The Challenge of Pervasive Software to the Conventional Wisdom of Software Engineering

Mary Shaw

Carnegie Mellon University

http://www.cs.cmu.edu/~shaw/

Institute for Software Research

Most software creators are not software professionals.

Ultra-Large-Scale systems are a qualitative shift.

How does current research match these challenges?

What new types of research does this suggest?

Most software creators are not software professionals.

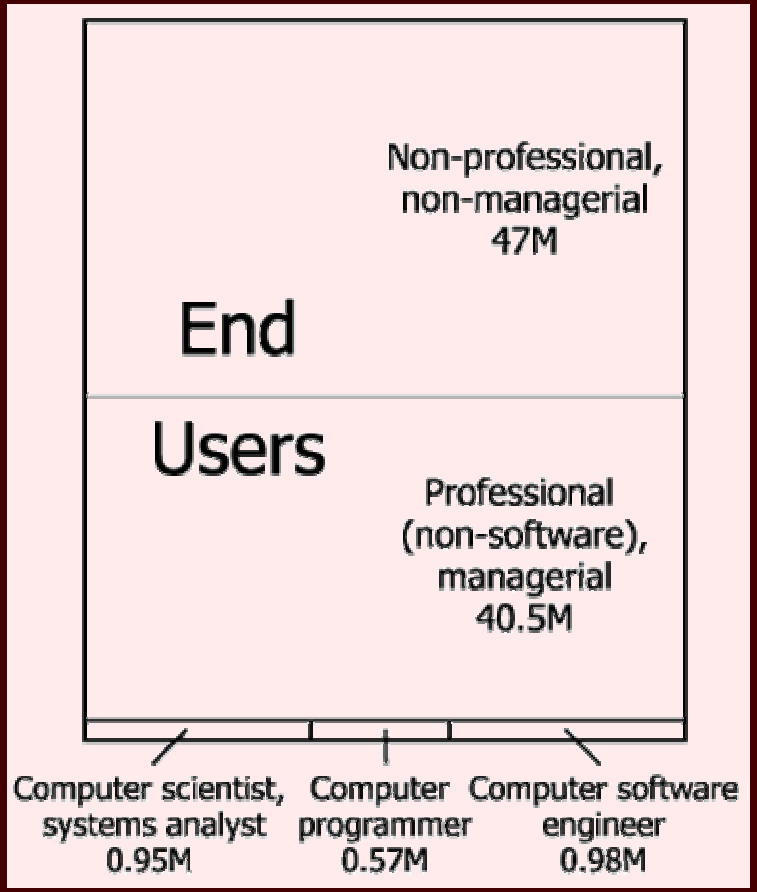
- End users are participants and developers, not passive consumers
- They do not reason about software like professionals

Ultra-Large-Scale systems are a qualitative shift.

How does current research match these challenges?

What new types of research does this suggest?

There are *lots* of end users

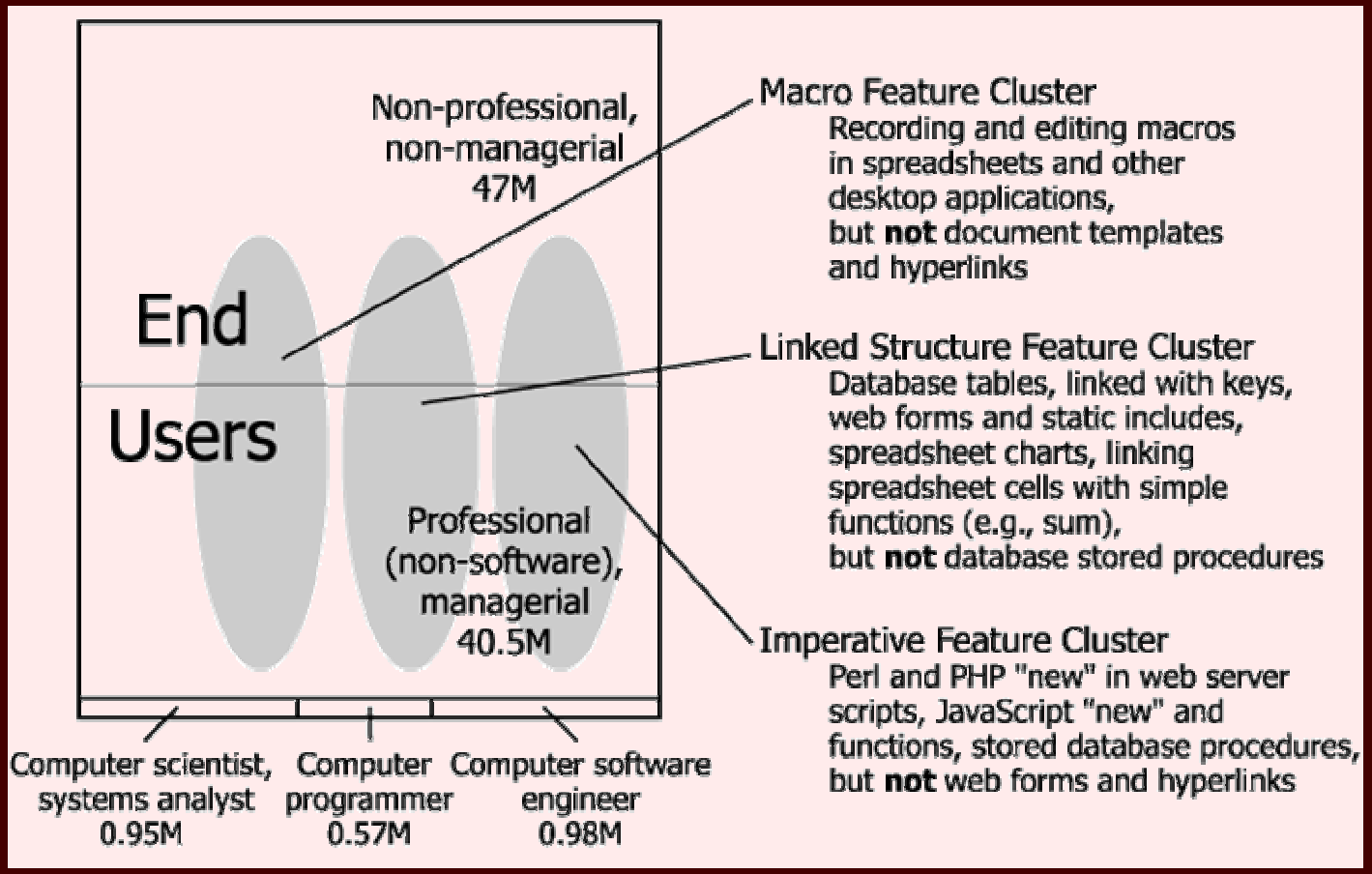


Using data from the Bureau of Labor Statistics, we estimate that over 90M Americans will use computers at work in 2012. Of these, only about 2.5M will be professional programmers; 40.5M will be managers and (non-software) professionals.

This does not include home users or non-US users, so there will be many more than 90M total end users. Most of them will "program" in some way.

C. Scaffidi, M. Shaw, and B. Myers. Estimating the Numbers of End Users and End User Programmers. *VL/HCC'05: Proc 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, pp. 207-214, 2005.

They are not all alike



Analysis of web-based survey of Information Week readers

C. Scaffidi, Andrew Ko, B. Myers, and M. Shaw. Dimensions Characterizing Programming Feature Usage by Information Workers. *VL/HCC'06: Proc2006 IEEE Symposium on Visual Languages and Human-Centric Computing*, pp. 59-62, 2006.

Their skills differ, even within clusters

- "Programming" can mean anything from editing spam filters to writing complex code
  - Copying the html for a web page hit counter *vs* creating that html yourself
- There is not a sharp criterion that says where "programming" starts

Undermines common assumptions:

- Software is mostly created by professionals
- End users just need good user interfaces

Demographics of US Internet users

Overall	Total adults	79%
	Women	77
	Men	81
Age	18-29	92%
	30-49	87
	50-64	79
	65+	42
Geography	urban	75%
	suburban	77
	rural	65
Education	< high school	50%
	high school	69
	some college	88
	college +	94

Pew Internet & American Life Project, 2009 http://www.pewinternet.org/Static-Pages/Trend-Data/Whos-Online.aspx

Internet resources

- *Information:* unstructured text, formatted text, databases, live data feeds, images, maps, current status (e.g., inventory)
- *Calculation:* reusable software components, applications that can be invoked remotely (e.g., services)
- *Communication:* messages, streaming media, synchronous communication, agent systems, alert/notification services
- *Control:* coordination for use of resources, registration and subscription services
- *Services:* simulation, editorial selection, evaluation, secondary (processed) information, responsive experts

Properties of internet resources

- *Autonomous*
  - Independently created and managed
  - May change structure or format without notice
- *Heterogeneous*
  - Different packagings, output for viewing only
  - Different business objectives, conditions of use
- *Open affordances*
  - Independent systems, not dependent components
  - Incidental effects may be useful

Undermines common assumptions:

- It's all about programs
- Someone is "in charge" or "in control"

Adults seek health information online

- American adults include internet among their sources of health information in 2009
  - 86% ask a doctor, 68% ask a friend, 57% ask the internet
  - They access user-generated information as well as professionally-generated information
- Adults with a disability or chronic disease (20% of adults) are less likely to go online (51% vs 74% in 2007)
  - Once disabled people are online, 86% look for health information
- They trust the information
  - 60% say online information affected a decision in 2009
  - Most patients do not check source and date of online health information (14% always, 18% mostly in 2007)

Pew Internet & American Life Project, 2007, 2009

End users are normal people

- End users lack rich and robust mental models of their computing systems
  - they fail to do backups
  - they can not safely configure a network
  - they do not understand storage models
    - especially local vs network storage
- End users put themselves at risk
  - they execute malware and open attachments
  - they do not understand privacy issues
  - they trust information without validating sources
    - and also software downloads
  - they innocently engage in other risky behavior.

End users are not software engineers

- The responses of SE to this mismatch between real computing systems and end users' models has been to seek ways to "fix" the users.
- But there is plenty of evidence that most people do not reason in the linear, rational form that computer scientists prefer.

Undermines common assumptions:

- Users can be trained to act "rationally"
- Usability is "screen deep"
- Validation is based on a few definitive analyses

Most software creators are not software professionals.

Ultra-Large-Scale systems are a qualitative shift.

- They are large along many dimensions
- More important, they are more complex and more open-ended than normal systems
- Scale and complexity make them qualitatively different

How does current research match these challenges?

What new types of research does this suggest?

Ultra-Large-Scale Systems

- Large size on many dimensions
  - Lines of code, amount of data, users, dependencies among and complexity of components, etc
- More than "systems of systems"
- Characteristics
  - Decentralized operation and control
  - Conflicting, unknowable, diverse requirements
  - Continuous evolution and deployment
  - Heterogeneous, inconsistent, changing elements
  - Indistinct people/system boundary
  - Normal failures
  - New forms of acquisition and policy

Decentralized operation and control

- ULS system scale offers only limited possibilities for central or hierarchical control
  - Long life, multiple users and objectives, span of physical jurisdictions are the norm at ULS scale
  - Many versions of subsystems must work together
  - Modifications are developed and installed by independent groups
  - Spontaneous, unanticipated new uses arise

Undermines common assumption:

- Specifications are complete, static, & homogeneous
- All conflicts must be resolved and must be resolved uniformly

Conflicting, unknowable, diverse requirements

- A ULS systems serves a wide range of competing stakeholders
  - Competing user groups contend for requirements
  - Understanding of problem evolves
  - Problem is deeply embedded in cultural context
  - Dependability is "better/worse", not "right/wrong"

Undermines common assumptions:

- Requirements are known in advance, evolve slowly
- Tradeoff decisions will be stable

Continuous evolution and deployment

- ULS systems have long lives and multiple independent developers
  - Different groups may install capability for their own needs; this may conflict with other groups
  - Evolution can't be controlled centrally, must be shaped by rules and policies that protect critical services and allow diversity at the edges

Undermines common assumption:

- System improvements are introduced in "releases"
- Users/developers know about releases and can choose to accept them or not

Heterogeneous, inconsistent, changing elements

- ULS systems will be composed from independently-created components
  - *Heterogeneous:* many sources, no single interface standard, often incorporating legacy systems
  - *Inconsistent:* evolution spontaneous, not planned; different objectives may cause inconsistent versions
  - *Changing:* hardware, software, operating environment change based on local decisions

Undermines common assumptions:

- Effect of change can be predicted adequately
- Configuration information is accurate & controlled
- Components and users are fairly homogeneous

Indistinct people/system boundary

- ULS systems' service to a user depends on actions of other users; user/developer distinction soft
  - User actions may affect overall system health
  - System must adapt to changing usage patterns
  - Aggregate analysis may be better than exact analysis

Undermines common assumption:

- Users' behavior doesn't affect overall system
- Collective behavior of people is not relevant
- Social interactions are not relevant

Analogy: Cities and city planning

- Cities are complex systems
  - Built of individual components chosen by individuals
  - Constantly evolve
  - Withstand failures and attacks
- Cities are not centrally controlled
  - Standards for infrastructures
    - Building codes, highway standards
  - Policies that allow individual action within constraints
    - Zoning laws
  - Regulations that govern individual action
    - Enforcement after the fact, rather than prior constraint
- *"Wicked problems"*

Normal failures

- ULS system scale implies inevitable failures, so systems must do protection/recovery/enforcement
  - Hardware failures are inevitable because of scale
  - Legitimate use of software and services outside planned capability will cause degradation/failure
  - Malicious use will cause problems

Undermines common assumptions:

- Failures will be infrequent and exceptional
- Defects can be removed

New forms of acquisition and policy

- ULS systems will evolve, but there must be governance to prevent anarchy
  - Success of system depends on organic evolution
  - Individual developers won't fully understand core infrastructure
  - Need effective guidance on allowed/unallowed change

Undermines common assumption:

- There is a single agent responsible for system development, operation, and evolution

Most software creators are not software professionals.

Ultra-Large-Scale systems are a qualitative shift.

How does current research match these challenges?

- What research paradigms are currently used?
- What are the right questions about current problems?
- What are appropriate methods to investigate those questions?

What new types of research does this suggest?

Research Styles

- Physics and medicine have well-recognized research styles
  - Hypothesis, controlled experiment, analysis, refutation
  - Double-blind large-scale studies
- Acceptance of results relies on process as well as analysis
- Simplified versions help to explain the discipline to observers

- Disciplines can be characterized by identifying what they value:
  - What kinds of questions are "interesting"?
  - What kinds of results help to answer these questions?
    - What research methods can produce these results?
  - What kind of evidence demonstrates the validity of a result?
- So, then, what does software engineering value?

ESEC/FSE 09 research topics

- Composing/integrating systems and services [9]
- Testing/debugging code [7]
- Analyzing/verifying code [6]

- Testing/monitoring composed systems [4]
- Analyzing composed systems [3]
- Developing code [3]
- Predictions about process [2]
- Generalizations [2]

- Total of 36 research and short papers

Building Blocks for SE Research

Question	Strategy/Result	Validation
Devlpmt method	Proc/technique	Analysis
Analysis method	Qual/desc model	Experience
Evaluate instance	Analytic model	Example
Generalization	Empirical model	Evaluation
Feasibility	Tool/notation	Persuasion
	Specific solution	
	Report	

Analysis technique, analytic validation

Question	Strategy/Result	Validation
Devlpmt method	*Cost est method*	*Statistical comparison*
Can we predict cost?	Qual/desc model	Experience
	Analytic model	
Evaluate instance	Empirical model	Example
Generalization	Tool/notation	Evaluation
	Specific solution	
Feasibility	Report	Persuasion

ESEC/FSE 09

Question	Strategy/Result	Validation
Devlpmt method	Proc/technique	Analysis
Analysis method	Qual/desc model	Experience
Evaluate instance	Analytic model	Example
Generalization	Empirical model	Evaluation
Feasibility	Tool/notation	Persuasion
	Specific solution	
	Report	

Typical ESEC/FSE 09 paper

Question	Strategy/Result	Validation
Devlpmt method	Prog/[illegible]nique	Analysis
Analysis [illegible]ethod	Qual/desc m[illegible]	Experience
Evaluate instance	Analytic model	Example
Generalization	Empirical model	Evaluation
Feasibility	Tool/notation	Persuasion
	Specific solution	
	Report	

Typical ESEC/FSE 09 paper

Contrast with EASE04

- Typical software engineering papers take the form
  - I set out to do X
  - I produced result Y
  - I validated the result by Z
- Current ESEC/FSE papers commonly take the form
  - I set out to analyze X
  - I created a technique Y and sometimes a tool T
  - I validated the result by Z
- However, a typical EASE04 paper takes the form
  - I wanted to understand P
  - I designed experiment Q
  - I learned R, with confidence S and caveats T

Typical EASE04 paper (about half)

Question	Strategy/Result	Validation
Devlpmt method	Proc/technique	Analysis
Analysis method	Qual/desc model	Experience
Evaluate feasibility	Analytic model	Example
Generalization	Empirical model	Evaluation
Feasibility	Tool/notation	Persuasion
	Specification/solution	
	Report	

ESEC/FSE 09 papers

- A strong majority were about analyzing some property of a piece of software
- The vast majority devised a technique for doing something
  - Often this included an implementation
- Mostly, they validated the usefulness of the technique with quantitative experiments or case studies at practical scale
  - Abstracts were often unclear about the rigor of the evaluation
  - More than 25% of abstracts did not explain the validation

Most software creators are not software professionals.

Ultra-Large-Scale systems are a qualitative shift.

How does current research match these challenges?

What new types of research does this suggest?

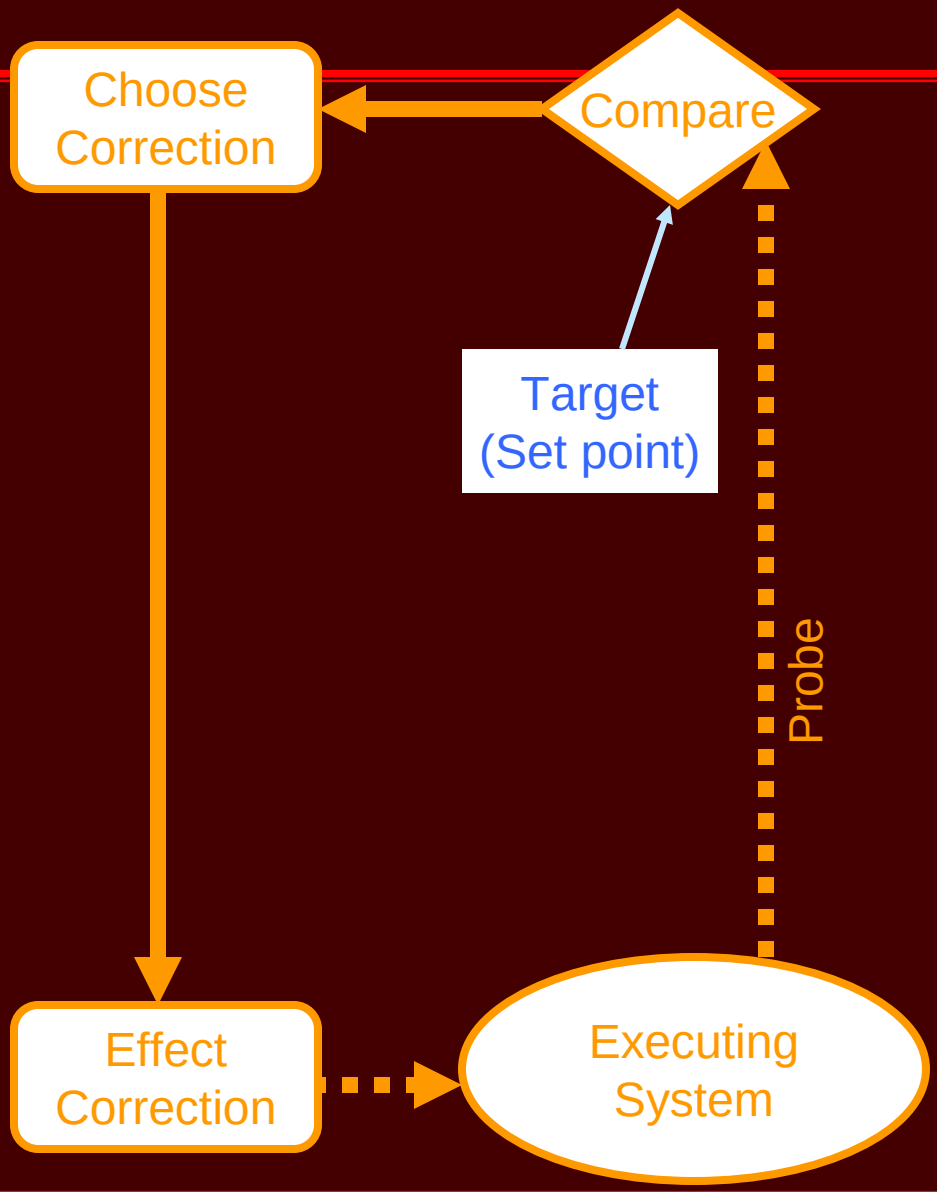
- Put adaptation via feedback on a more systematic basis
- Develop usable techniques to integrate web resources
- Extend architectural models to explain cyberspace
- Reconsider "design"

Adaptive Solutions

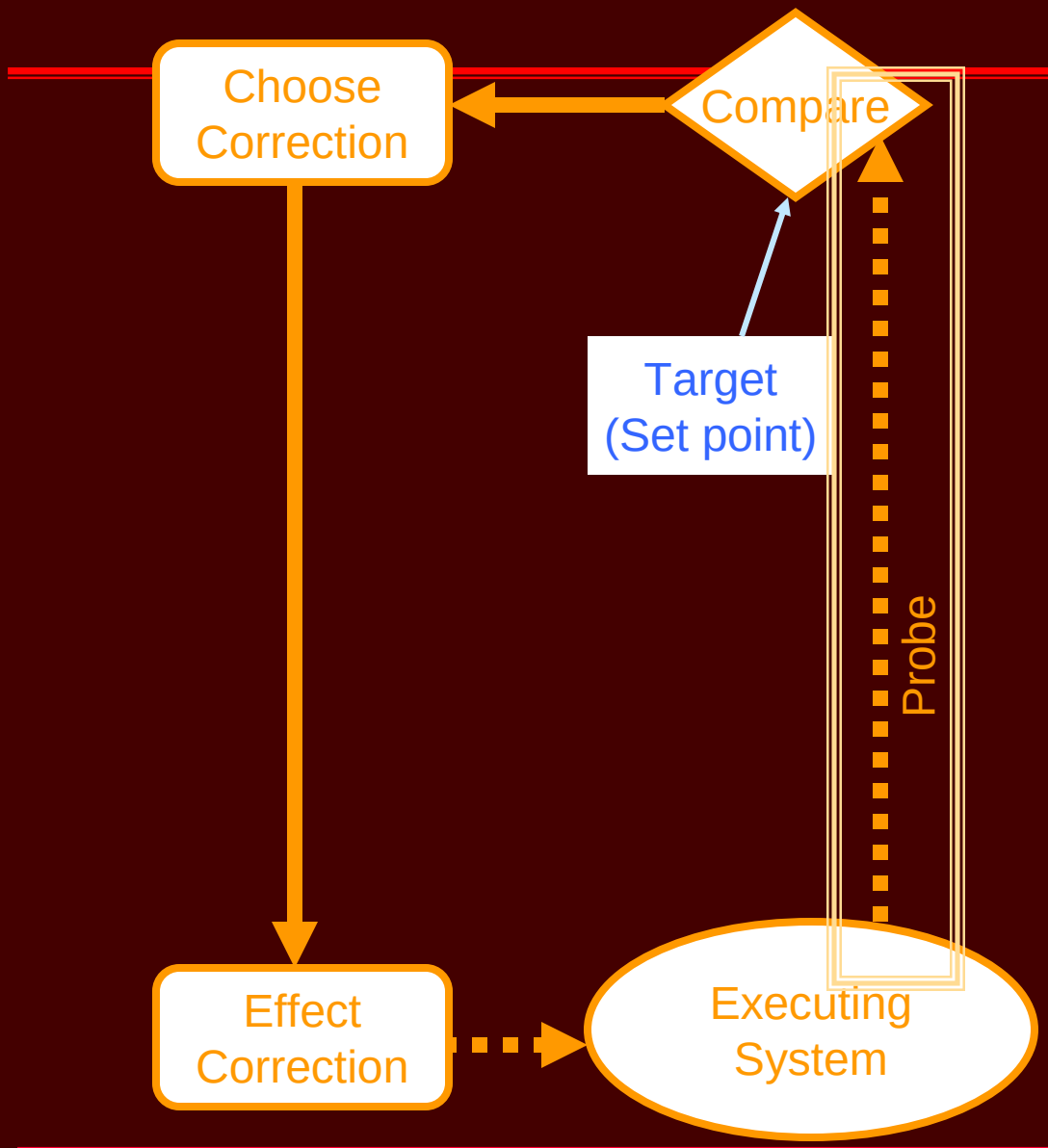
These characteristics of ULSs favor adaptive solutions

- Substantial uncertainty in the environment
  - that leads to substantial irregularity or other disruption
  - that may arise from insufficient knowledge
  - that may arise from rapid change of conditions
- Nondeterminism in the environment
  - of a sort that requires significantly different responses
- Incomplete control of system components
  - for example, mechanical components
  - for example, humans in the loop (they're only biddable)

Feedback Control Loops

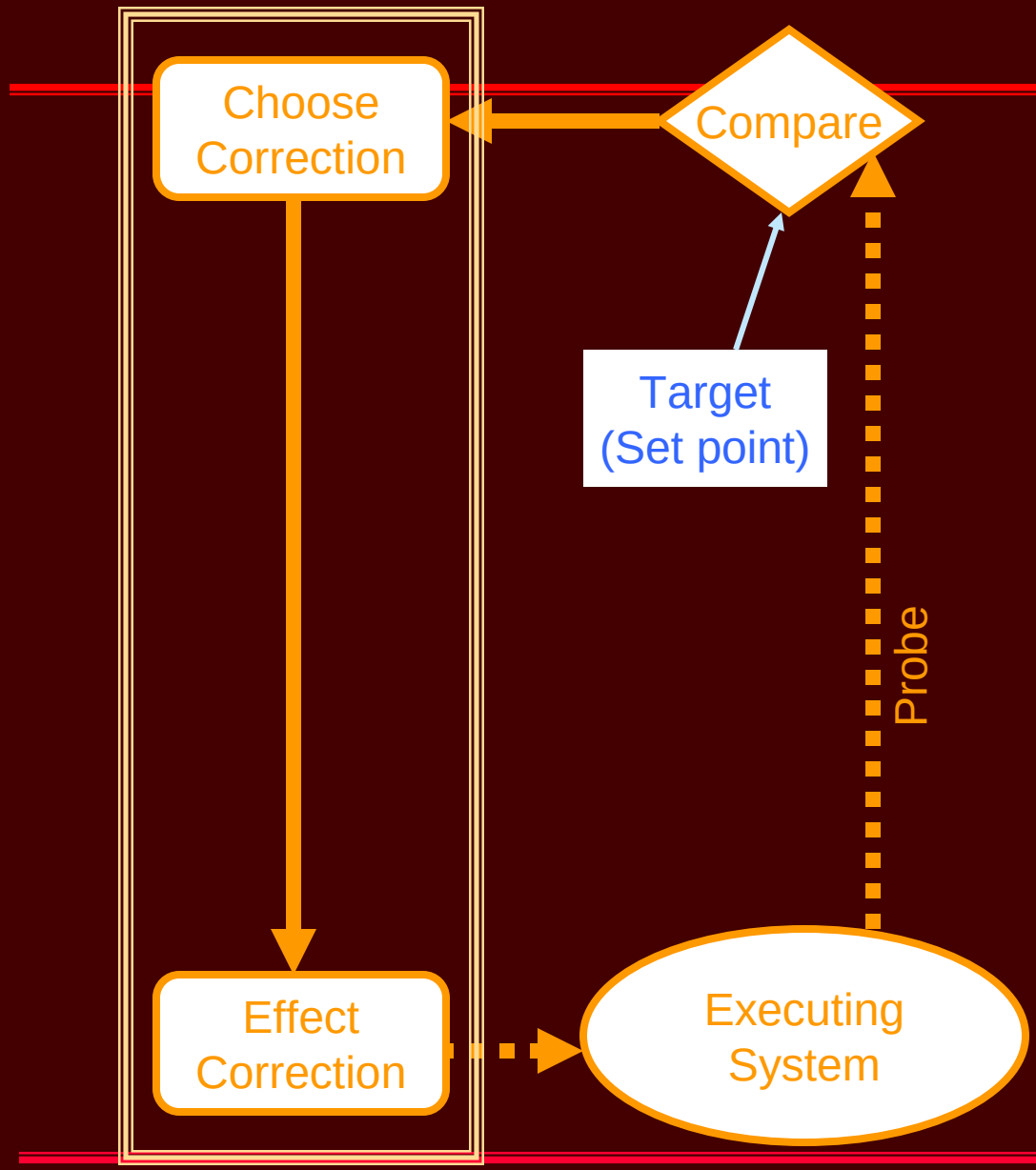


Adaptation can take many forms, but it often involves one or more feedback control loops



Probe must yield model of operating state.

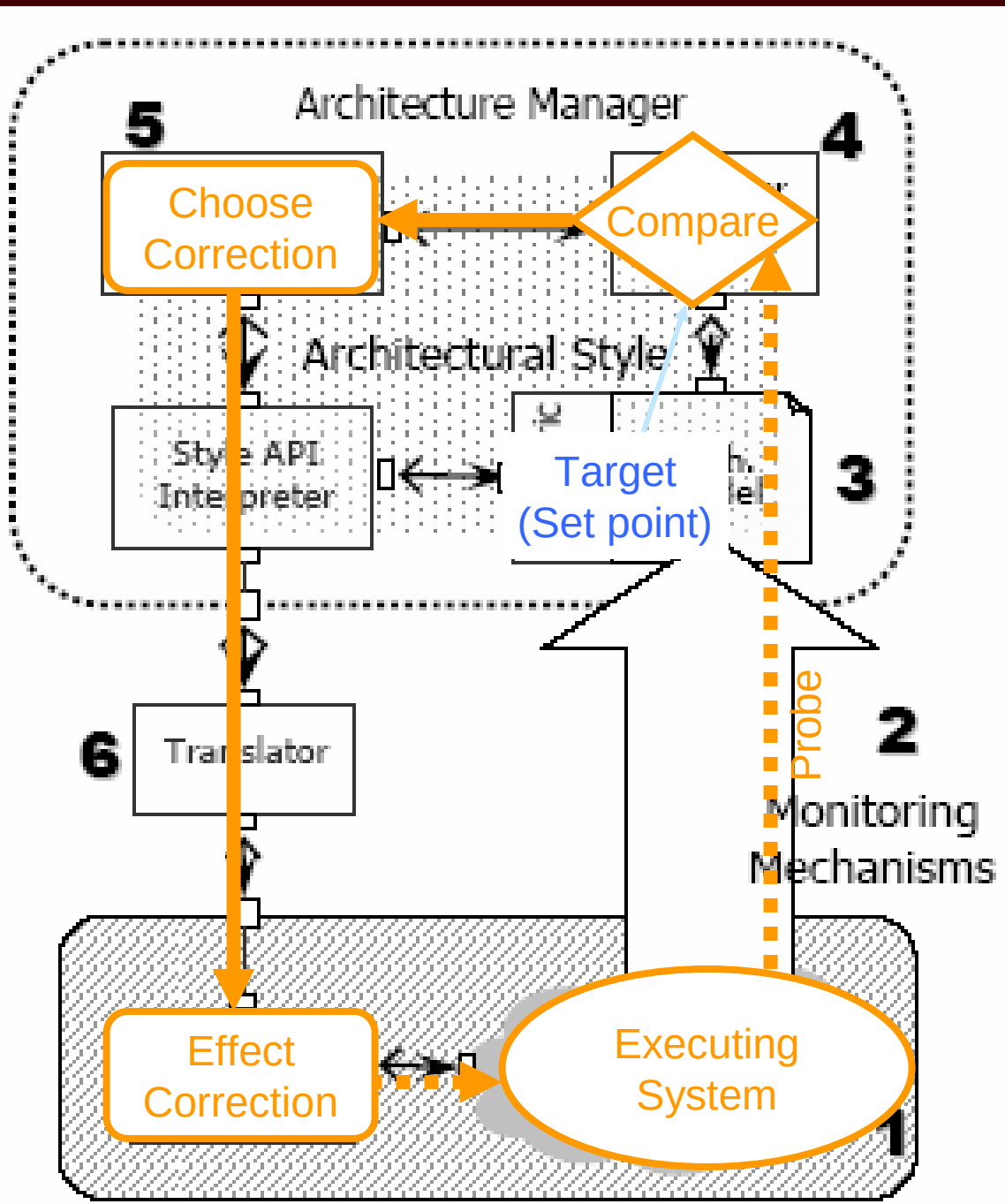
However, it often relies on proxy data of variable quality.



Controller must have sufficient command authority to actually control executing system

Obligations for a feedback loop

- In Requirements
  - Specify the objective, with tolerances and timing
- In Design
  - Identify feedback elements explicitly, in a separate view
  - Choose adaptation (control) strategy
- In Analysis/V&V, show how ..
  - current state is modeled from sensor data
  - the correction plan works (it can be achieved with commands available, it will have desired effect)
  - time constraints are achieved
- In Implementation
  - Map elements of design to elements of implementation (control is not necessarily a separate element)



Adaptable system

Garlan, Cheng, Schmerl

Increasing System Dependability Through Architecture-Based Self Repair

In *Architecting Dependable Systems*
de Lemos, Gacek, Romanovsky (eds)
Springer Verlag 2003

Proposition

- For a system to be called "adaptive", it must change in response to changes in its internal or external environment.
- This usually requires a closed feedback loop.
- ***The control loop should be an explicit, visible element of the system, in design, analysis, and implementation.***
  - *But usually it is not !*

User composition of internet resources

- The internet provides a rich set of resources
  - autonomous, heterogeneous, open affordances
  - meager specification and documentation
- End users should be able to compose resources
  - all resources, not just code
  - current state of art is ad hoc "mash-up"
  - need usable means to combine elements from diverse sources under local control
- End users should be able to understand whether a composition is good enough for their needs
  - Information about resources is low-ceremony and incremental

Example: Pat and Lou

Objective: compose autonomous heterogeneous resources

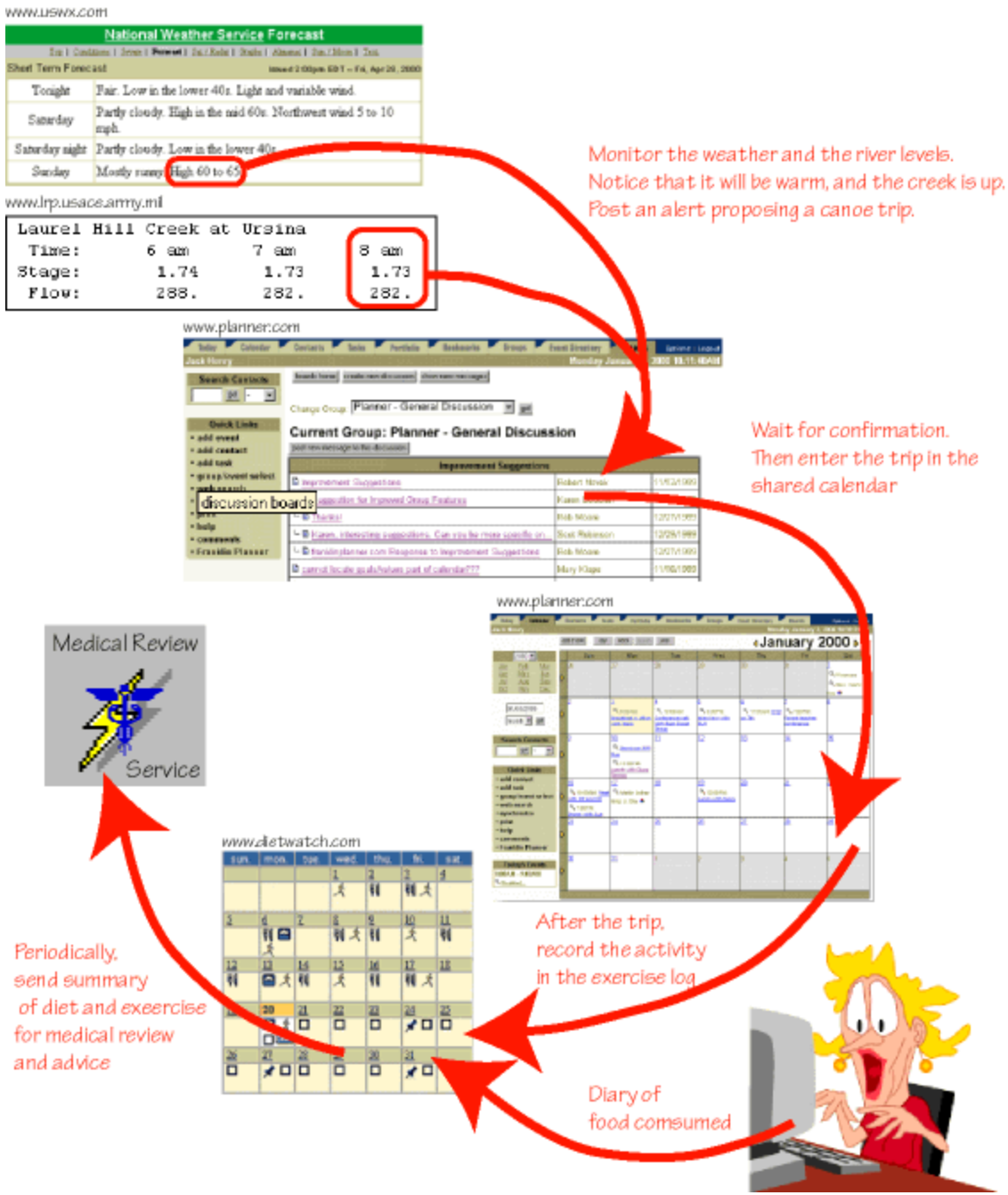
Pat is diabetic

Pat and Lou exercise together outdoors

They plan outings based on online forecasts

Pat logs food and exercise

A medical service reviews logs and offers advice



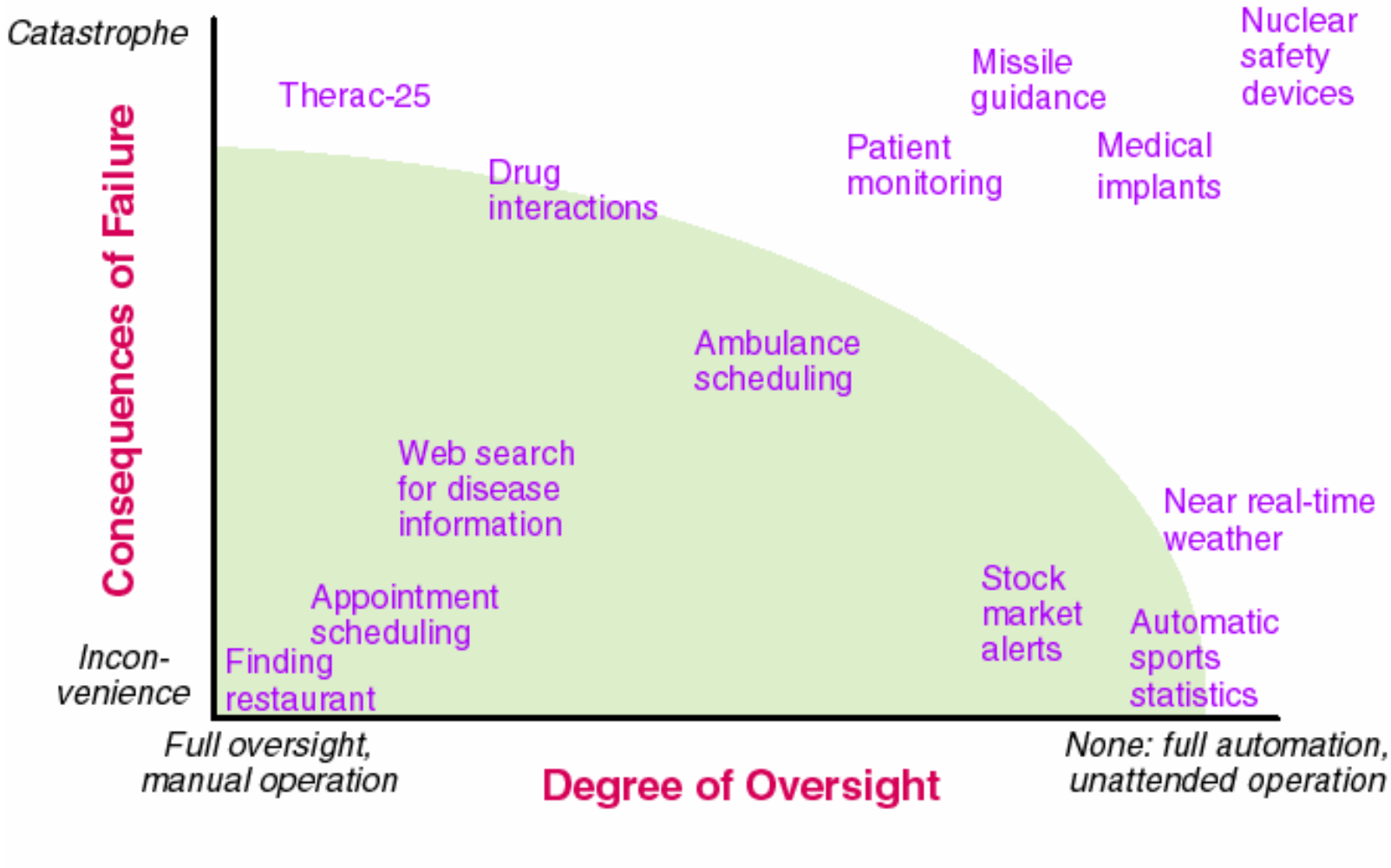
High ceremony evidence

- Widely accepted among computer scientists
- Potentially high levels of assurance
- Need precise specifications, substantial effort
- The Academic Big Four – the "gold standard"
  - Formal verification
  - Results from trusted automatic generator
  - Systematic testing
  - Empirical studies in operation
- And also
  - Inspections
  - Assurance cases, other sound certification

Low ceremony evidence

- Widely available information
  - used informally and incrementally
- Largely ignored by professionals
- Not suitable for high assurance, but inexpensive
- Examples
  - "best X" reports (linear functions of subjective marks)
  - editorial reviews, user reports, advice from friends
  - advertising claims by vendors, branding, seller reputation
  - auction and betting mechanisms, "wisdom of crowds",
  - checklists

Situations suitable for informal reasoning



Yes, also needs axis for rate of exposure

Architectures of cyberspace

- Users can choose from many applications
  - For communication alone, we have email, blogs, facebook, newsgroups, BitTorrent, distribution lists, twitter, Yahoo groups, Wikis, Second Life, Flickr, YouTube, personal webs, …
- Users need help matching application to need
  - Recall, their mental models are weak
- End-user developers often get this wrong
  - For example, they choose a web page update (which requires users to pull the information) rather than email (which pushes the information to the user)
- Comparison through selected architectural properties brings out significant differences

Example architectural comparison

Appli-cation	*Activ-ation*	*Content*	*State*	*Author-ship*	*Inter-activity*	*Synch-ronicity*
email	sender push	text + attach	per user	originator	K known	minutes
news-groups	reader pull	text	server archive	submitter	N anony-mous	hours
chat	inter-active	short text	none	all present	K known	immed-iate
web	reader pull	web page + files	web server	web owner	no	immed-iate
wiki	reader pull	struct text	wiki server	anyone	no	immed-iate
blog	reader pull	text + photo	blog server	originator + readers	N anony-mous	hours to days
2nd life	inter-active	image +	2ndlife server	all members	high	immed-iate
twitter	s-push + r-pull	struc 140-char text	sender tw-page	originator	N knowable	soon

Architectural operators

- RSS feeds
  - Abstractly, convert "reader pull" to "sender push"
  - Can be applied to many sorts of "reader pull"
- Yahoo pipes
  - Merge and filter RSS feeds
  - Do not get much attention from software engineers
  - *Recall that software engineers neglected spreadsheets*

"Design"

- Software engineering separates requirements elicitation from "design"
- This falls squarely in the Simon tradition:
  - [devising] "courses of action aimed at changing existing situations into preferred ones"
  - problem solving
  - satisficing – finding a "good enough" solution
- But both disintermedition of the internet and ultra-large-scale systems lack explicit requirements
- Other design traditions consider problem-setting an integral part of design
- What should software engineering do about this?

- I suggested examples of research that respond to current engineering problems of software
  - Put adaptation via feedback on a more systematic basis
  - Develop usable techniques to integrate web resources
  - Extend architectural models to explain cyberspace
  - Reconsider "design"
- Think about how well current software engineering research paradigms suit our current problems
  - If an a priori specification is not feasible, what does that mean for formal methods? for design?
  - If end users will create widely-used software, how can they gain confidence in it?

Most software creators are not software professionals.

Ultra-Large-Scale systems are a qualitative shift.

How does current research match these challenges?

What new types of research does this suggest?