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

Evolutionary Design of Complex Systems

Gerhard Fischer
Center for LifeLong Learning & Design (L^3D)
http://www.cs.colorado.edu/~l3d/
Department of Computer Science and Institute of Cognitive Science
University of Colorado, Boulder

Tutorial (December 5, 2000) at OZCHI 2000

Overview

- System Design Problems and Challenges
- Domain-Oriented Design Environments (DODEs)
- Evolution: The SER Model
- Assessment

Problems of 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" (between different communities of practice) → **communication and coordination problems**
- reality is not user-friendly → **useful and usable**

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**
- reality is not user-friendly → useful *and* usable — **collaborative work practices, power users**

Computational Environments Need to Be Open and Evolvable

- **the basic message:** computational environments of the future
  - will be complex, embedded systems
  - need to be open and not closed
  - will evolve through their use by collaborating communities of practice acting as “active contributors/designers” and not just “consumers”

- **examples:**
  - SimCity
  - operating systems and high-functionality applications
  - domain-oriented design environments
  - courses as seeds
  - electronic journals (JIME) → Journal of Interactive Media in Education at http://www-jime.open.ac.uk/
  - open source environments

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:
    - perceived by the designers as being unnatural, and
    - does not correspond to actual design practice

- **2nd Generation (in the early 70's):**
  - 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's):**
  - 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)

Seeding, Evolutionary Growth, and Reseeding

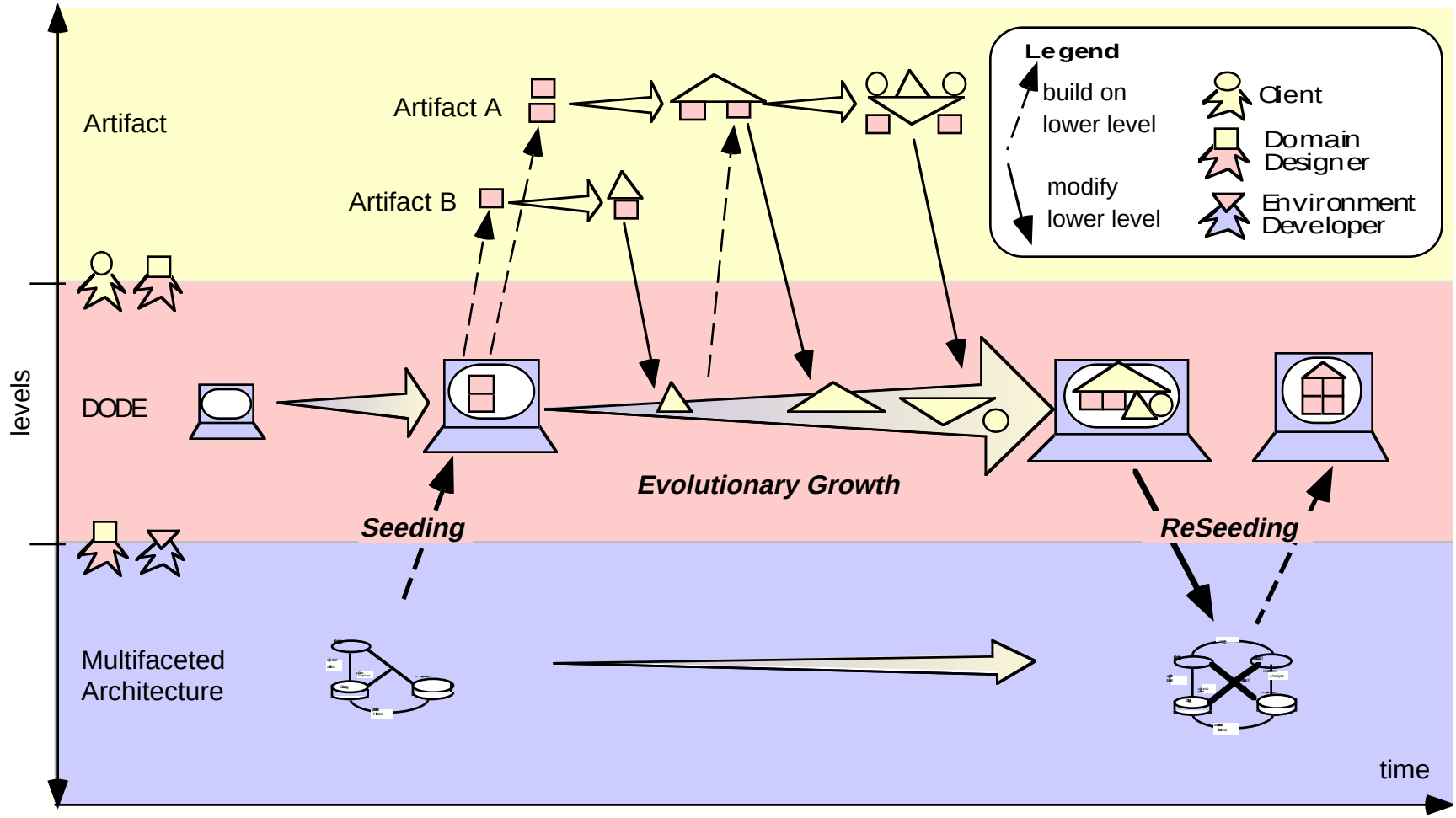
- **seeding**
  - seed a specific domain-oriented design environment using the domain-independent, multi-faceted architecture
  - provide representations for mutual learning and understanding between the involved stakeholders
  - make the seed useful and usable enough that it is used by domain workers

- **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 developments
  - courses as seeds

The Seeding, Evolutionary Growth, and Reseeding (SER) Model



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

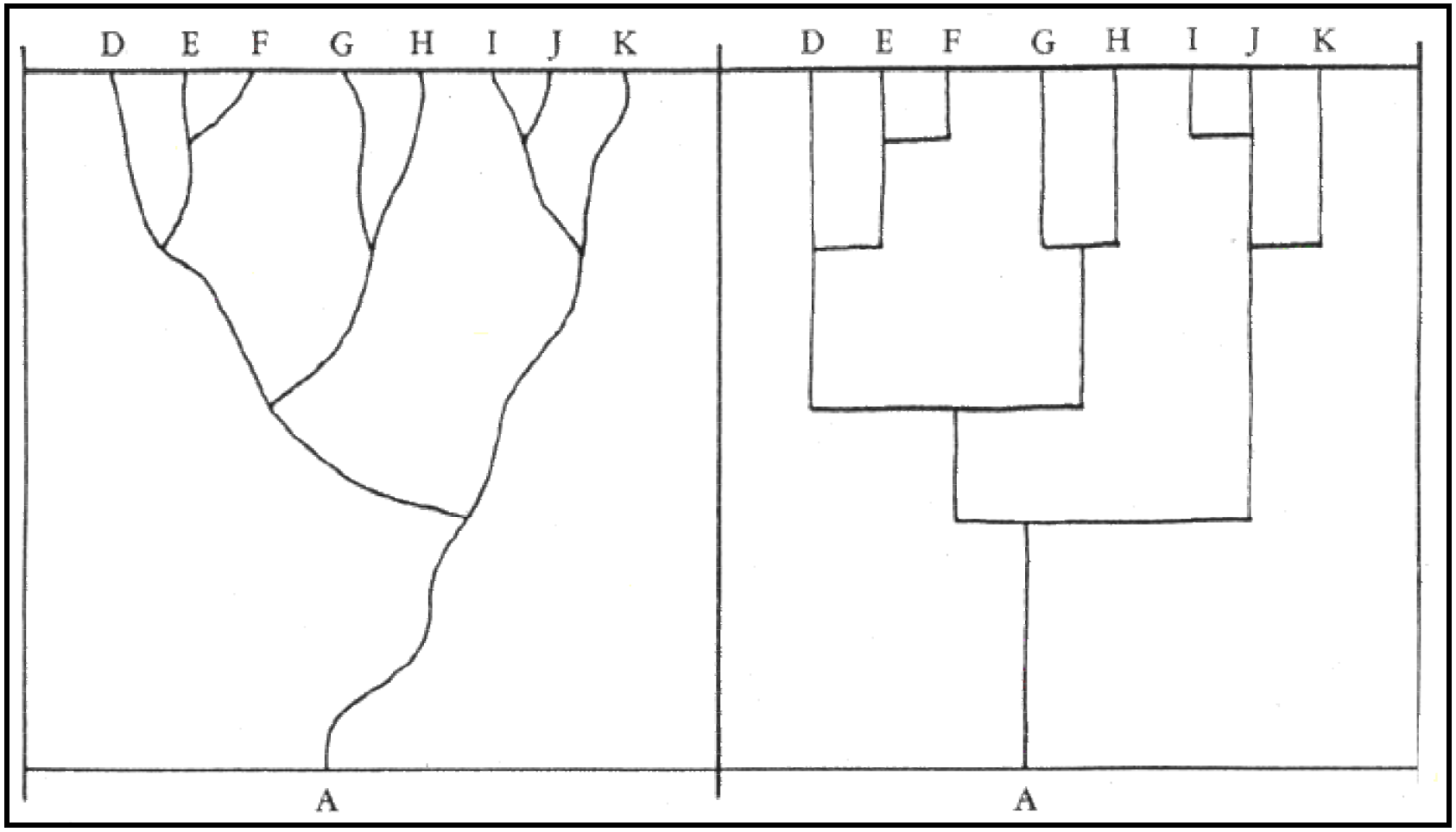
- **co-evolution**
  - problem framing and problem solving (specification and implementation)
  - individual artifact and generic, domain-oriented design environment

Evolution in Biology versus Evolution in the Human-Made World — a Word of Caution

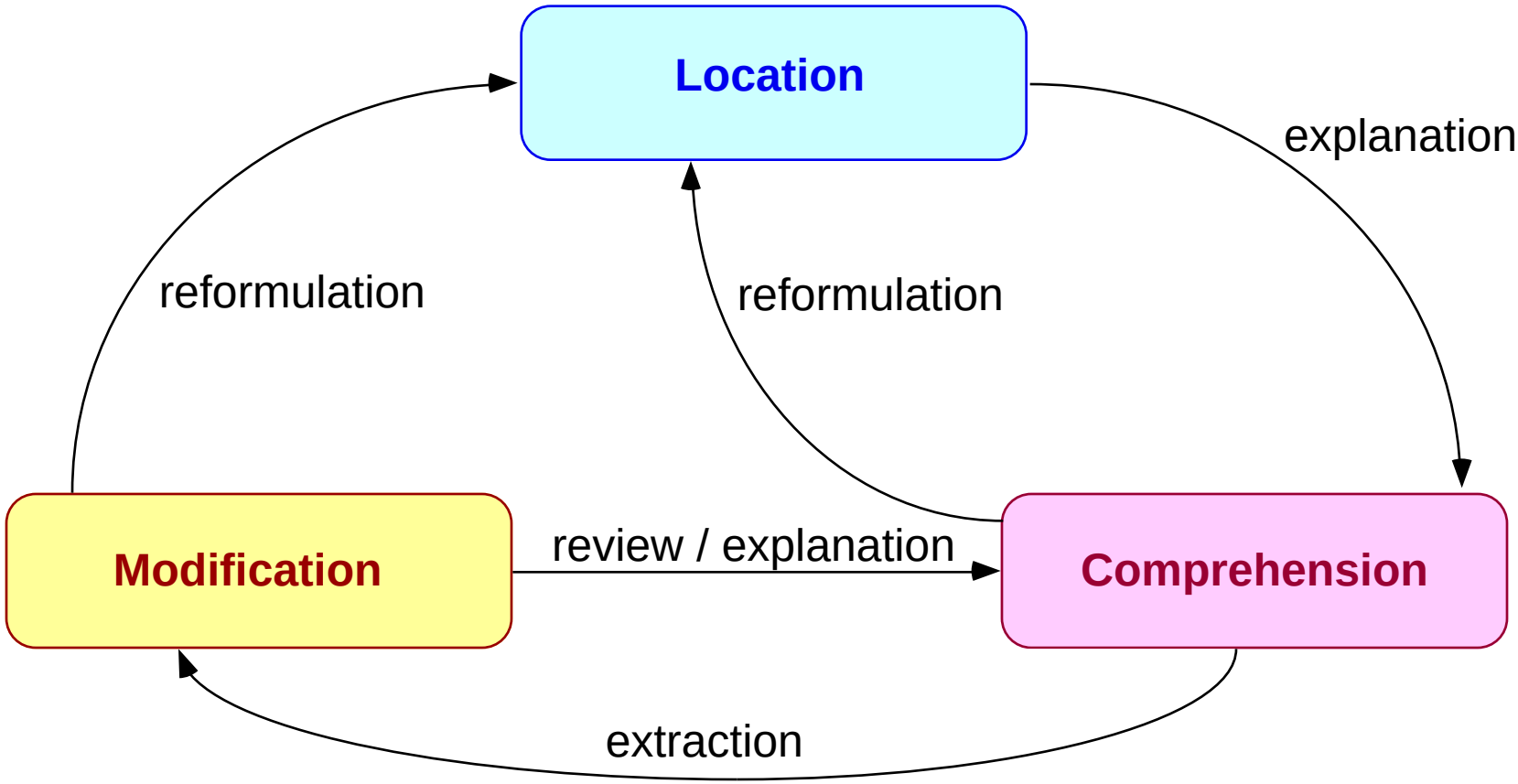
- **the evolutionary metaphor must be approached with caution because**
  - there are *vast differences* between the world of the made and the world of the born
  - one is the result of purposeful human activity, the other the outcome of a random natural process

- **does software develop according to the "punctuated equilibrium" theory?**
  - if yes, what causes the periods of increased change (subroutines, object-oriented programming, the Web)?

Punctuated Equilibrium



A Conceptual Framework for Evolution and Reuse



End-User Computing

- competent practitioners usually know more than they can say — **tacit knowledge** is triggered by situations, by breakdowns

- **end-users:**
  - are the *owners* of problems, have the domain knowledge, are the users of computational artifacts
  - regard computers as useful machines capable of helping them work more productively, creatively, and with greater pleasure
  - like computers because they get their work done

- **computer scientists / programmers**
  - find computer themselves intrinsically interesting
  - like computers because they get to program

- **ultimate goal/belief:**
  - end-users will use, tailor, extend and create their own computational artifacts when they have domain-oriented design environments
  - community of users will develop: power users, local developers, gardeners

Prototypes of Systems Supporting Evolution

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

- **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

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

- **Dynasites**
  - Dynagloss
  - Living Books
  - Virtual Libraries
  - Courses as Seeds

Comparing Conventional Books and Living Books

http://Seed.cs.colorado.edu/LivingBook.Home.fcgi

Conventional Book	Living Book
closed – the content is finalized at write-time	open – content evolves through small contributions at read-time
static – the book is always viewed in the same way	dynamic – views are computed at read-time; many different ways of viewing the book are possible
a reference artifact	a medium of communication
authors known at write-time	new authors can join at anytime.
in danger of becoming obsolete	long lifecycle driven by continual authoring
content controlled by authors	content contributed ad hoc (but just how this is realized is a design decision)
linkages between parts of the book and between book and other artifacts are implicit; reader does work to follow linkages	linkages are supported by hypermedia (as much as possible)

"Courses as Finished Products" versus "Courses as Seeds"

http://www.cs.colorado.edu/~l3d/courses/atlas-2000/

Courses as finished products	Courses as seeds
learners answer problems given to them by the instructor	learners construct knowledge about topics that are personally meaningful
learners interact mainly with the teacher and compete with other learners for grades	learners are a community of practice and collaborate to build shared understanding
learners are complete novices in the subject matter and make no contribution to other students	course participants are knowledgeable people in their own working environments who have much to offer
a course is given over a period of years, more or less in the same form	a course is considered as a seed that will evolve continuously
learners are recipients of knowledge (the assumption is that the teacher/instructional designer has all the relevant knowledge)	learners are not just passive recipients of knowledge, but active contributors, i.e., they actively co-design the class curriculum
from time to time the teacher/instructional designer will incorporate new ideas into the course so the course doesn't become outdated	the content of the course is enriched through the interaction of knowledgeable people, and important and relevant additions are incorporated into the course before it is taught the next time

Lessons Learned from the "Design for Evolution" of our Socio-Technical Systems

- **seeds** need to be functional enough that they are used by skilled domain designers in their work

- **evolutionary growth** requires support for end-user modification and programming, sociological structure of communities of practice with power users and local developers

- **reseeding**
  - of the application (technological reseeding) — evolving the tool
  - of the information space (structural reseeding) — evolving the content
  - experience with Dynasites → specific tools are needed: *"Dynasites was designed to accumulate information, but not to edit or restructure the accumulated information"*

Assessment of DODEs

- **current limitation of DODEs:**
  - limited success models — specifically lack of experience with evolutionary growth in naturalistic settings
  - tool mastery burden

- **research issue for DODEs**
  - design rationale
  - case-based reasoning
  - integrated artifact memories
  - multi-user DODEs
  - evolutionary growth through use
  - new contracts between stakeholders
  - sustainability

- **challenges**
  - the question is how — not why?
  - how large or small, general or specific should a domain be?
  - cost-effectiveness: powerful substrates are needed

Conclusions

- software systems should be regarded as **“living entities”**

- DODEs and the SER model are **feasible architectures and models**
  - for the evolutionary design of complex software systems
  - for constructing, capturing and evolving knowledge

- **domain-specificity** is critical

- individual artifacts within a DODE, domains as specific DODEs and domain-independent architectures for DODEs **co-evolve**