Software Evolution

Alexander Serebrenik

Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

Organisation

- **Quartile 3:**
  - Lectures:
    - Tuesday: 13:45-15:30 (AUD 14, except April 2)
    - Thursday: 10:45-12:30 (PAV J17)
  - **http://www.win.tue.nl/~aserebre/2IS55/2012-2013/**
  - **http://videocollege.tue.nl/**
- Master students: CSE, ES, BIS, AT
- 5 ECTS = 140 hours
  - 140 – 2*14 = 112 hours
- a.serebrenik@tue.nl
  - 3595, MF 7.143

Assistant:
Bogdan Vasilescu 2010

Learning objectives

- list important **challenges** associated with software evolution;
- develop and discuss **methods and tools** addressing these challenges, their advantages and disadvantages;
- **apply** the methods and tools to existing software systems;
- **interpret the results** obtained in a scientifically responsible way.

Assignments and NO EXAM!

- http://peach.win.tue.nl/

- 5 assignments
  - "Assignment 0" – technical/administrative preparation.
  - **Final grade = average of the four best grades**

- Some individual, some in pairs
  - Estimate your strengths and weaknesses when teaming up.
  - Look at the assignments *in advance*.
  - If something is not clear, ask.
  - Workload might seem higher since you work mostly during the semester and not during the exam period

A typical assignment

- **Report**
  - What **problem** will you study?
  - What software **system**(s) will you study?
  - How did you **design** your experiment?
    - Design **decisions** and **tools** used to obtain, analyse and visualize data
    - Reproducibility: include all **commands/source** code you have used
  - What **results** have you obtained?
  - How do you **interpret** these results?
  - What might have affected **validity** of your results?

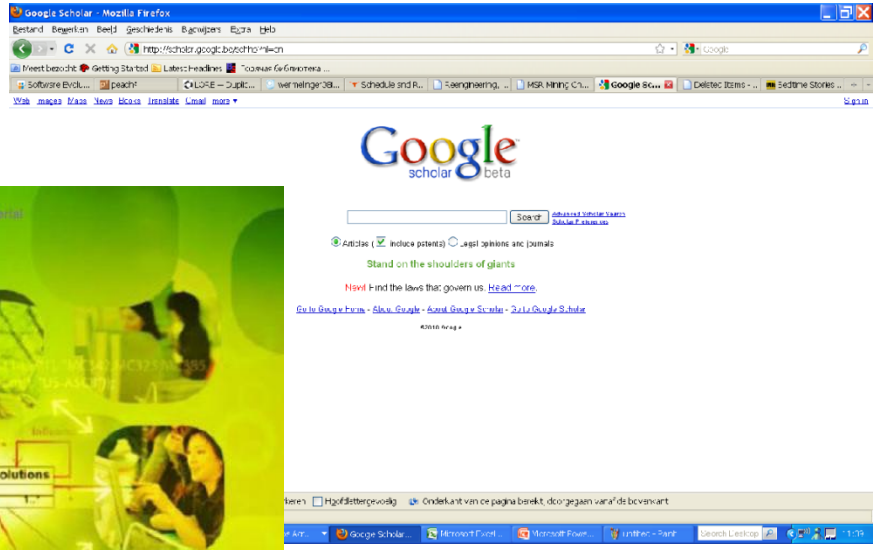
Software evolution and others…

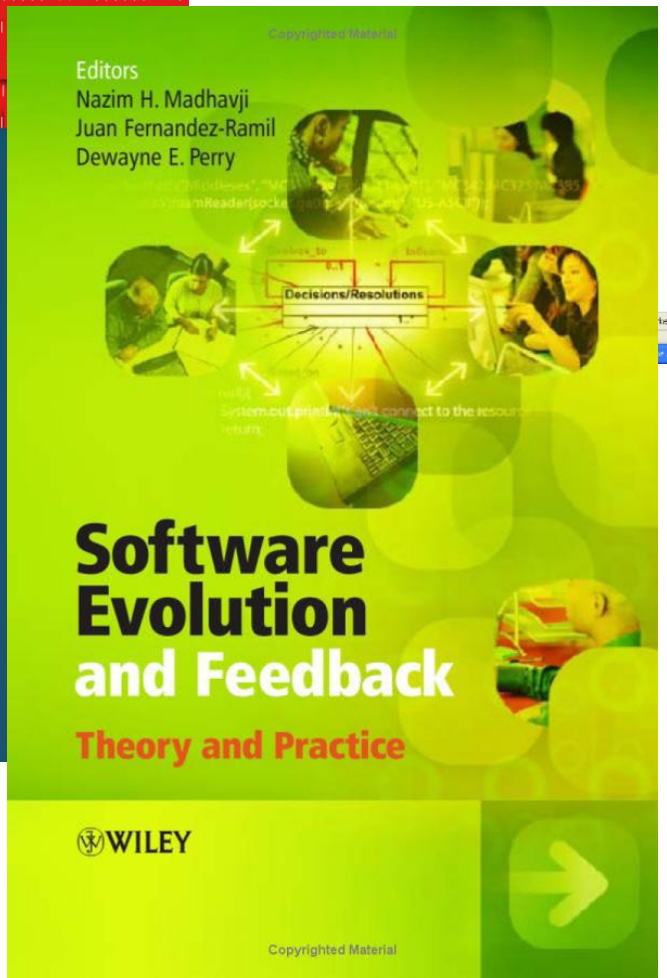
- **No formal prerequisites**
  - “Generic language technology” is recommended Basic statistics is an advantage (e.g., correlation)

- Interest in
  - analysis of software systems
  - tool development
  - social aspects of software development

- Programming skills
  - Java
  - scripting languages (recommended)

Reading material (mostly…)

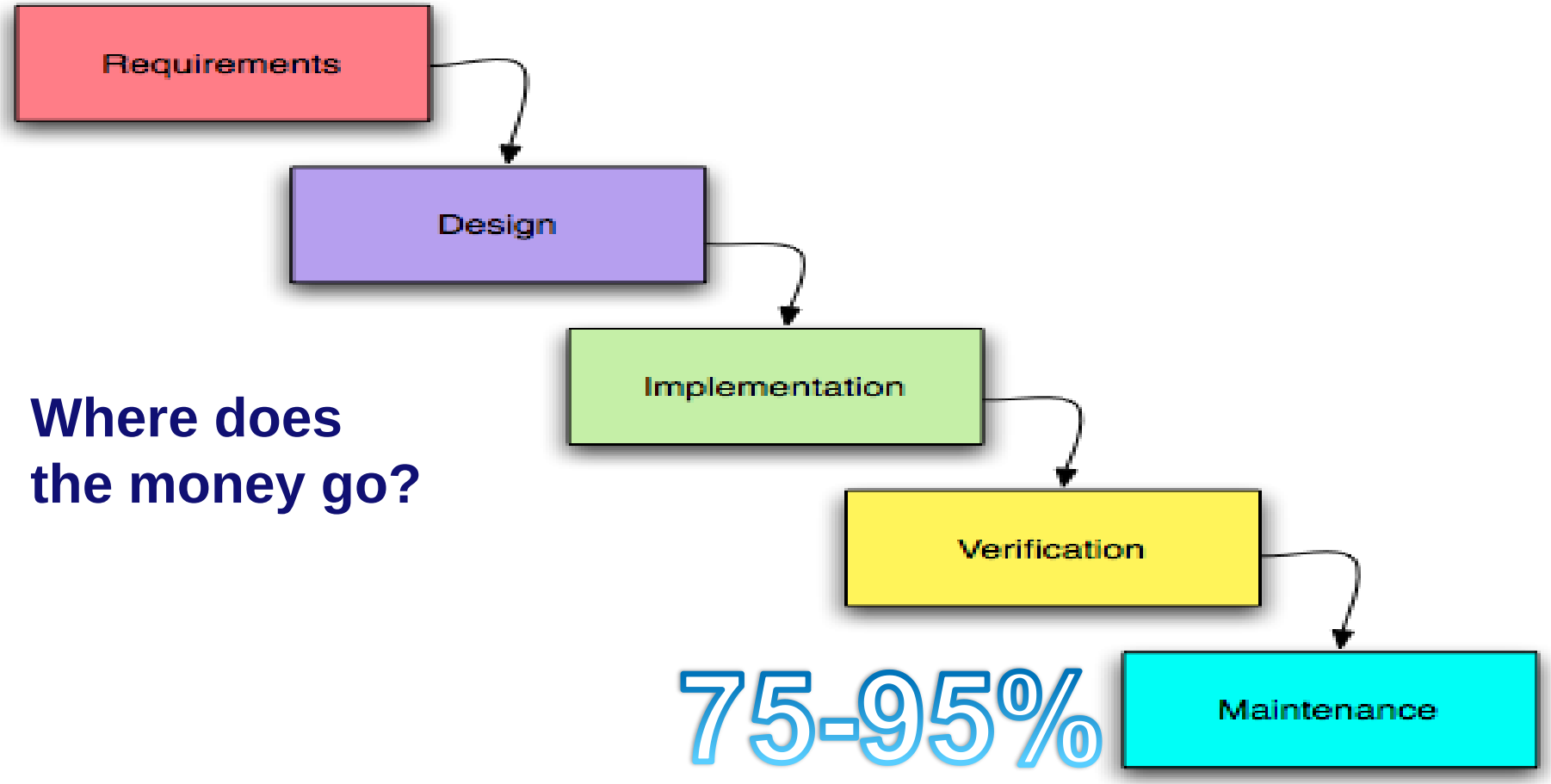




What is this course about?

Software Evolution. In the beginning.

- **Royce 1970, “Waterfall model”.**



Where does the money go?

Why is maintenance so expensive?!

- **Software is**
  - **crucial for modern society**

Toyota Prius. 160,000 vehicles recalled.

Ariane 5. Launch failed

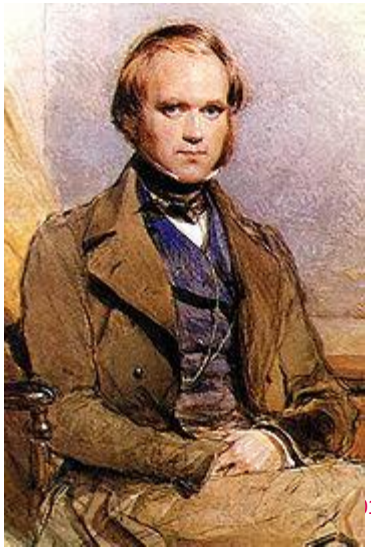
Change?

- **GNOME project**
  - **Sarma, Maccherone, Wagstrom, and Herbsleb ICSE'09**
  - **10 years, 1000 developers**
  - **2.5 millions changes**
- **Mozilla**
  - **Lanza**
  - **6 years, hundreds of developers**
  - **> 1 million changes**

- **Lots of small changes leading to a new behaviour…**

Evolution: one change a top of another one

Evolution is staged process of progressive change over time in the properties, attributes, characteristics, behaviour of some material or abstract, natural or artificial, entity or system

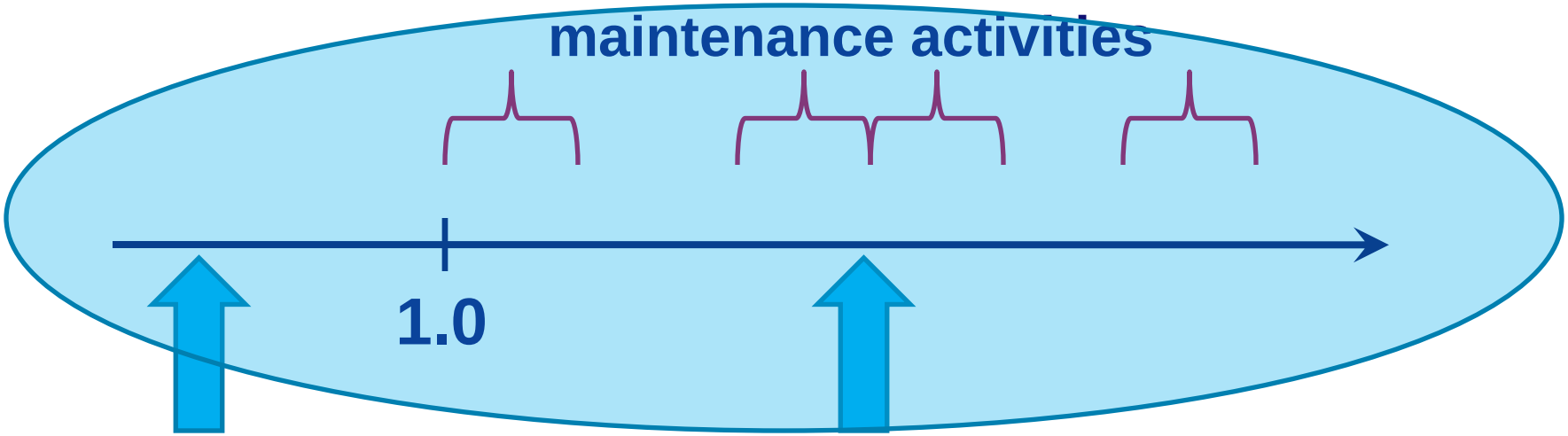


Charles Darwin

Meir Manny Lehman

Evolution vs. Maintenance

Software evolution



“Software development” (Royce 1970)

“Software maintenance” (Royce 1970)

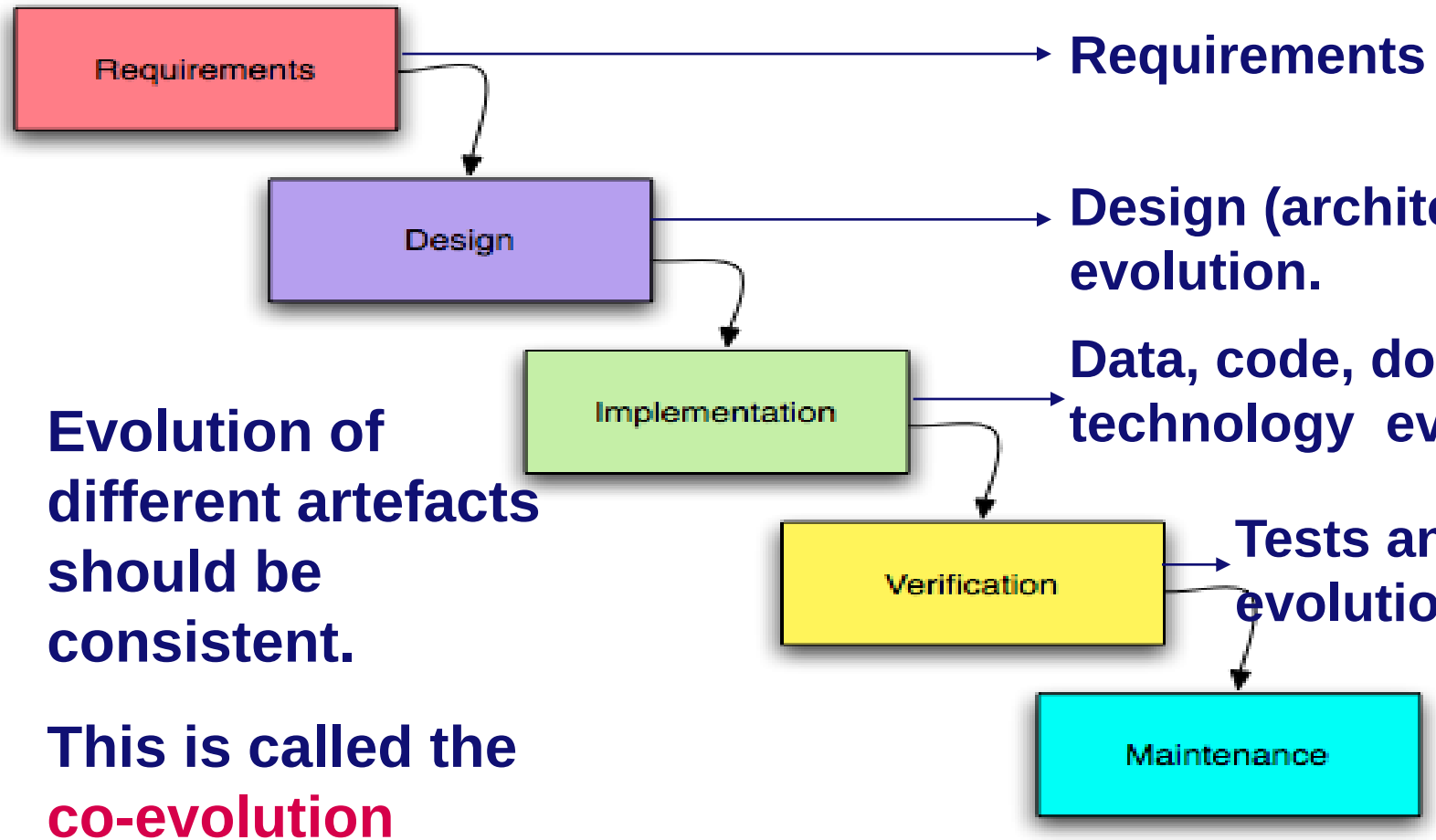
Why do we want to study software evolution?

- Software = the weakest link (often)
- Evolution "in general" makes things more complex
  - Science:
    - What is the nature of software evolution?
      - Psychology, sociology and organization theory, economics, law…
  - Engineering:
    - Where did the things go wrong?
      - incorrect, too complex, out of sync with other artefacts
    - Where can/will the things go wrong?
      - prediction, week spot identification, …
    - What can we do to prevent the things from going wrong?

Three questions for today

- **What software artefacts are subject to evolution?**
- **Why do they evolve?**
- **How do they evolve?**

What does evolve?



Requirements evolution

Design (architecture) evolution.

Data, code, documents, technology evolution

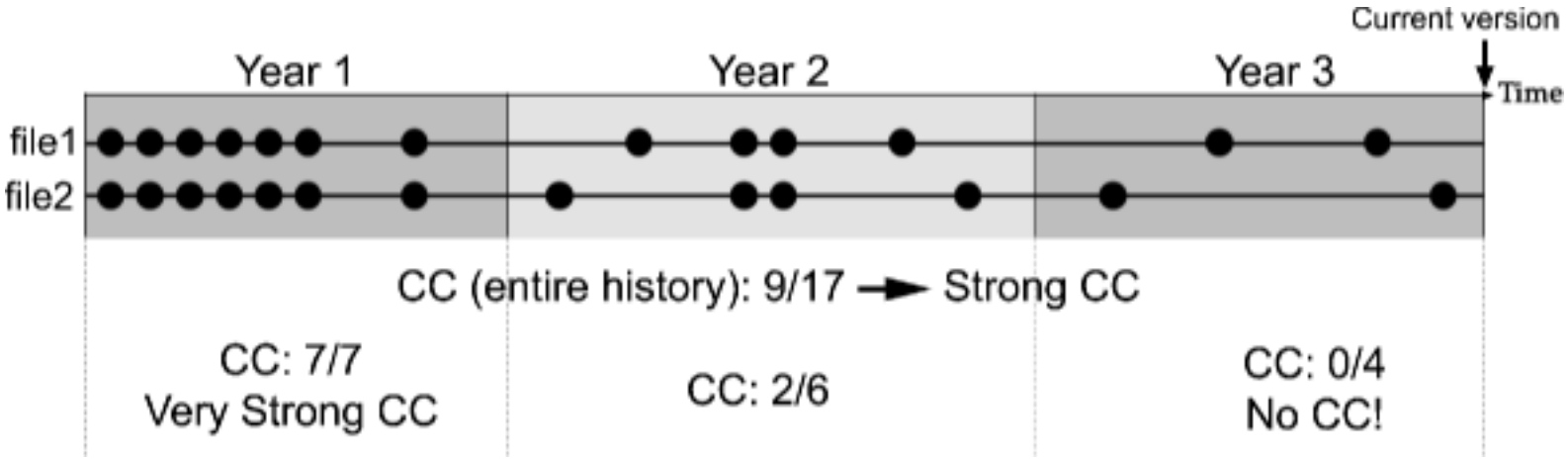
Tests and proofs evolution

Evolution of different artefacts should be consistent.

This is called the co-evolution problem.

Co-evolution is time-dependent!

- **Assumption: files committed together are co-evolving.**



D'Ambros, Gall, Lanza, and Pinzger. "Analysing Software Repositories to Understand Software Evolution"

Co-evolution: Points of discussion

- **What should co-evolve?**
  - **Code and database table definitions**
  - **Code and design documentation**
  - **Code and tests**
  - **Code and programming language**
  - **Code and 3rd party software**
  - **Different code elements (packages, modules, files…)**
  - **…**

- **What constitutes inconsistency?**
- **How to detect inconsistencies?**
- **How to ensure absence of inconsistencies?**

Why does software evolve?

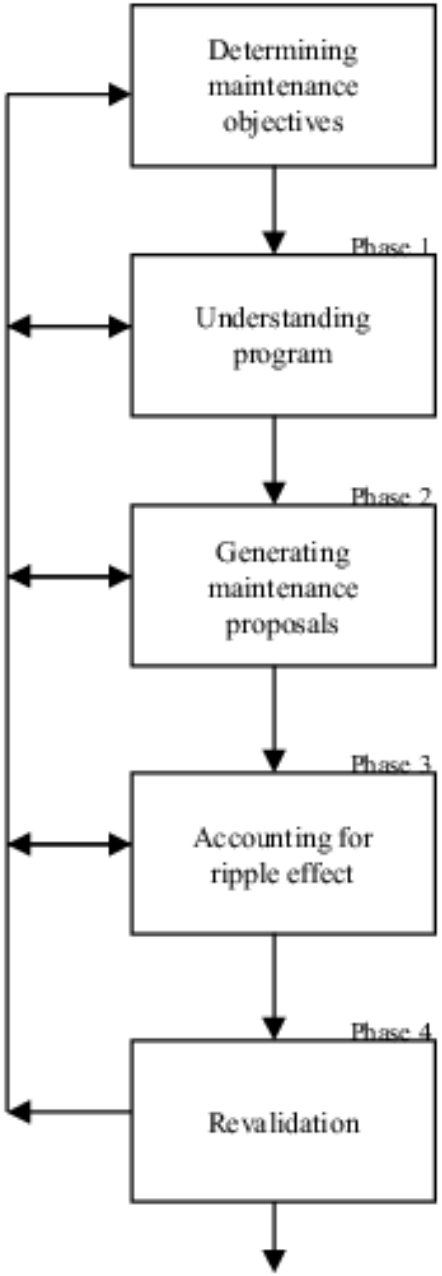
Swanson 1976

- **corrective maintenance: to address processing, performance or implementation failure;**
- **adaptive maintenance: to address change in the data or processing environments;**
- **perfective maintenance: to address processing efficiency, performance enhancement and maintainability**

Which maintenance type tries to address the co-evolution problem?

How do the system evolve?

- **Evolution in small: series of changes**
- **Each change has to be understood and confirmed: impact analysis**
- **Yau and Collofello 1980**



Do all programs evolve?

- **What about your student assignments?**

- **There is a specification**
- **Specification is complete and fixed**
  - **All that matters has been completely defined**
  - **Changed specification = new specification**
- **Correctness wrt the specification is all it matters**
  - **No "extra" requirements**

- **S-type systems [Lehman 1985a] = no evolution**
- **NB: Presence of specification: not enough for S-type**

S-type systems

- **Rarely observed in practice: Why?**

- **Important:**
  - **Some design approaches aim at improving formality**
  - **But implicitly assume absence of evolution**
    - **S-type systems definition clarifies shortcomings of such approaches (Acme ADL, …)**

- **Are S-type systems useless in practice?**

P-type: Restricted evolution

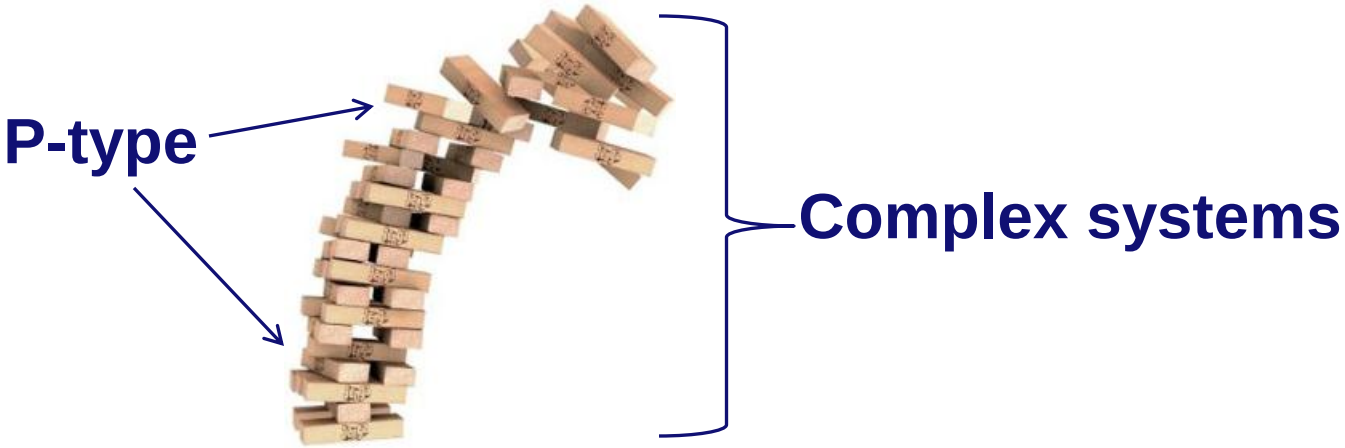
- Problem [Lehman 1980] or paradigm [CHLW 2006]
- P-type systems = systems that should always be consistent with a single external paradigm:
  - Virgo: laws of physics
- Standards can be regarded as a paradigm

- What are the differences between a paradigm and a specification?

- In what way is the evolution of P-type systems restricted?

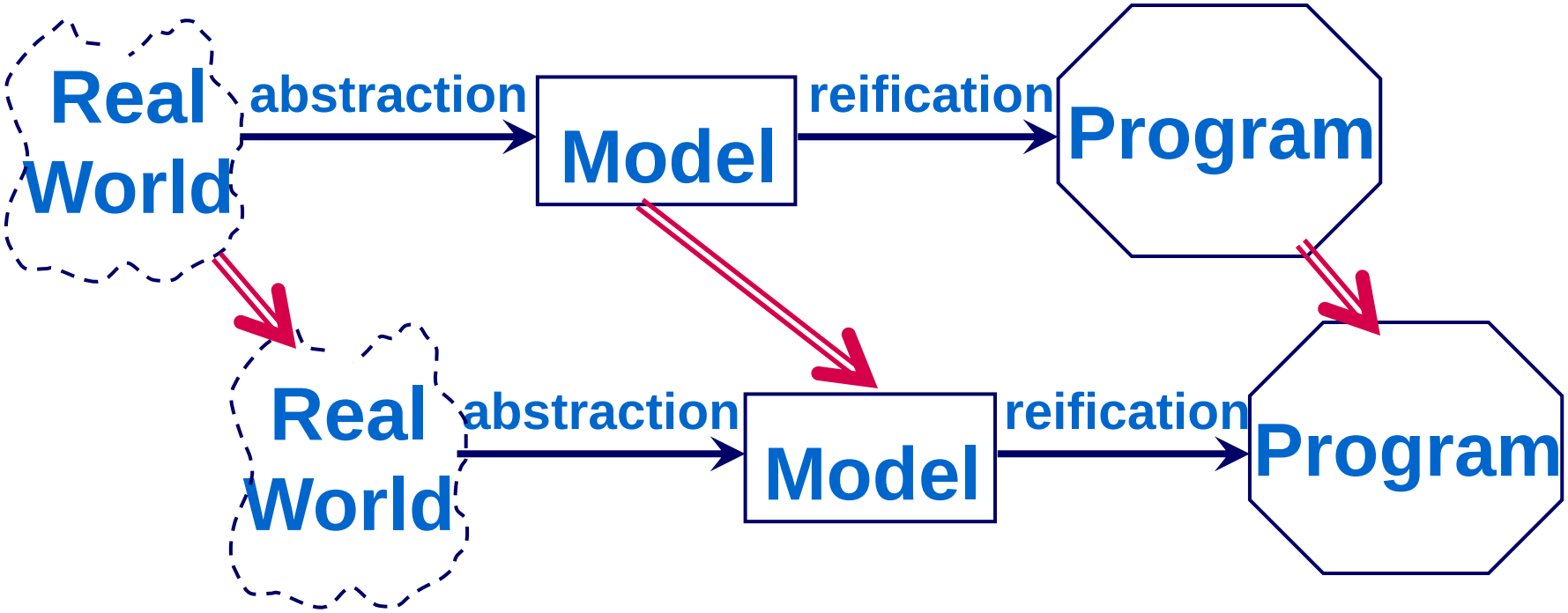
P-type systems and reuse

- **Reuse techniques foster P-type solutions**
  - **Design patterns**
  - **“Stable intermediate forms” (Simon 1969)**
    - **Building blocks to construct more complex systems**



E-type systems

- **E-type systems = systems that operate in the real world (or address it)**
- **Should evolve since the real world evolves:**



E-type systems

- **“Unrestricted evolution”**

- **Lehman has observed 8 laws of software evolution**
  - **Experience-based**
  - **Some might sound obvious**
  - **Have been changed and reformulated a number of times…**

- **We will look at them one by one…**

1. Continuing Change

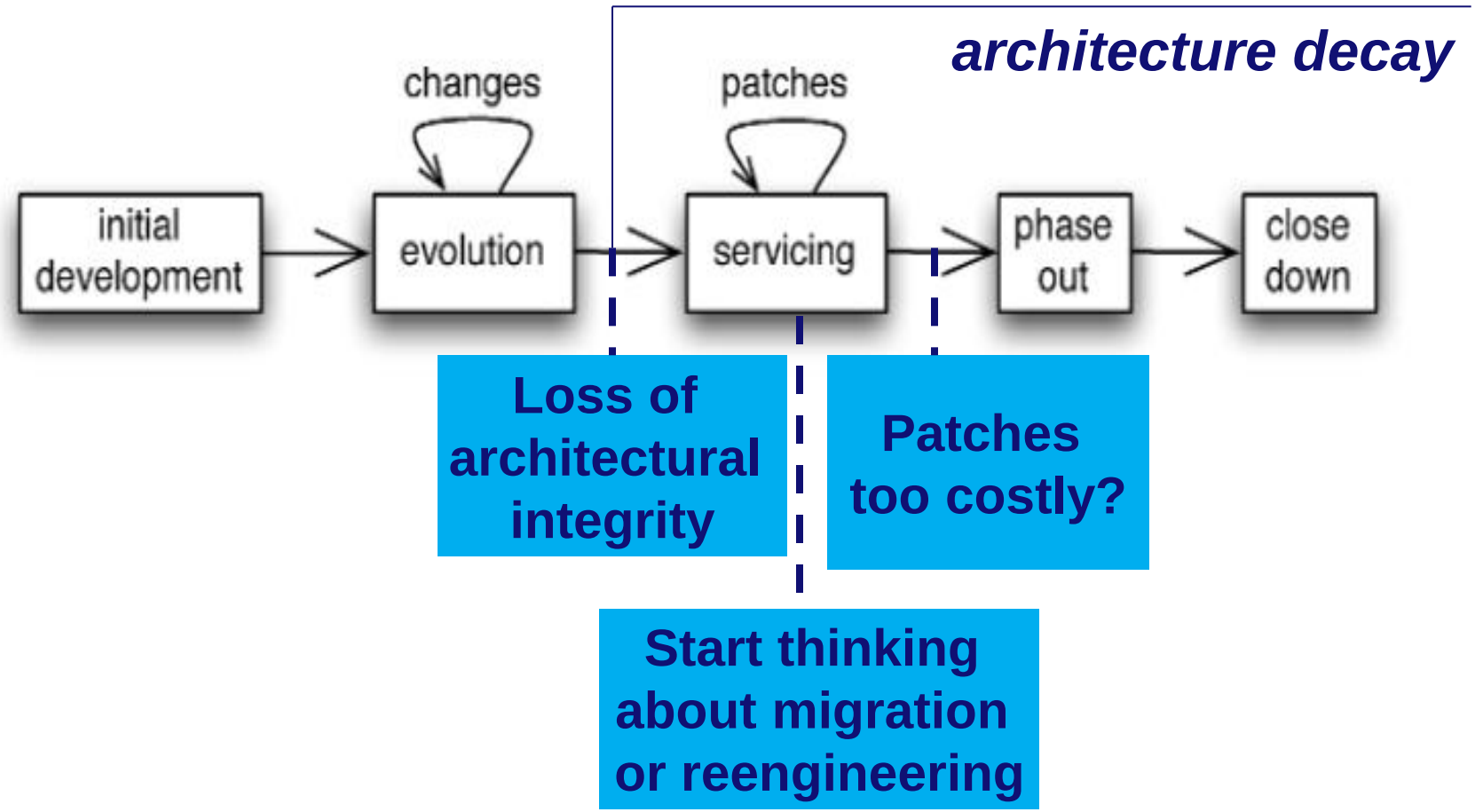
- An E-type system must be continually adapted, else it becomes progressively less satisfactory in use

- Follows from:
  - E-type systems operate in the real world (or address it)
  - Real world changes

- Would you use 

today?

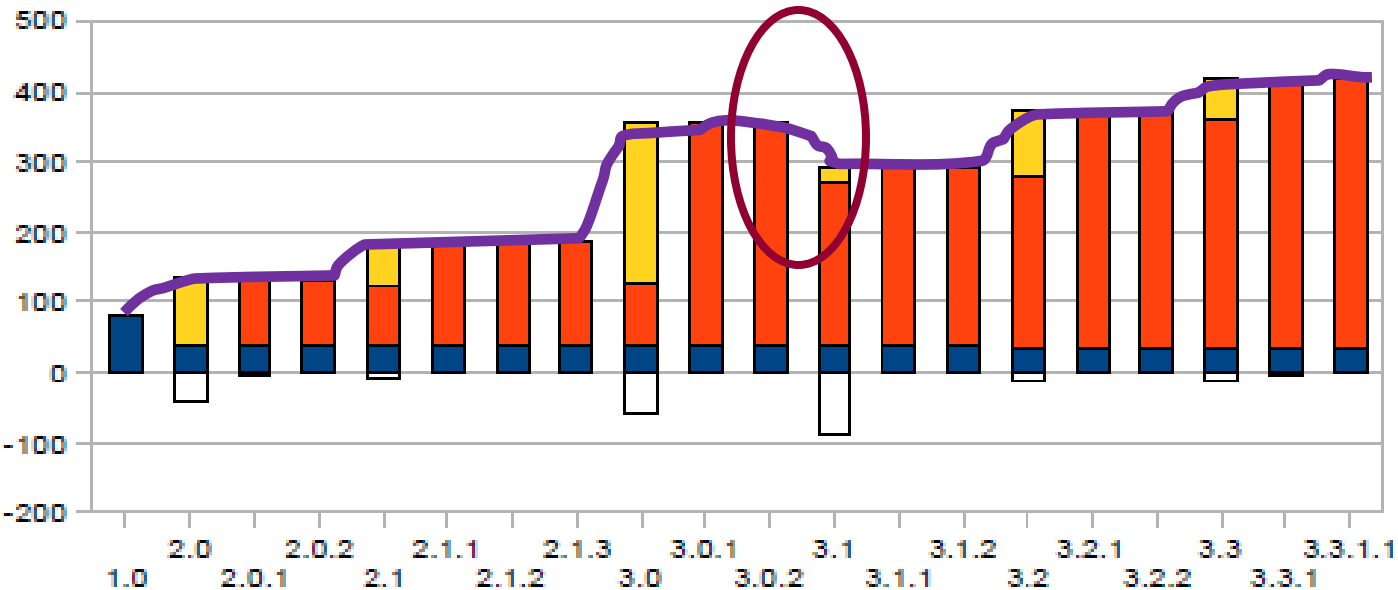
Step aside: is the continuous change always possible?



- **Bennett and Rajlich (2000)**

2. Increasing complexity

- **As an E-type system is changed its complexity increases and becomes more difficult to evolve unless work is done to maintain or reduce the complexity.**



Number of internal dependencies in Eclipse: added, kept, kept from r1, deleted.

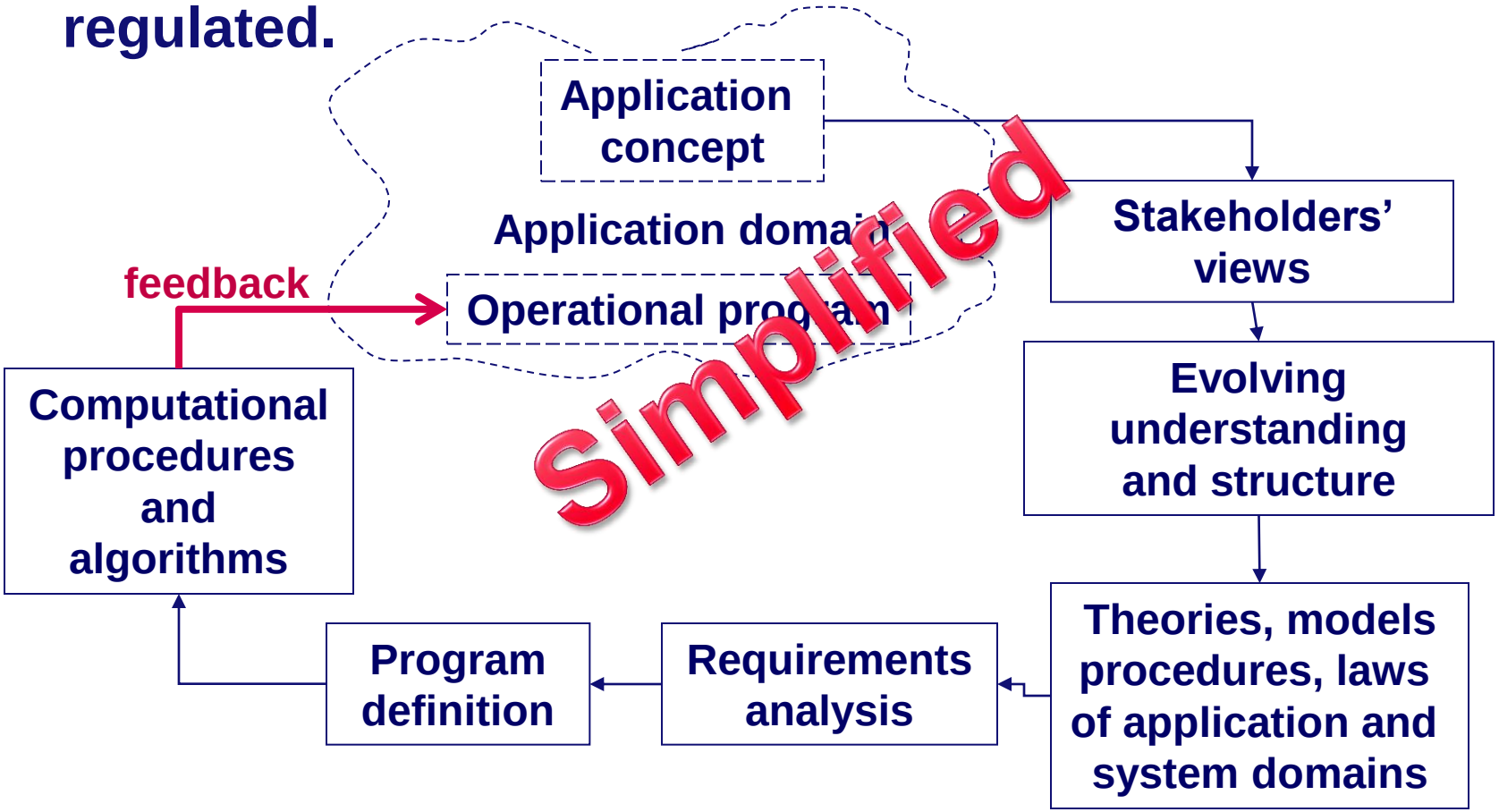
Wermelinger, Yu, Lozano, ICSM 2008

Increasing complexity

- How do we define complexity?
  - Dependencies, execution paths, inheritance, …
  - Metrics
- How do we detect trends, changes and outliers?
  - Visual inspection
  - Statistical analysis
- How we map the change points to recorded improvement activities?
  - Logbooks
  - Automatic detection of refactorings

3. Self Regulation

- **Global E-type system evolution is feedback regulated.**

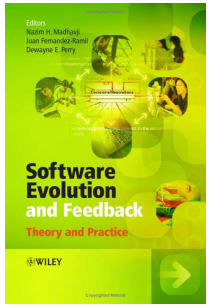


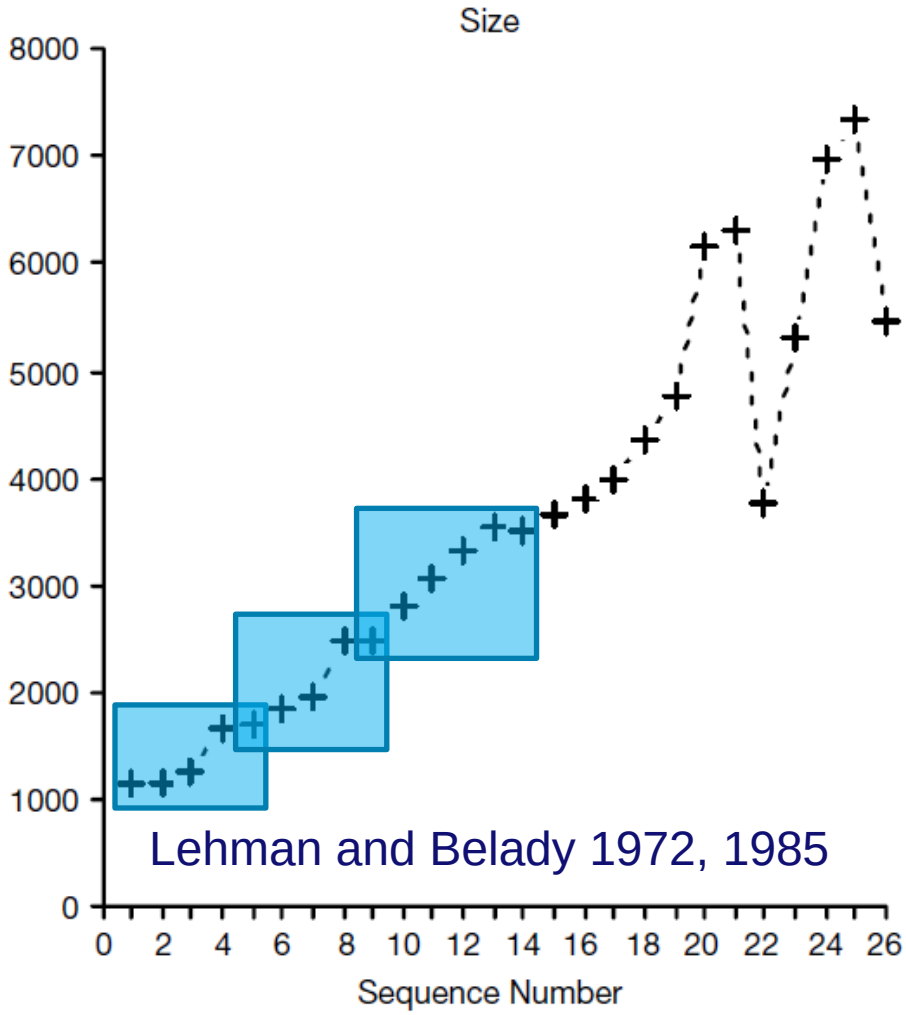
M.M. Lehman: Software evolution as a feedback loop (simplified)

Feedback and Growth Pattern

- **Feedback necessitates ability to react , i.e., control.**
- **Repeated cyclic pattern is typical for feedback.**
- **More?**

  Ch. 17 of





4. Conservation of Organizational Stability

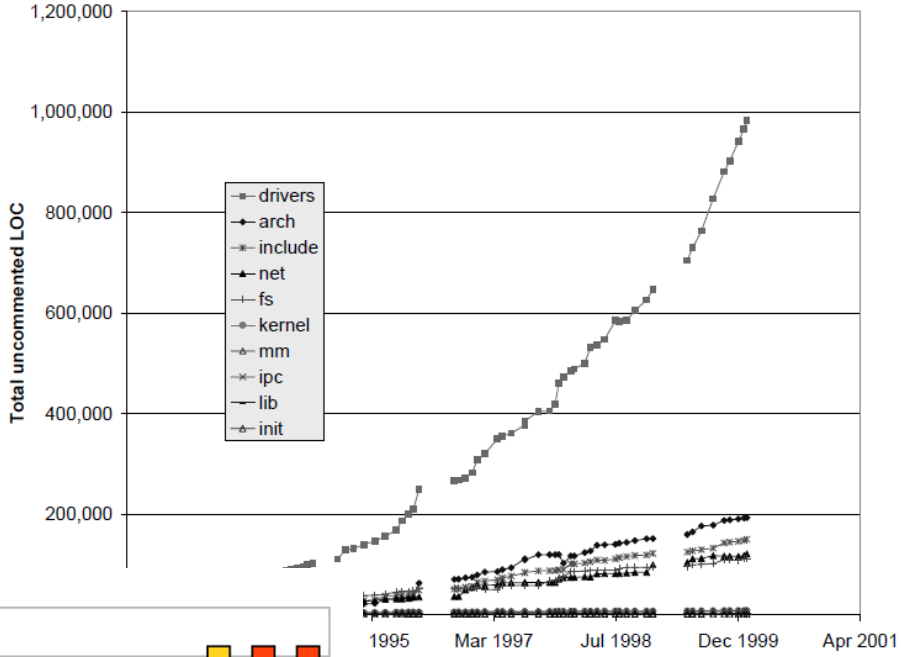
- **The work rate of an organization evolving an E-type system tends to be constant over the operational lifetime of that system or phases of that lifetime.**

- **Rather controversial: managers have no power?**
  - **Supported, e.g., by Gefen and Schneberger**
  - **No evidence found, e.g., by Lawrence**

- **"Phases of that lifetime" – discontinuity!**

5. Conservation of Familiarity

- **In general, the incremental growth (growth rate trend) of E-type systems is constrained by the need to maintain familiarity.**



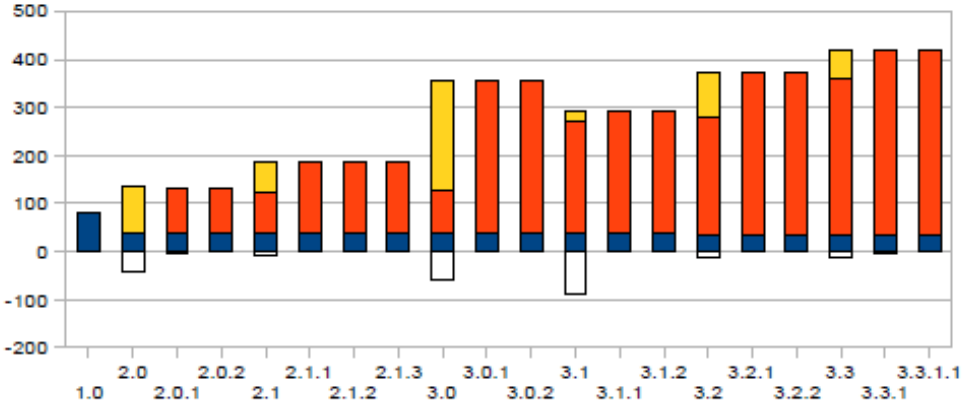
- **Lehman: growth is linear**
- **Godfrey and Tu: not**

for Open Source (Linux)

- **Wermelinger, Yu, Lozano:**

linear for architecture

(Eclipse)

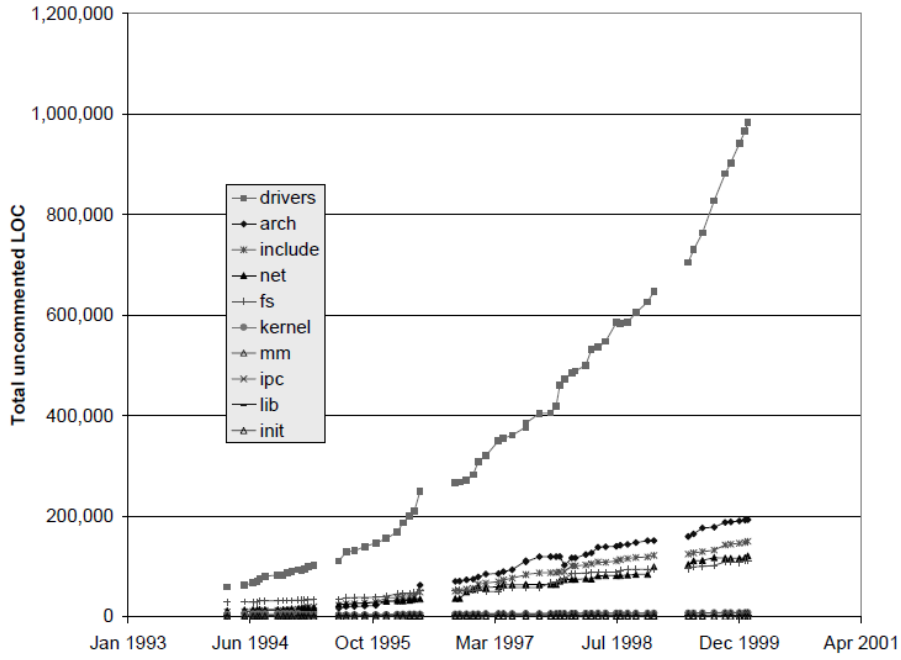


2000

6. Continuing Growth

- **The functional capability of E-type systems must be continually enhanced to maintain user satisfaction over system lifetime.**

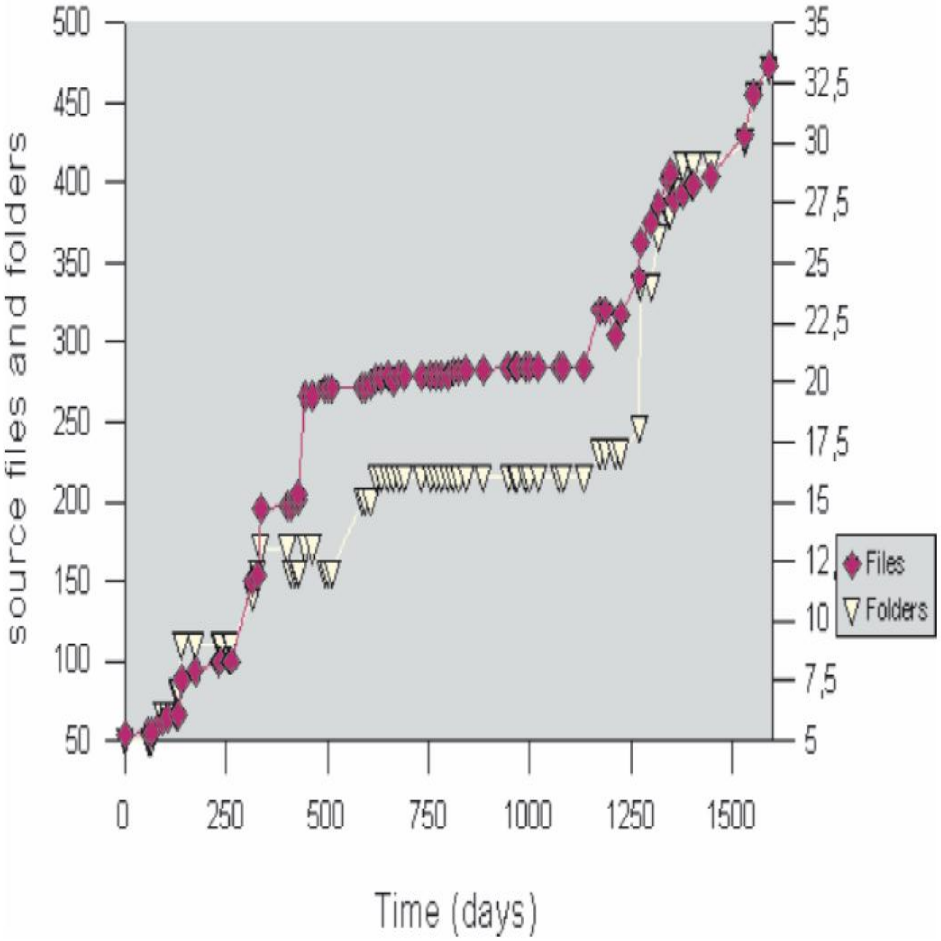
- **In earlier versions the law talked about "size".**



6. Continuing Growth (cntd.)

- How to measure functional capability?
  - Is it reflected in # lines of code, modules, etc.?
  - Measure of functionality: Function points
    - Usually require description to be calculated
    - Description ≠ functionality
- Continuing(?) growth

Staged growth: typical for Open Source Software?



Growth in Gaim (Ramil, Capiluppi 2004)

7. Declining Quality

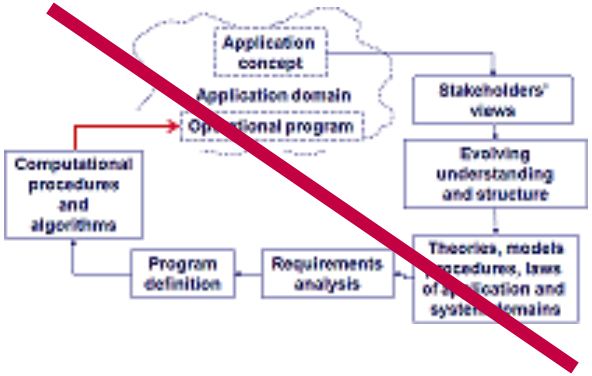
- **Unless rigorously adapted and evolved to take into account changes in the operational environment, the quality of an E-type system will appear to be declining.**

- **Again, would you use**

- **How would you define quality? Adaptation?**

8. Feedback system

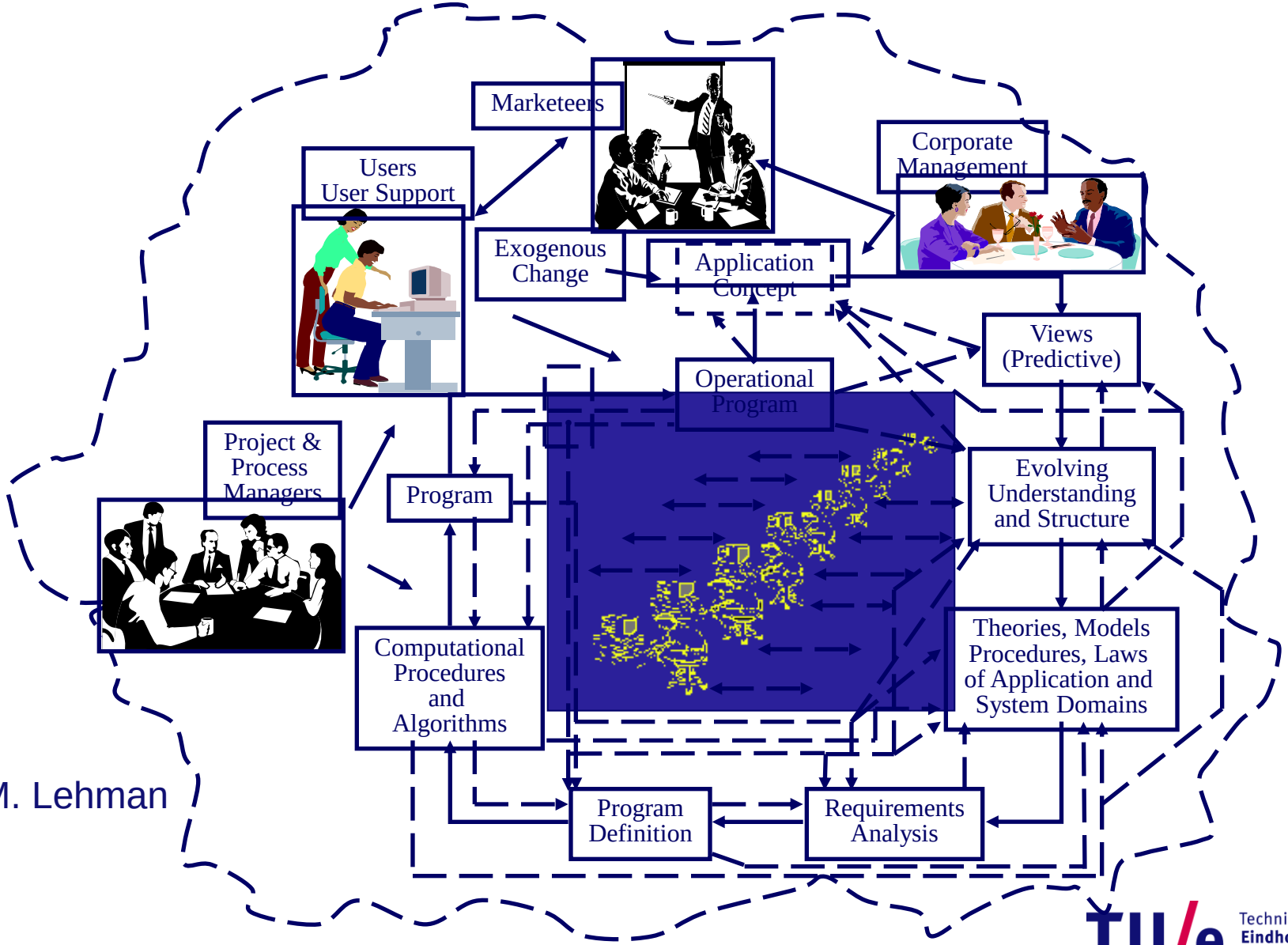
- **E-type evolution processes are multi-level multi-loop multi-agent feedback systems.**



- **This process model is far too simplistic!**

- **Agents: corporate managers, process managers, product managers, SW architects, SW developers, SW testers, marketers, users, …**
- **Loops: document/code reviews**
- **Levels: granularity, parallel versions, …**

In fact…



M.M. Lehman

Laws of Software Evolution: Summary

1. Continuing change
2. Increasing complexity
3. Self-regulation
4. Conservation of organizational stability
5. Conservation of familiarity
6. Continuing growth
7. Declining quality
8. Feedback system

Laws of SW Evolution: Limitations and critique

- Applicability to “non-standard” software?
  - Open source, co-developed, based on dev. frameworks
- Applicability to “non-standard” languages?
  - Specification languages, process models, domain-specific languages
- Applicability to “non-standard” artefacts?
  - Requirements documents, architecture, test scripts?
- Empirical validation
  - Case studies conducted and more case studies needed
  - Statistical validity of the results?!
- Vagueness

Summary so far…

- Software usually undergoes numerous small changes ⇒ evolution
- What, why and how of software evolution
  - Why: types of maintenance
  - How: types of software systems:
    - S-type: fixed spec, no evolution
    - P-type: fixed paradigm, restricted evolution
    - E-type: the general case
      - Lehman's laws of software evolution

Topics for the coming lectures

- Requirements Evolution
- Reverse Engineering and Evolution of SW Architecture
- Code Duplication and Differencing
- Mining Software Repositories
- Code Measurement: Size and Complexity
- Controlling Evolution: Refactoring and Reengineering

- Is there a topic not listed you are interested in?
  - Please let me know!

Software Evolution @ TU/e

- Software metrics, aggregation, repository mining and statistical analysis – Alexander Serebrenik
- Aggregation, mining, developers' specialization, multi-linguals systems – Bogdan Vasilescu
- Mining, traceability – Joost Gabriels
- Evolution of Eclipse plugins – John Businge
- Reverse engineering, tooling – Sergey Dubinov
- Tooling, case studies – Yenia Post
- Requirements – Martijn Klabbers
- Automotive software architecture – Yanja Dajsuren
- All of the above – Mark van den Brand

and also YOU!