Wisdom is not the product of schooling
but the lifelong attempt to acquire it.
- Albert Einstein

Seeding, Evolutionary Growth, and Reseeding

Ernesto Arias & Gerhard Fischer
and
Andy Gorman, Rogerio de Paula & Eric Scharff

ATLAS TAM Course, Spring 2000
Complex Systems: Why Do They Need to Evolve and How Can Evolution Be Supported

• **the basic message:** computational systems of the future
  - will be complex, embedded systems
  - need to be open and not closed
  - will evolve through their use

• **examples:**
  - domain-oriented design environments (DODEs)
    * kitchen design: extensions for microwaves, critics checking appliances against the wall (unless island kitchens), designs for disabled people (blind, in wheelchairs)
    * computer network design: new computers, new communication devices
  - Envisionment and Discovery Collaboratory (EDC) (versus SimCity)
  - operating systems (Linux) and high-functionality applications (MS-Word, Canvas, ..............)
  - courses as seeds
  - buildings (see Stewart Brand: “How Buildings Learn - What Happens after they’re built”)
The Past and The Future

<table>
<thead>
<tr>
<th>Theme</th>
<th>Past</th>
<th>Future</th>
</tr>
</thead>
<tbody>
<tr>
<td>focus of interest</td>
<td>algorithm</td>
<td>complex system</td>
</tr>
<tr>
<td>relevant theories</td>
<td>physics, mathematics</td>
<td>biology</td>
</tr>
<tr>
<td>design methodology</td>
<td>building from scratch</td>
<td>reuse, redesign, adaptation, evolution</td>
</tr>
</tbody>
</table>

- claims/challenges:
  - (many) software systems must evolve (they cannot be completely designed prior to use)
  
  - (many) software systems must evolve at the hands of the users
  
  - (many) software systems must be designed for evolution
Problems of Complex (Computer) System Design

- problems in semantically rich domains ----> thin spread of application knowledge

- modeling a changing world ----> changing and conflicting requirements

- turning a vague idea about an ill-defined problem into a specification ----> “design disasters”, “up-stream activities”

- symmetry of ignorance ----> communication and coordination problems
Answers to Problems of System Design

• problems in semantically rich domains → thin spread of application knowledge — domain-orientation

• modeling a (changing) world → changing and conflicting requirements — evolution

• turning a vague idea about an ill-defined problem into a specification → “design disasters”, “up-stream activities” — integration of problem framing and problem solving

• symmetry of ignorance → communication and coordination problems — representation for mutual understanding and mutual learning
Theory and Practice of Design—A Quest for Evolution

Dawkins — “The Blind Watchmaker”: big-step reductionism cannot work as an explanation of mechanism; we can't explain a complex thing as originating in a single step.

Simon — “The Sciences of the Artificial”: complex systems evolve faster if they can build on stable subsystems.

Petroski — “To Engineer Is Human”: the role of failure in successful design.

Brooks — “No Silver Bullet”: successful software gets changed, because it offers the possibility to evolve.

Polanyi — “The Tacit Dimension”: knowledge is tacit → we know more than we can say.
Karl Popper: Conjectures and Refutations

• John Archibald Wheeler: “Our whole problem is to make the mistakes as fast as possible.” (foreword to the book) — **breakdowns as opportunities**

• criticism of our conjectures is of decisive importance and all of our knowledge grows only through the correcting of our mistake — **critiquing systems**

• there are all kinds of sources of our knowledge but none has authority — **symmetry of ignorance and mutual competency**

• the advance of knowledge consists in the modification of earlier knowledge — **evolution**
The Economic Forces for Evolution in Software Systems

• the most critical software problem is the cost of maintenance and evolution
  - empirical studies of software costs: two-thirds of the costs of a large system occur after the system is delivered

  - claim: much of this cost is due to the fact that a considerable amount of essential information (such as design rationale) is lost during development and must be reconstructed by the designers who maintain and evolve the system

• make enhancements and evolution “first class” activities in the lifetime of an artifact
  - accept the reality of change

  - acknowledge increased up-front costs (cognitive and economic)
Integrating Problem Framing and Problem Solving

• **Simon:**
  “in oil painting every new spot of pigment laid on the canvas creates some kind of pattern that provides a continuing source of new ideas to the painter. The painting process is a process of cyclical interaction between the painter and canvas in which current goals lead to new applications of paint, while the gradually changing pattern suggests new goals.”

• **Rittel:**
  one cannot understand a problem without having a concept of the solution in mind
  one cannot gather information meaningfully unless one has understood the problem but one cannot understand the problem without information about it

• **concepts derived from these quotes:**
  - back-talk of artifacts/situations
  - reflection-in-action
  - incremental development
  - co-evolution between problem and solution
  - integration / co-evolution of upstream and downstream activities

• **empirical study: McGuckin**
AEGIS: Human Nature versus Human Error

• core of Aegis (worth 600 millions dollars): combat information center (CIC)

• in the Strait of Hormuz incident
  - search in a ordinary paperback airline flight guide
  - scenario fulfillment

• congressional hearings (Navy, Psychologists, ....)
  - Navy: “Aegis system's performance was excellent — it functioned as designed”
  - Psychologist: “Aegis software was churning out more unrelated data than the crew could readily digest”

• Aegis was the wrong system in the wrong place: designed for the open ocean, not for the twenty-five mile Strait of Hormuz → unarticulated background knowledge

• limits in testing (we test for what we are anticipating)
Three Generations of Design Methods from the History of Architectural Design

• 1st Generation (before 1970):
  - directionality and causality
  - separation of analysis from synthesis
  - major drawbacks: (a) perceived by the designers as being unnatural, and
    (b) does not correspond to actual design practice

• 2nd Generation in the early 70'es:
  - participation — expertise in design is distributed among all participants
  - argumentation — various positions on each issue
  - major drawback: insisting on total participation neglects expertise possessed by well-informed and skilled designers

• 3rd Generation (in the late 70'es):
  - inspired by Popper: the role of the designer is to make expert design conjectures
  - these conjectures must be open to refutation and rejection by the people for whom they are made (--- end-user modifiability)
Domain-Oriented Design Environments and Evolution

• support the construction and evolution of domains (program families)

• empirical fact: reuse is most successful within domains

• not just objects, but:
  - case libraries (different granularity)
  - critiquing (accumulated “wisdom” of a community of practice, “virtual” stakeholders)
  - specification component — partial characterization of a situation model
  - simulation — to understand the behavior
  - argumentation — to explore the rationale behind the artifact
The use of the TCP/IP Protocol makes file sharing among the Macintoshes more difficult.

Would you like to see the argumentation?

Yes

No
Examples of DODEs

- user interface design — **Framer**
- floor plan design for kitchens — **Janus, KID**
- graphics software — **Explainer**
- computer network design — **Network, Pronet**
- water management — **Cadswes** (with CU research center)
- Cobol programming and service provisioning — **GRACE** (with NYNEX)
- voice dialog design — **VDDE** (with USWest)
- lunar habitat design — **HERMES** (with NASA)
- graphic arts, information design, information visualization — **Schemechart, Chart ‘n’ Art**
- multi-media design environment — **eMMa** (with SRA)
Seeding, Evolutionary Growth, and Reseeding

**seeding**
- seed a domain-specific DODE using the domain-independent, multi-faceted architecture
- provide representations for mutual learning and understanding between the involved stakeholders
- make the seed useful and usable enough that it is used by domain workers

**evolutionary growth**
- co-evolution between individual artifacts and the DODE
- learning on demand and end-user modifiability complement each other
- emerging human resources: local developers, power users, gardeners

**reseeding**
- formalize, generalize, structure
- a social and technical challenge

**success example of the SER model:**
- development of operating systems
- open source movement
- courses as seeds
Evolution at All Three Levels

• evolution at the **conceptual framework** level
  - end-user modifiable DODEs
  - example: multifaceted, domain-independent architecture

• evolution of the **domain**
  - evolution was driven by new needs and expectations of users as well as new technology
  - example: computer network design

• evolution of **individual artifacts**
  - long-term, indirect collaboration
  - design rationale
  - example: the computer network at CU Boulder

• **co-evolution**
  - problem framing and problem solving (specification and implementation)
  - individual artifact and generic, domain-oriented design environment
The Seeding, Evolutionary Growth, and Reseeding (SER) Model

Artifact A
Artifact B

Evolutionary Growth

ReSeeding

Multifaceted Architecture
The Evolution towards End-User Modifiable DODEs

- General Programming Environments, e.g., Lisp, ...
  → limited reuse

- Object-Oriented Design, e.g., Smalltalk, Clos, C++, .......
  → lack of domain-orientation

- Domain-Oriented Construction Kits, e.g., Pinball, Music Construction Kits
  → no feedback about quality of artifact

- Constructive Design Environments, e.g., critics, explanations
  → design is an argumentative process

- Integrated Design Environments, e.g., combining construction and argumentation
  → lack of shared context

- Multifaceted Architecture
  → limited evolution

- Programmable End-User Modifiable Design Environments
Understanding Pitfalls Associated with Evolutionary Design

- **example:**
  - Oregon Experiment (Alexander et al., 1975)
  - a housing experiment at the University of Oregon instantiating the concept of end user-driven evolution
  - an interesting case study that end user-driven evolution is no guarantee for success

- **the analysis of its unsustainability indicated two major reasons:**
  - there was a lack of continuity over time
  - professional developers and users did not collaborate, so there was a lack of synergy

- **rationale for reseeding:**
  - making evolutionary development more predictable
  - developers and users engage in intense collaborations \( \rightarrow \) with design rationale captured, communication enhanced, and end user modifiability supported, developers have a rich source of information to evolve the system in the way users really need it
Evolution in Biology versus Evolution in the Human-Made World — a Word of Caution

• the evolutionary metaphor must be approached with caution because
  - there are *vast differences* between the world of the made and the world of the born
  - one is the result of purposeful human activity, the other the outcome of a random natural process.

• does software develop according to the “punctuated equilibrium” theory?
  - if yes, what causes the periods of increased change (subroutines, object-oriented programming, the world-wide web)?
Punctuated Equilibrium
Prototypes of Systems Supporting Evolution

- **Modifier** (end-user modifiability component of Janus)
  - mechanisms to add new objects and new behavior by the domain designer

- **Gimme**
  - web-based group memory system
  - supports communication between all stakeholders

- **Expectation Agents** (with NYNEX, UC Irvine)
  - support communication between developers and end-users
  - observe actions of end-users and compare them to descriptions of the intended use

- **Chart ‘n Art** (self-disclosure)
  - a gentle transition from direct manipulation interfaces to end-user programming

- **Visual Agent Talk (VAT)**
  - representations of conditions, actions and rules as graphical objects
  - interface support (drag and drop) for end-user programming
Conclusions

• complex (software) systems should be regarded as “living” entities which are open and evolve

• the seeding, evolutionary growth, reseeding (SER) model is a feasible model for the evolutionary design of complex software systems

• complex (software) systems need to be evolvable by their users, not just by their developers

• these requirements create many interesting research challenges for
  - end-user modifiability
  - decentralized system development
  - new conceptualization of the WWW
    - culture changes in individuals (consumers → designers) and organizations