DOI:10.1145/1409360.1409371

Peter J. Denning, Chris Gunderson, and Rick Hayes-Roth

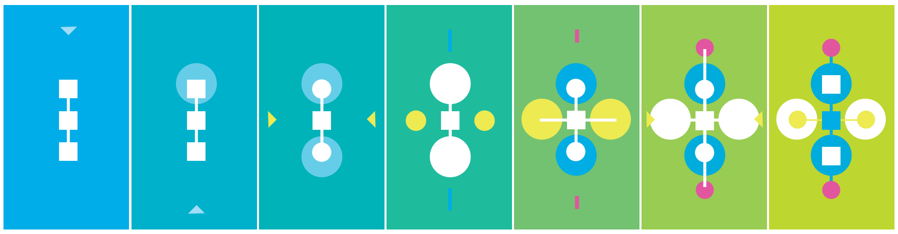
# The Profession of IT
# Evolutionary System Development

*Large systems projects are failing at an alarming rate. It's time to take evolutionary design methods off the shelf.*

Many critical large systems are failing. The replacement FAA air traffic control system, the FBI virtual case file, and the Navy Marine Corps Internet (NMCI), are a few of the many billion-dollar systems that could not deliver the functions needed. In stark contrast, the Boeing 777 aircraft, the Global Positioning System (GPS), and the U.S. Census database system have been outstanding successes. Why do some systems fail and others succeed?

Development time is the critical factor. This is the time to deliver a system that meets the requirements set at the beginning of the development process. If development time is shorter than the environment change time, the delivered system is likely to satisfy its customers. If, however, the development time is long compared to the environment change time, the delivered system becomes obsolete, and perhaps unusable, before it is finished. In government and large organizations, the bureaucratic acquisition process for large systems can often take a decade or more, whereas the using environments often change significantly in as little as 18 months (Moore's Law).

The Boeing 777, GPS, and U.S. Census data systems were developed for stable environments—they were completed before any significant changes occurred in their requirements. In contrast, the FAA replacement system, FBI Virtual Case File (see www.spectrum.ieee.org/sep05/1455), NMCI (GAO[4], www.nmcistinks.com) all faced dynamic environments that changed faster than their development processes could. Predecessors of these systems were successful because their environments were stable, but the current generations en-



countered trouble because their environments had become too dynamic.

The traditional acquisition process tries to avoid risk and control costs by careful preplanning, anticipation, and analysis. For complex systems, this process usually takes a decade or more. Are there any alternatives that would take much less time and still be fit for use?

Yes. Evolutionary system development produces large systems within dynamic social networks. The Internet, World Wide Web, and Linux are prominent examples. These successes had no central, preplanning process, only a general notion of the system's architecture, which provided a framework for cooperative innovation. Individuals in the network banded into small groups to quickly produce or modify modules in the architecture. They tested their modules by asking other users to try them. The systems evolved rapidly in many small increments that aligned with current perceptions of the using environment.

Moreover, the evolutionary process embraces risk, and the patience to see what emerges. It works with nature's principle of fitness in the environment: components that work well survive, and those that do not are abandoned.

The astonishing success of evolutionary development challenges our common sense about developing large systems. We need to learn from these systems, because evolutionary development may be the only way to achieve satisfactory replacements for aging large systems and to create new, unprecedented systems.

Evolutionary development is a mature idea that has languished away from mainstream practice. In this column, we will analyze why evolutionary development does not fit the current common sense and why we need to work to change that.

## Our Current Common Sense

From its founding in 1968, the software engineering field set out to address the "software crisis," a persistent inability to deliver dependable and usable software. Fritz Bauer, one of the field's founders, believed a rigorous engineering approach was needed. He famously quipped, "Software engineering is the part of computer science that is too hard for computer scientists." Over the years, software engineers produced many powerful tools: languages, module managers, version trackers, visualizers, and debuggers are some examples. In his famous "No silver bullet" assessment (1986), Fred Brooks concluded that the software crisis had not abated despite huge advancements in tools and methods; the real problem was getting an intellectual grasp of the problem and translating that understanding into an appropriate system architecture.[2] The tools of 1986, while better than those of 1968, relied on concepts that did not scale up to ever-larger systems. The situation today is much the same: tools are more powerful, but we struggle with scalability, usability, and predictability.

Current software engineering is based on four key assumptions:

- Dependable large systems can only be attained through rigorous application of the engineering design process (requirements, specifications, prototypes, testing, acceptance).
- The key design objective is an architecture that meets specifications derived from knowable and collectable requirements.
- Individuals of sufficient talent and experience can achieve an intellectual grasp of the system.
- The implementation can be completed before the environment changes very much.

What if these assumptions no longer hold? The first assumption is challenged by the failures of large systems that used the traditional design process and the successes of other large systems that simply evolved. The remaining assumptions are challenged by the increasingly dynamic environments, often called ecosystems, in which large systems operate. There is no complete statement of requirements because no one person, or even small group, can have complete knowledge of the whole system or can fully anticipate how the community's requirements will evolve.

**The astonishing success of evolutionary development challenges our common sense about developing large systems.**

## System Evolution: A New Common Sense

To avoid obsolescence, therefore, a system should undergo continual adaptation to the environment. There are two main alternatives for creating such adaptations. The first, successive releases of a system, is the familiar process of software product releases. It can work in a dynamic environment only when the release cycle is very short, a difficult objective under a carefully prescribed and tightly managed process. Windows Vista, advertised as an incremental improvement over XP, was delivered years late and with many bugs.

The second approach to adaptation is many systems competing by mimicking natural evolution; the more fit systems live on and the less fit die out. Linux, the Internet, and the World Wide Web illustrate this with a constant churn of experimental modules and subsystems, the best of which are widely adopted.

Evolutionary system design can become a new common sense that could enable us to build large critical systems successfully. Evolutionary approaches deliver value incrementally. They continually refine earlier successes to deliver more value. The chain of increasing value sustains successful systems through multiple short generations.

## Designs by Bureaucratic Organizations

Fred Brooks observed that software tends to resemble the organization that built it. Bureaucratic organizations tend toward detailed processes constrained by many rules. The U.S. government's standard acquisition practices, based on careful preplanning and risk avoidance, fit this paradigm. Their elaborate architectures and lengthy implementation cycles cannot keep up with real, dynamic environments.

It may come as a surprise, therefore, that practices for adaptability are allowed under government acquisition rules. In 2004, the Office of Secretary of Defense sponsored the launch of W2COG, the World Wide Consortium for the Grid (w2cog.org) to help advance networking technology for defense using open-development processes such as in the World Wide Web Consortium (w3c.org). The W2COG took advantage of a provision of acquisition regulations that allows Limited Technology Experiments (LTEs). The W2COG recently completed an experiment to develop a secure service-oriented architecture system, comparing an LTE using evolutionary methods against a standard acquisition process. Both received the same government-furnished software for an initial baseline. Eighteen months later, the LTE's process delivered a prototype open architecture that addressed 80% of the government requirements, at a cost of \$100K, with all embedded software current, and a plan to transition to full COTS software within six months.

In contrast, after 18 months, the standard process delivered only a concept document that did not provide a functional architecture, had no working prototype, deployment plan, or timeline, and cost \$1.5M. The agile method produced a "good enough" immediately usable 80% success for 1/15 the cost of the standard method, which seemed embarked on the typically long road to disappointment.

## Agile Methods for Large Systems

Agile system development methods have been emerging for a decade.[1,3,6] These methods replace the drawn-out preplanning of detailed specifications with a fast, cyclic process of prototyping and customer interaction. The evolutionary design approach advocated here is a type of agile process.

The U.S. Government Accounting Office (GAO) has scolded the government on several occasions for its uncommitted lip service to agile processes.[4] The GAO believes agile processes could significantly shorten time to delivery, reduce failure rate, and lower costs. Many people resist the GAO advice because they assume careful preplanning minimizes risk and maximizes dependability and usability. However, more leaders are pushing for agile acquisition because the track record of the normal process in dynamic environments is so dismal.

The software engineering community has hotly debated preplanned versus agile processes. After a while they reached a truce where they agreed that preplanning is best for large systems where reliability and risk-avoidance are prime concerns, and agile is best for small to medium systems where adaptability and user friendliness are prime concerns.

We challenge that conclusion. Preplanning is ceasing to be a viable option for large systems. Moreover, many small systems aim to be ultra-reliable.

## Evolutionary Ecosystems

Evolutionary development uses "loosely managed" processes. Numerous successful large systems evolved through such a process—CTSS, Unix, Linux, Internet, Google, Amazon, eBay, Apple iPhone Apps, and banking applications are notable examples. All these systems relied on a common platform used by all members of the community, from developers to users. In such an ecosystem, successful prototypes transition easily to working products. It appears that the common ecosystem provides enough constraints that loose management works. The successful ecosystems were guided by a vision and a set of interaction rules that everyone in the community accepted. Building ecosystems for governments is quite challenging because of organizational impediments to information sharing.[5] We advocate much more aggressive use of loosely managed ecosystems. The W2COG was conceived to allow government to join a large ecosystem that could adaptively address its information networking needs.

Loosely managed does not mean unmanaged. Scrum and Extreme Programming (XP) are often cited as successful management approaches for agile processes.[6] Even the respected Capability Management Model (CMM) is amenable to agile development.

Whereas preplanned development seeks to avoid risks, evolutionary development mimics nature and embraces risks. The developers purposely expose emerging systems to risks to see how they fail, and then they build better system variants. It is better to seek risk out and learn how to survive it. In a natural ecosystem, only the most fit organisms survive. Fitness is nature's way of managing risk.

All the evidence says that that evolutionary processes works for systems large and small, and that risk seeking is the fastest route to fitness. There is too much at stake to continue to allow us to be locked into a process that does not work. C

**Whereas preplanned development seeks to avoid risks, evolutionary development mimics nature and embraces risks.**

**References**

1. Boehm, B. Making a difference in the software century. *IEEE Computer* (Mar. 2008), 32–38.
2. Brooks, F. *The Mythical Man Month.* Anniversary Edition. Addison-Wesley, 1995.
3. Cao, L. and Balascubramaniam, R. Agile software development: Ad hoc practice or sound principles? *IEEE Pro* (Mar.–Apr. 2007), 41–47.
4. GAO. *Defense Acquisitions: Assessments of Selected Weapons Programs.* Report GAO-06-391 (Mar. 2006); http://www.gao.gov/new.items/d06391.pdf, and *Information Technology: DOD Needs to Ensure That Navy Marine Corps Intranet Program Is Meeting Goals and Satisfying Customers.* Report GAO-07-51. (Dec. 2006); http://www.gao.gov/new.items/d0751.pdf.
5. Hayes-Roth, R., Blais, C., Brutzman, D. and Pullen, M. How to implement national information sharing strategy. *AFCEA-GMU C4I Center Symposium: Critical Issues in C4I*, George Mason University, Fairfax, VA, AFCEA (2008); http://c4i.gmu.edu/events/reviews/2008/papers/25_Hayes-Roth.pdf.
6. Schwaber, K. *Agile Project Management with Scrum.* Microsoft Press, 2004.

**Peter J. Denning** (pjd@nps.edu) is the director of the Cebrowski Institute for Information Innovation and Superiority at the Naval Postgraduate School in Monterey, CA, and is a past president of ACM.

**Chris Gunderson** (cgunders@w2cog.org), Captain (retired) U.S. Navy, is Principal Investigator of the Naval Postgraduate School W2COG and Netcentric Certification Office initiatives.

**Rick Hayes-Roth** (hayes-roth@nps.edu) is Professor of Information Systems at the Naval Postgraduate School in Monterey, California, and was CTO for Software at Hewlett-Packard Company.

## Calendar of Events

**December 17-20**
**HiPC '08: 15th International Conference on High Performance Computing**
**Bangalore, India**
**Sponsored: SIGARCH**
**Contact: Ajay K. Gupta,**
**Phone: 269-276-3104**
**Email: ajay.gupta@wmich.edu**

**January 7-9**
**International Conference on Multimedia Modeling**
**Sophia Antipolis, France**
**Contact: Benoit Huet,**
**Phone: +33-0-493008179**
**Email: benoit.huet@eurecom.fr**

**January 14-17**
**International Conference on Bio-Inspired Systems and Signal Processing**
**Porto, Portugal**
**Contact: Joaquim B. Filipe,**
**Phone: 351-91-983-3996**
**Email: jfilipe@insticc.org**

**January 19-21**
**International Conference on Agents and Artificial Intelligence**
**Porto, Portugal**
**Contact: Joaquim B. Filipe,**
**Phone: 351-91-983-3996**
**Email: jfilipe@insticc.org**

**January 19-22**
**Asia and South Pacific Design Automation Conference**
**Yokohama, Japan**
**Contact: Yutaka Tamiya,**
**Phone: +81-44-754-2663**
**Email: tamiya.yutaka@jp.fujitsu.com**

**January 20-23**
**The Eleventh Australasian Computing Education Conference**
**Wellington, New Zealand**
**Contact: Margaret Hamilton,**
**Phone: 613-992-52939**
**Email: mh@cs.rmit.edu.au**

**January 23-24**
**International Conference on Advances in Computing, Communication and Control**
**Mumbai, India**
**Contact: Srija Unnikrishnan,**
**Phone: +919869005457**
**Email: srija.unni@gmail.com**