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

Mary Shaw
Carnegie Mellon University
http://www.cs.cmu.edu/~shaw/

Institute for Software Research

"Computer science is the study of the phenomena surrounding computers"

-- Perlis, Newell, Simon

Most software creators are not software professionals.

Ultra-Large-Scale systems represent a qualitative shift.

What new types of research does this suggest?

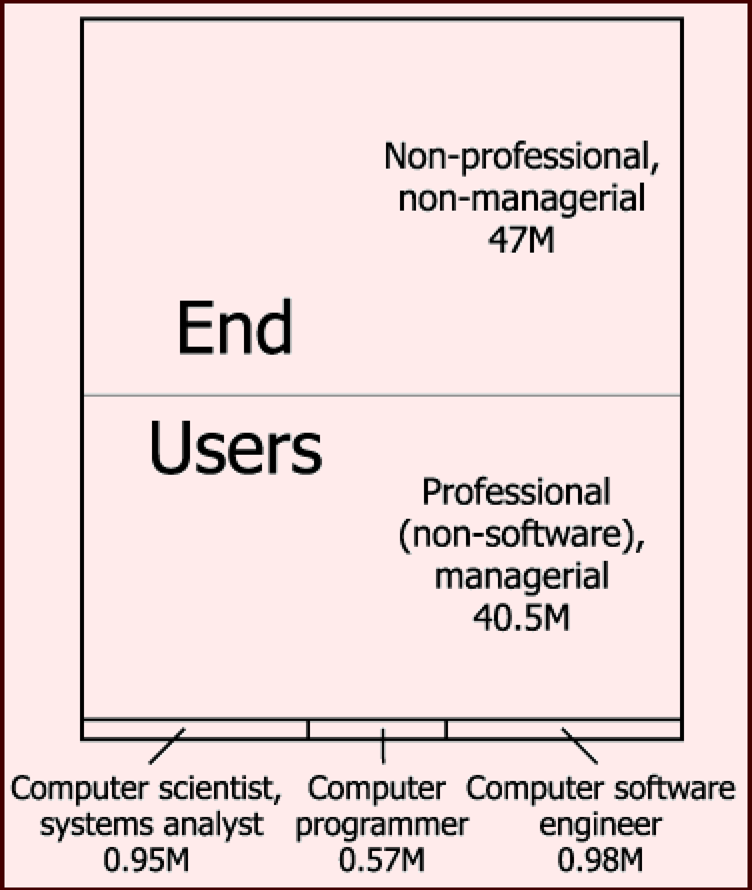
Most software creators are not software professionals.

- End users are participants and developers, not passive consumers
- They do not reason about software like professionals
- "Software" is much more than just code

Ultra-Large-Scale systems represent a qualitative shift.

What new types of research does this suggest?

There are *lots* of end users



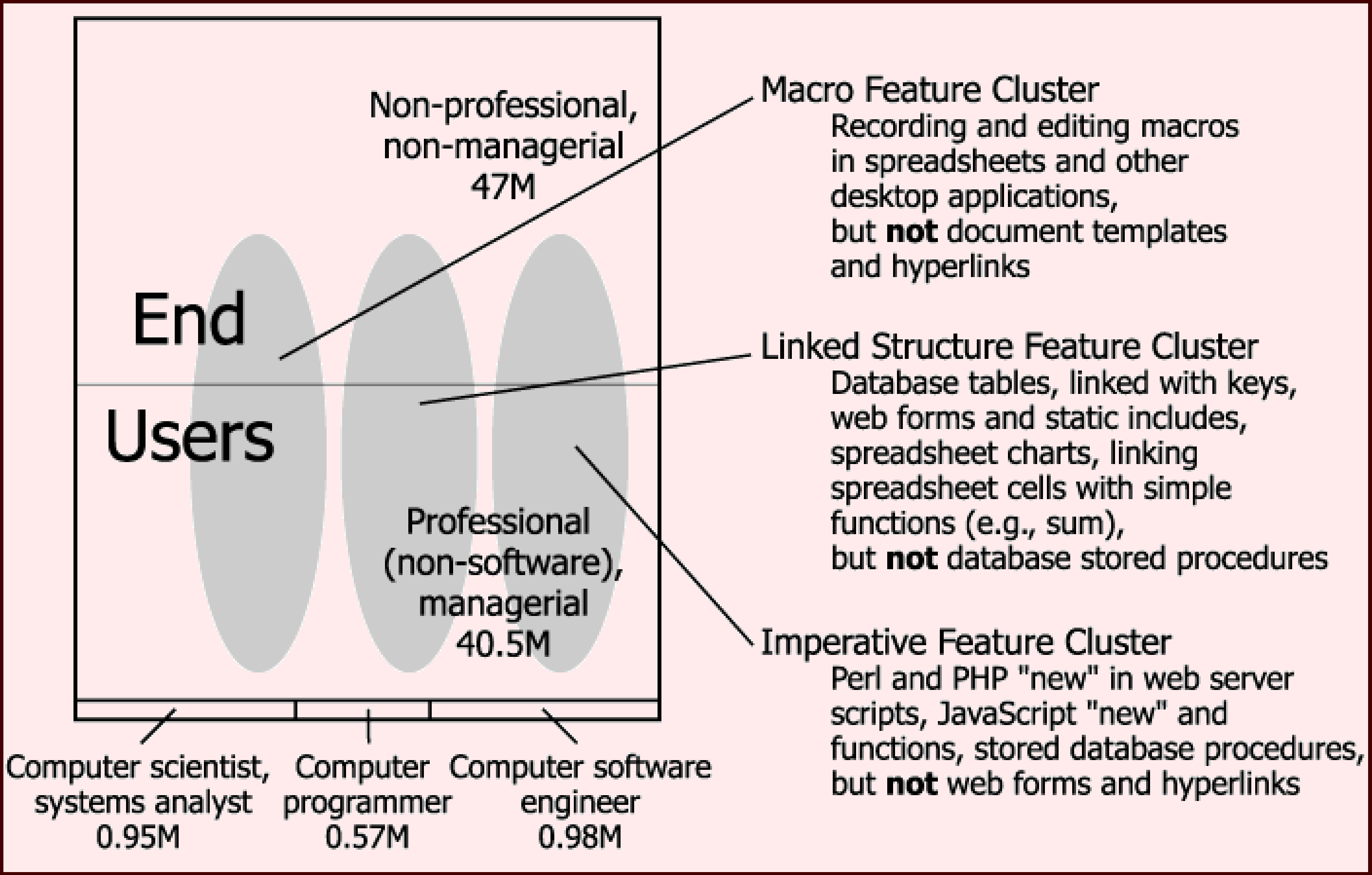
Non-professional, non-managerial 47M
End Users
Professional (non-software), managerial 40.5M
Computer scientist, systems analyst 0.95M
Computer programmer 0.57M
Computer software engineer 0.98M

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


Non-professional, non-managerial 47M
End Users
Professional (non-software), managerial 40.5M
Computer scientist, systems analyst 0.95M
Computer programmer 0.57M
Computer software engineer 0.98M
Macro Feature Cluster
Recording and editing macros in spreadsheets and other desktop applications, but **not** document templates and hyperlinks
Linked Structure Feature Cluster
Database tables, linked with keys, web forms and static includes, spreadsheet charts, linking spreadsheet cells with simple functions (e.g., sum), but **not** database stored procedures
Imperative Feature Cluster
Perl and PHP "new" in web server scripts, JavaScript "new" and functions, stored database procedures, but **not** web forms and hyperlinks

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 "only" need good user interfaces

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 the 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

Internet resources

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

Properties of internet resources

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

Undermines common assumptions:

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

Most software creators are not software professionals.

Ultra-Large-Scale systems represent 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

What new types of research does this suggest?

Ultra-Large-Scale Systems

- It's not just about size !!!
- Some societal problems don't have clean specs, well-defined boundaries, objective tests of success
- Characteristics
  - Multiple stakeholders with different objectives
  - No problem statement that satisfies all stakeholders
  - Different success criteria for different stakeholders
  - Evolving understanding of problem as solution grows
  - Irreversible consequences of any attempt to solve
  - Technical elements, but deep community engatement
- In planning/policy world, "wicked problems"

Decentralized 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, developed and installed independently, must work together
  - Minimal central control governs a few critical parts
  - Spontaneous new uses arise ("network effects")

Undermines common assumption:

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

Conflicting, unknowable, diverse requirements

- Requirements may not be adequately understood until the system is in use
  - Competing user groups contend for requirements
  - Understanding of problem evolves
  - Problem is deeply embedded in cultural context
  - Dependability is "better/worse", not "right/wrong"
- Thus the uncertainty about requirements is inherent to the class of problems

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 open completely new opportunities (“emergence”)
    - This may also conflict with other groups
  - Evolution can’t be controlled centrally; it must be shaped by rules and policies that protect critical services and allow diversity at the edges

Undermines common assumption:

- System improvements are introduced explicitly and at discrete intervals

Heterogeneous, inconsistent, changing elements

- ULS systems will be composed from diverse 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
  - User behavior will be part of system capability
  - System will be used in unanticipated ways
  - 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

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
    - Even well-intentioned users make mistakes

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
  - Comprehensive detailed rules are not feasible
  - Need effective guidance on allowed/unallowed activity, but policy rather than prescription
  - Success of system depends on organic evolution
  - Individual developers won't fully understand core infrastructure

Undermines common assumption:

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

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"*

Most software creators are not software professionals.

Ultra-Large-Scale systems represent a qualitative shift.

What new types of research does this suggest?

- Extend architectural models to explain cyberspace
- Put adaptation via feedback on a more systematic basis
- Reconsider "design"

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

Design spaces

- Design space: set of decisions about a design with alternative choices for the decisions
  - Intuitively, discrete Cartesian space with design decisions as dimensions
  - Complete designs correspond to points in the space
- In practice, design spaces are very rich
  - Representations focus on principal dimensions
  - Some choices preclude others, so hierarchy is used to subdue detail
- Long history in CS
  - Computer architecture, user interfaces, distributed sensors, software architecture styles, typefaces

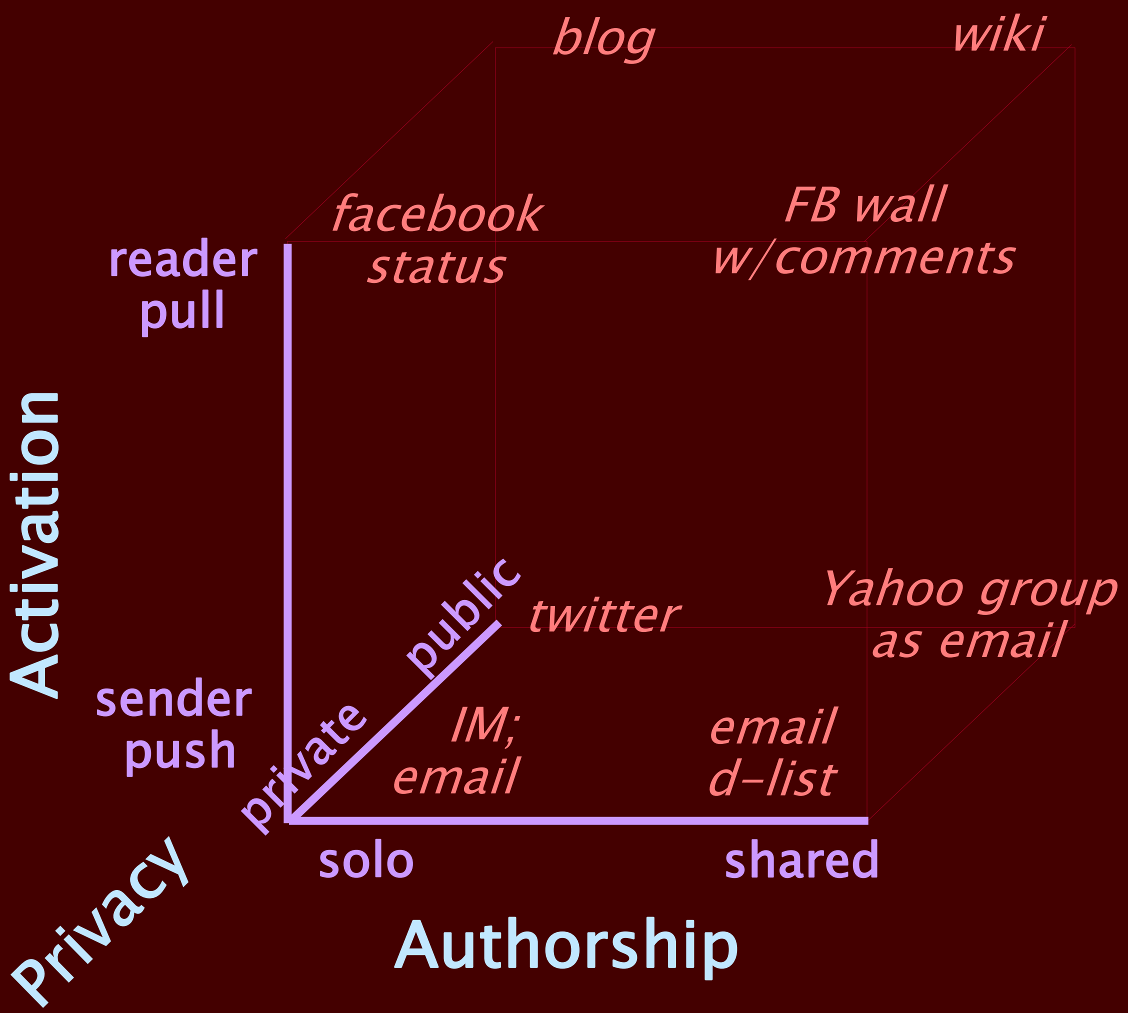
Example architectural comparison

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

Example architectural comparison

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

Design space for web communication



Other Dimensions

- content type
- state location
- interactivity
- synch delay

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*

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

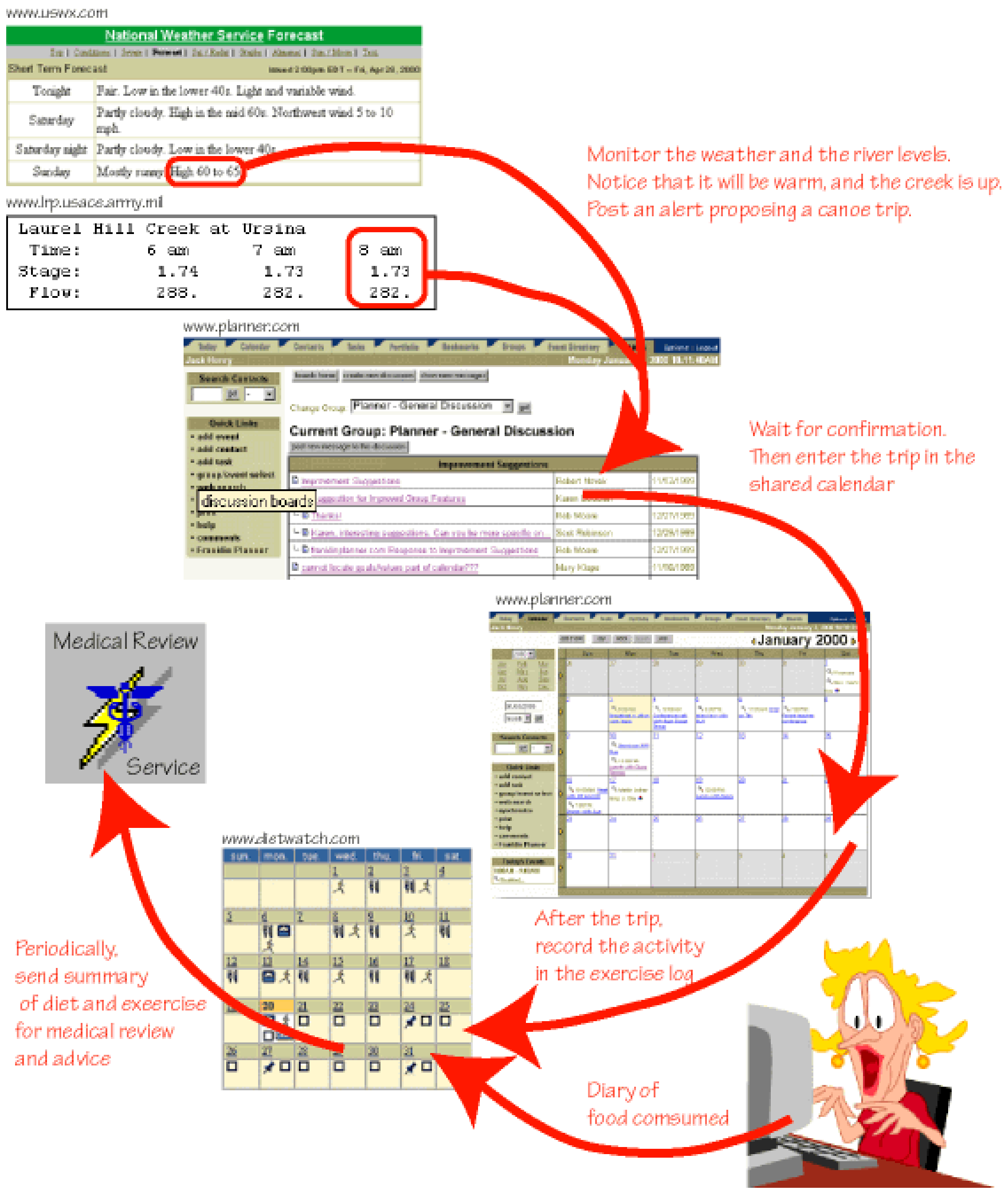
Pat is diabetic

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



www.uswx.com

National Weather Service Forecast

Short Term Forecast

Tonight	Fair. Low in the lower 40s. Light and variable wind.
Saturday	Partly cloudy. High in the mid 60s. Northwest wind 5 to 10 mph.
Saturday night	Partly cloudy. Low in the lower 40s.
Sunday	Mostly sunny. High 60 to 65.

www.lrp.usace.army.mil

```
Laurel Hill Creek at Ursina
 Time:      6 am      7 am       8 am
Stage:      1.74      1.73       1.73
 Flow:      288.      282.       282.
```

www.planner.com

Current Group: Planner - General Discussion

Monitor the weather and the river levels. Notice that it will be warm, and the creek is up. Post an alert proposing a canoe trip.

Wait for confirmation. Then enter the trip in the shared calendar

After the trip, record the activity in the exercise log

Diary of food consumed

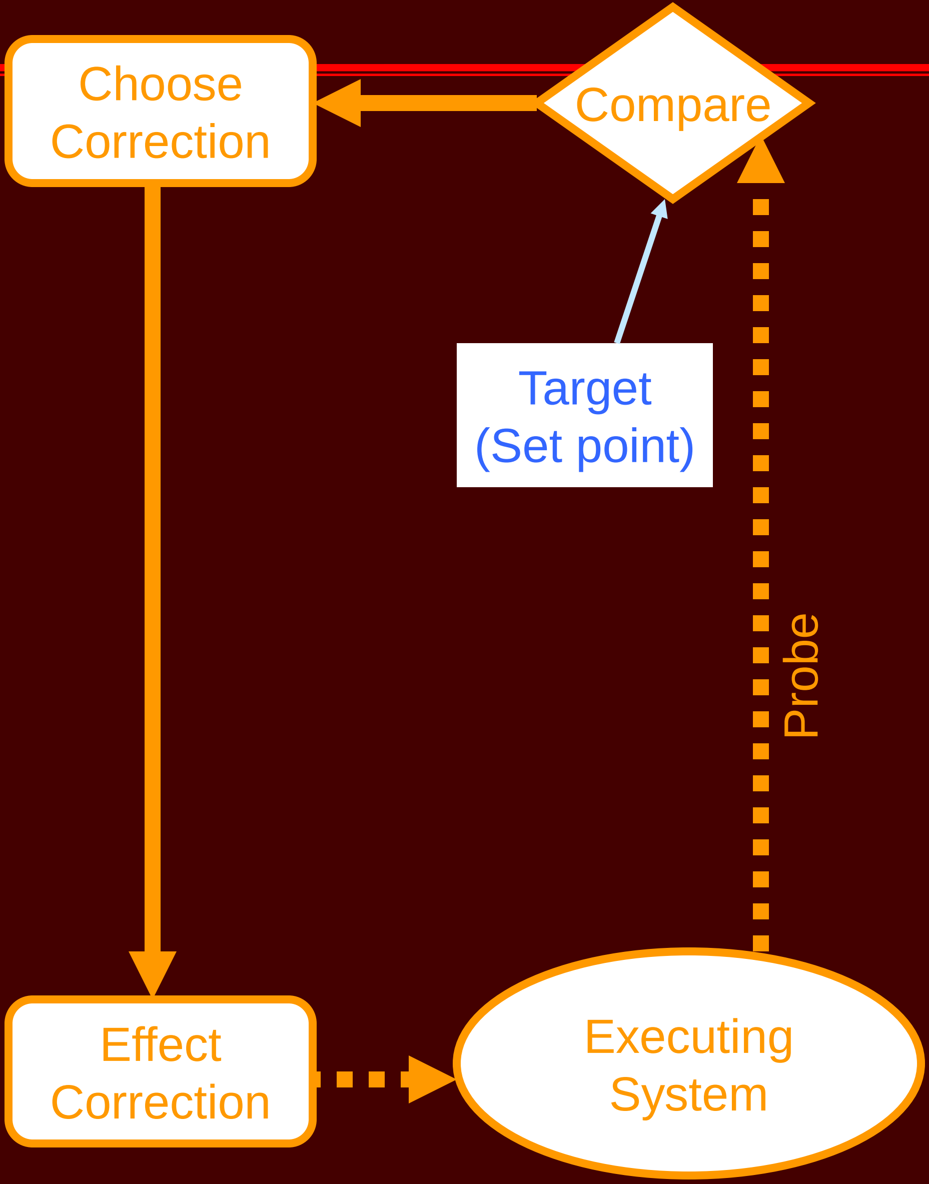
Periodically, send summary of diet and exercise for medical review and advice

Adaptive solutions

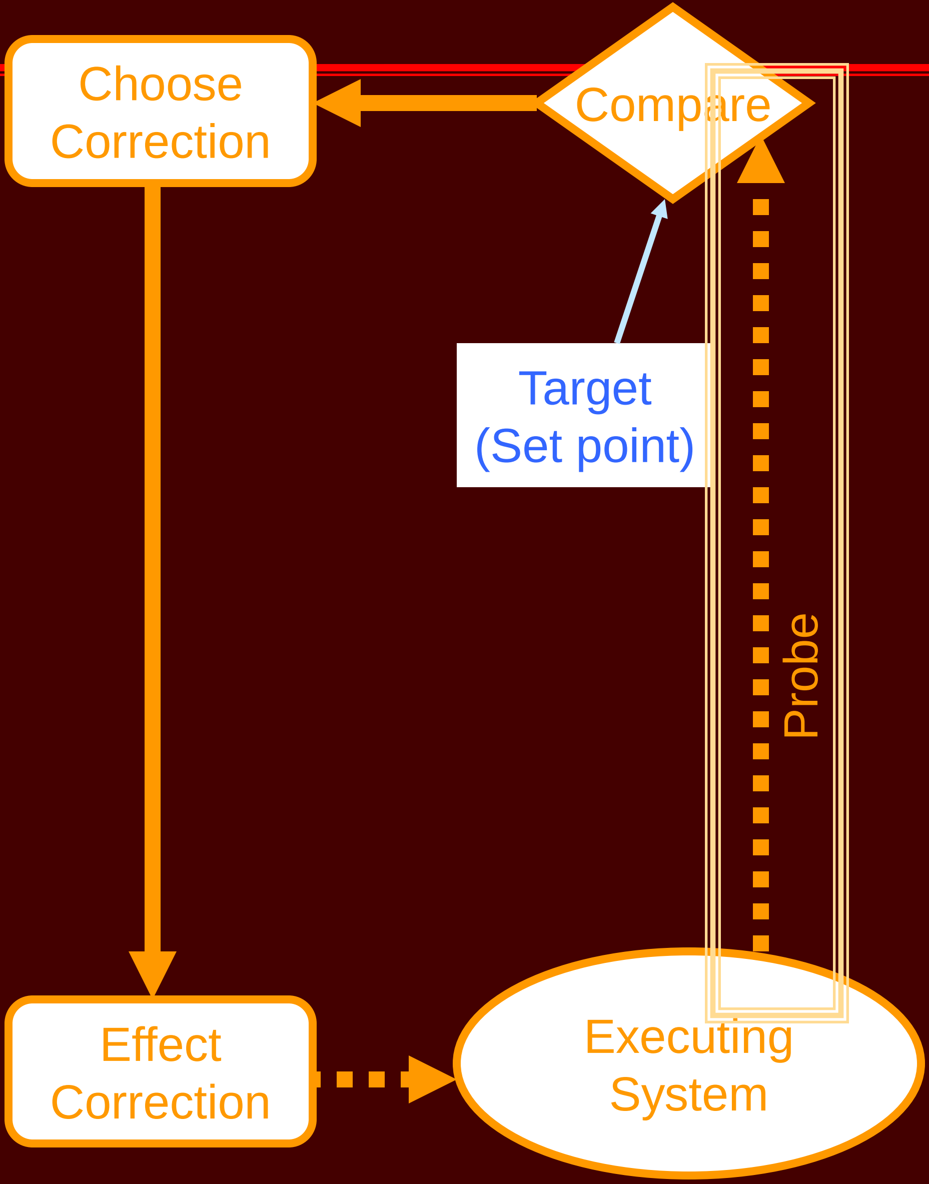
These characteristics of ULSs favor adaptive solutions

- 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

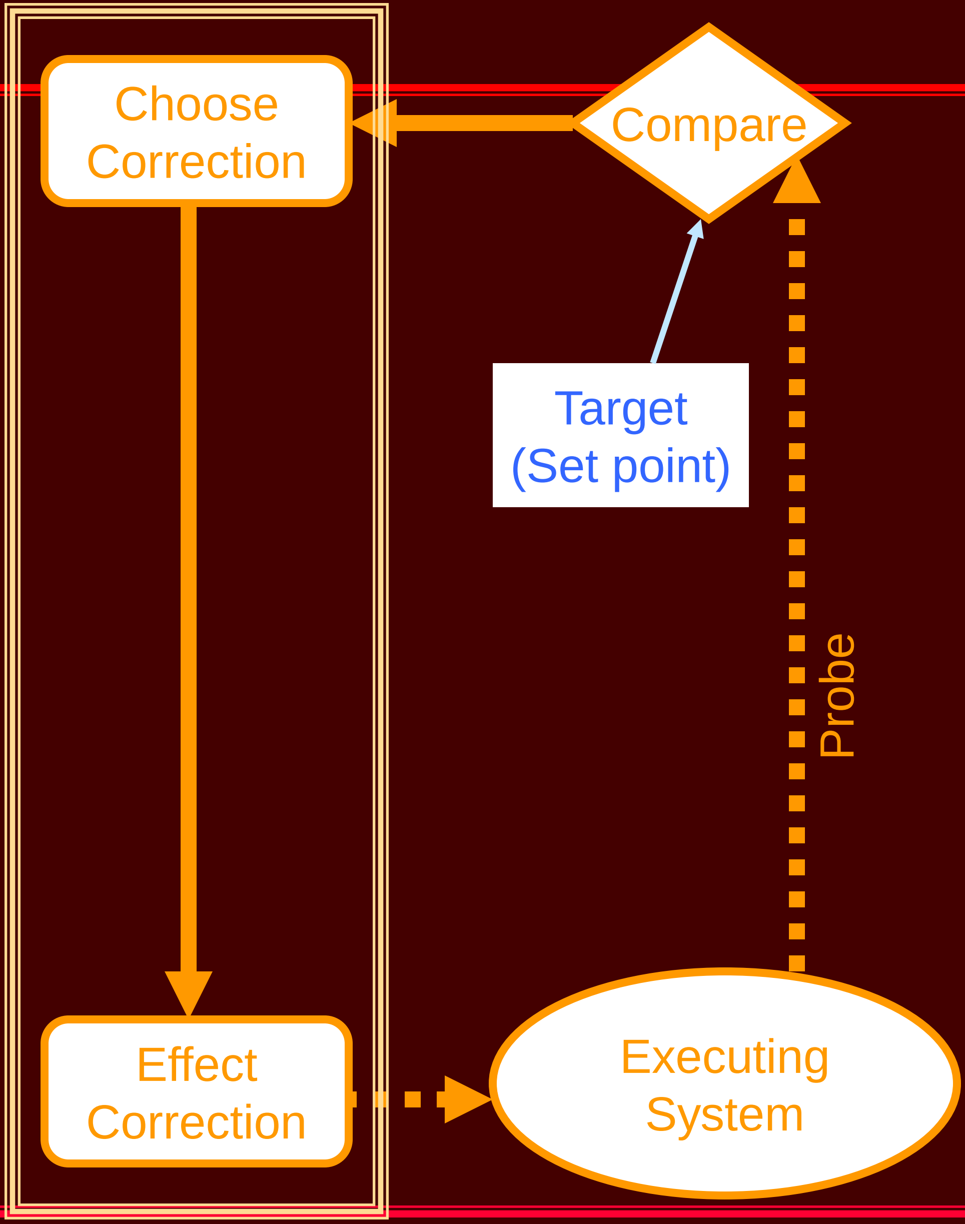


Adaptation can take many forms, but it often involves one or more feedback control loops



Probe must yield model of operating state.

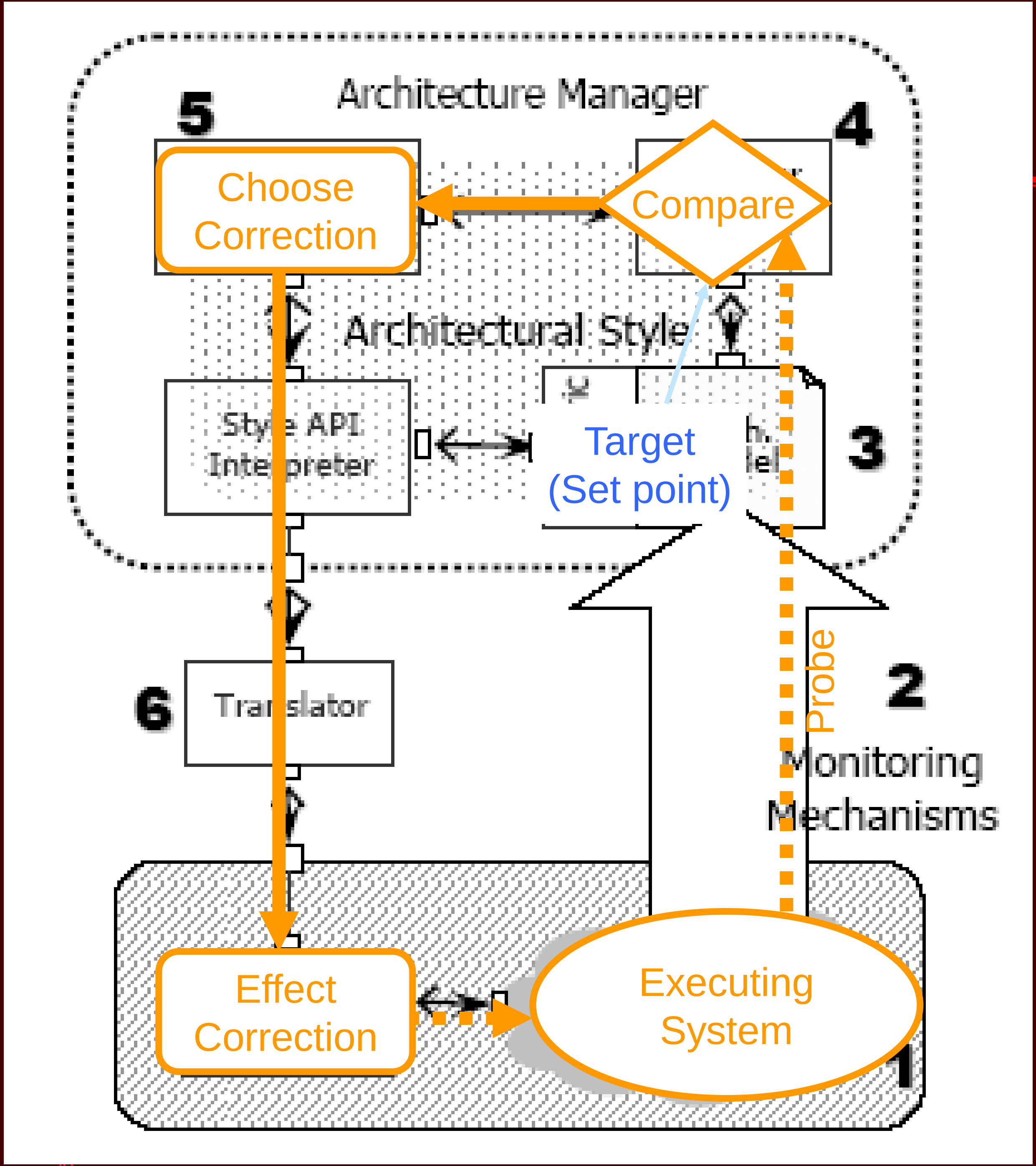
However, it often relies on proxy data of variable quality.



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 !*

“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
  - Extend architectural models to explain cyberspace
  - Put adaptation via feedback on a more systematic basis
  - 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 represent a qualitative shift.

What new types of research does this suggest?