, the sum of the atoms, molecules, macromolecules, cells, structures, metabolism, energy utilization, tissues, organs, reflexes and behaviors; anything that is part of the observable structure, function or behavior of a living organism.

Genotype: the "internally coded, inheritable information" carried by all living organisms. This stored information is used as a "blueprint" or set of instructions for building and maintaining a living creature.

Biological Characteristics

- Evolution
  - Co-evolution
  - Slow/fast(?) to change
  - No planning
  - Creativity
  - Uses randomness
  - Evolvability
  - Actively modify their environment
  - Open-ended growth of complexity
  - Not designed (perhaps "emergent")
  - Fitness function
  - Preservation of legacy
  - Diversity

Biological Characteristics

- Methods and Mechanisms
  - Massive "parallelism," concurrency, non-sequential, inter-influencing, but with no computational overtones or implications
  - Responds to averages
  - Opportunistic
  - Reproduction
  - Mortality
  - Remembering/forgetting
- Embodiment
  - Working complex systems
  - Embodiment
- Adaptation
  - Adaptive
  - Spontaneous
  - Exploitation of abundance
- Communication
  - Coordination
  - Ability to cooperate
  - Use of language

Biological Characteristics

- Structure
  - Continuous/discrete reactiveness
  - Domain stability
  - Levels of aggregation
  - Conservation of mechanism, of successful mechanisms (common model for every cells)

Formalisms of Mutability

Construct a formal language in which it is possible to measure and express the degree and manner of mutability of a system.

Example mechanisms that provide mutability or tinkerability:

- Interpreted
- Strongly-dependent on configuration files/uses macros
  - humanly readable/writeable
  - additive
- Not statically typed
- Late/dynamic bound
- First-class user-created "objects"
  - system/user-supplied indistinguishability
- Doesn't require (global) recompilation, restarting, reloading
- System hacks: hooks, plugins, APIs, layers, callbacks, frameworks, modularity
- Open source
- "undo"

Principles of Good Tinkerability

- Hard to screw up system
- Think locally, act locally
- Personalizable (but still additive)
- No special skills or tools required
- Nothing hidden
- No legal impediments
- No priviledged actor
- Learnable/no fear/local learning
- Shouldn't take a performance hit for tinkering;
  - no performance hit even if they don't use it
- Many system concepts are in turn first-class objects
- Tinkerings are first-class objects and thus can be sent to other users and are additive

Layers of Tinkerability

- Appearance/interfaces
- Preferences/options/setting defaults—parameter tweaking
- Vocabulary of actions or objects
- Meaning of built-in actions or objects
- Meaning of system concepts

Emergence Multipiler

Construct a language, system, and/or environment in which the goals and global or emergent properties of a population may be expressed, and which produces or assists implementing a local behavior or rules that exhibit the specified properties.

- The manner of specifying the global properties may or may not be linguistic.
- The emphasis is on describing or specifying population behavior not the behavior of individuals (though there is the issue of stating population behavior as individual behavior such as "each bird is always flying").
- This problem is intended to spawn research into designing and constructing systems with globally emergent behaviors, such as distributed process control in a chemical processing plant or the swarming behavior of a finite set of repair robots. Human intervention, assistance, or tuning is allowed.

Biological Programming Language

Construct a polysynthetic programming language which provides entity evolution, fuzzy binding, continuous gradient event streams, and massive redundancy with no overhead of user effort.

- Understand the implications of different kinetic models of entity interaction.
- What would the implications be of a model that supported a strict conservation principle, that is, that entities cannot be created or destroyed?

A polysynthetic language is a language in which words tend to consist of several morphemes.

Yup'ik Inuit

tuntussuqatarniksaitengqiggtuq

'He had not yet said again that he was going to hunt reindeer.'

Non-Deductive Programming Model

Construct a non-deductive (locally deductive) programming model and develop a theory of organization (or perhaps computing) for it.

The basic idea is to construct a programming language or programming model in which there can never be certainty (within a program) whether all elements of any "data structure" have been traversed or whether the program has halted or should halt. This is not the halting problem because we are not reasoning about whether a given program will terminate, but whether the very program can decide whether it should terminate. To do this, we recommend eliminating the ability to index data structures and the ability to traverse deterministically any data structure.

This can destroy the ability to reason about programs and, more importantly, for a program to reason about itself. We might want to allow "local" inferences to be made, suitably defined. The idea would be to permit deterministic traversal only in small locales. We could accomplish this by either requiring some sort of very local determinism, or by making indexing and reliable traversal be probablistic with determinism only on small data structures and having the probabilities drop as structures increase in size.

Given a model or language like this, it is not clear what can be programmed, but there may be organizations that can emerge because local inference works while larger inference radii don't (to use a geometric metaphor). So, we may not be able to program computations but affinities and local organizations.

Back to Immunology

Design a processor (and instruction format) that continuously establishes that the instructions that it executes are from a given administrative domain (self versus non-self). Find a way to distribute programs within the domain that preserves its unique sense of self.

Farewell

Few people who are not actually practitioners of a mature science realize how much mop-up work ... a paradigm leaves to be done or quite how fascinating such work can prove in the execution.... Mopping-up operations are what engage most scientists throughout their careers. They constitute what I am here calling normal science. Closely examined, whether historically or in the contemporary laboratory, that enterprise seems an attempt to force nature into the preformed and relatively inflexible box that the paradigm supplies. No part of the aim of normal science is to call forth new sorts of phenomena; indeed, those that will not fit the box are often not seen at all. Nor do scientists normally aim to invent new theories, and they are often intolerant of those invented by others....

Perhaps these are defects. The areas investigated by normal science are, of course, miniscule....

—Thomas S. Kuhn

Confusionists and superficial intellectuals move ahead *while the 'deep thinkers'* descend *into the darker regions of the status quo or, to express it in a different way, they remain stuck in the mud.*

—Paul Feyerabend

Where I'm from the birds sing a pretty song

and there's always music in the air

—The Little Man