2IS55 Software Evolution

Software Evolution

Alexander Serebrenik

TU/e Technische Universiteit Eindhoven University of Technology

Where innovation starts

Organisation

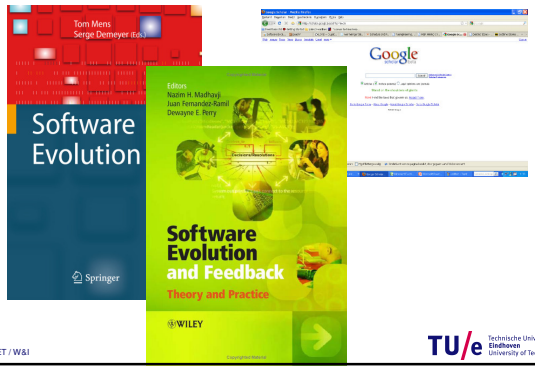
- Quartile 3 and 4:
  - 2 hours lectures:
    - Monday:
      - Time: 1th and 2th hour (8:45-10:30)
      - Location: Matrix 1.44
- Master students: CSE, ES, BIS
- 5 ECTS = 140 hours
  - 140 – 2*16 = 108 hours
- a.serebrenik@tue.nl
  - 3595
  - HG 5.41

Assignments

- http://peach.win.tue.nl/
- 8 assignments (on average: 2 weeks to complete)
- Some individual, some in small teams

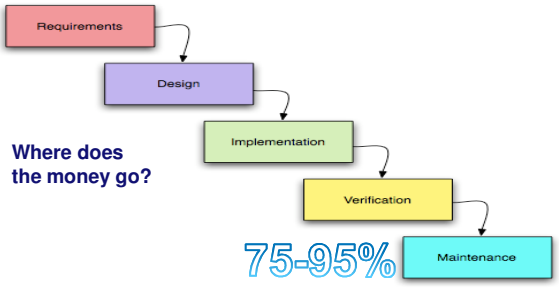
Number	1	2	3	4	5	6	7	8
Quartile	3					4		
Weight	0.05	0.1	0.1	0.1	0.15	0.15	0.15	0.2

Reading material (mostly…)



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: accumulation of changes through succeeding generations of organisms that results in the emergence of new species [Wikipedia]

Charles Darwin

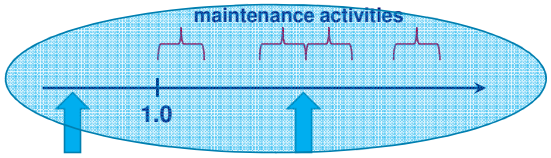
Meir Manny Lehman

What do you think about software evolution?

- What are our organisms?
- What are our generations?

Evolution vs. Maintenance

Software evolution



"Software development" (Royce 1970)

"Software maintenance" (Royce 1970)

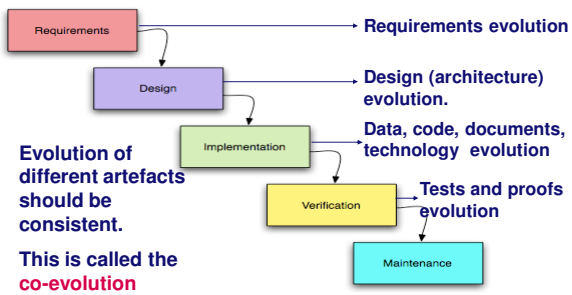
Why do we want to study software evolution?

- 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?

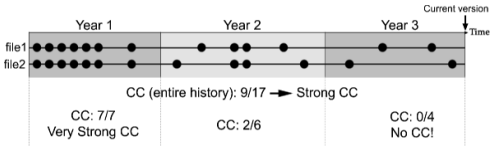


Evolution of different artefacts should be consistent.

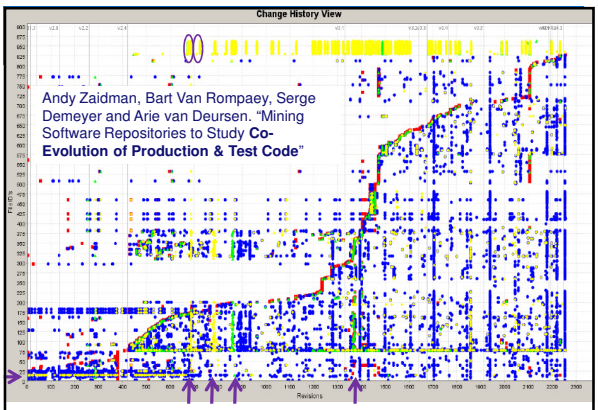
This is called the co-evolution problem.

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"



Andy Zaidman, Bart Van Rompaey, Serge Demeyer and Arie van Deursen. "Mining Software Repositories to Study **Co-Evolution of Production & Test Code**"

Added Source Modified Source Added Test Modified Test

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?

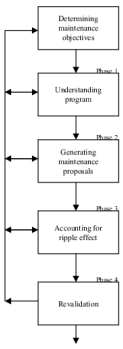
Swanson 1976

- 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?

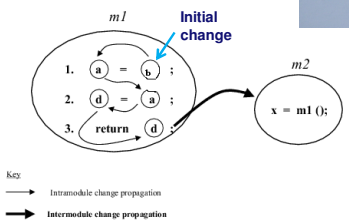
How do the system evolve?

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



Ripple effect?

- One change causes another one
- More and more changes



- Evolution in large: behaviour emerging from millions of changes

But does all software evolve?

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

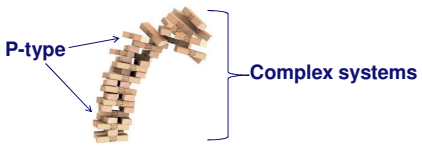
- Rarely observed in practice: Why?
- Important:
  - Some design approaches aim at improving formality
  - But implicitly assume absence of evolution
    - S-type systems definition clarifies shortcomings of such approaches (Acme ADL, …)
- Are S-type systems useless in practice?

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?

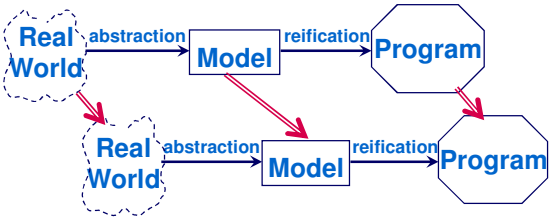
P-type systems and reuse

- 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:



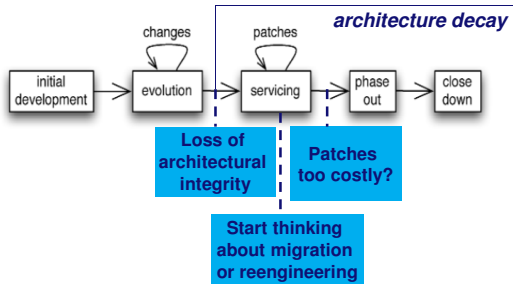
E-type systems

- "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?

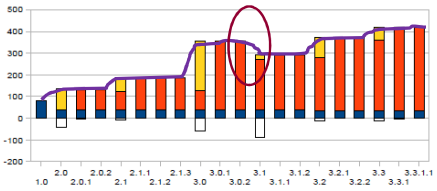
Step aside: is the continuous change always possible?



- 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.

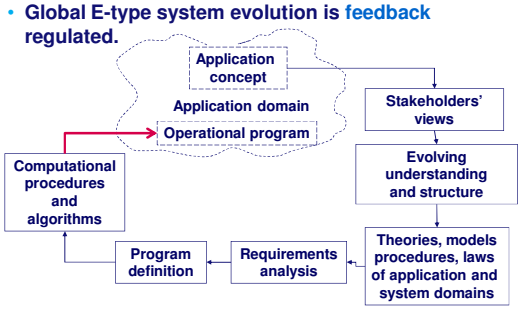
Wermelinger, Yu, Lozano, ICSM 2008

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

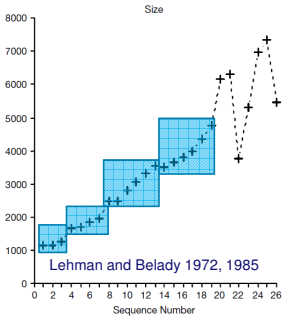
- Global E-type system evolution is feedback regulated.



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

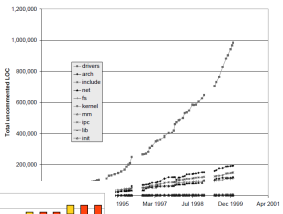


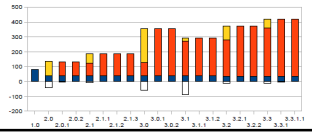
4. Conservation of Organizational Stability

- 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: the growth is linear.
- Godfrey and Tu: not for Open Source (Linux)
- WYL: linear for architecture (Eclipse)

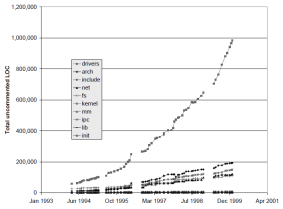




2000

6. Continuing Growth

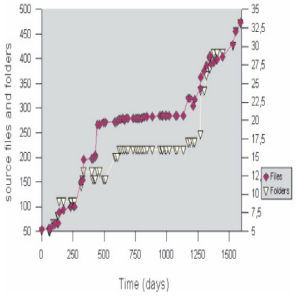
- The functional capability of E-type systems must be continually enhanced to maintain user satisfaction over system lifetime.
- In earlier versions the law talked about "size".



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?

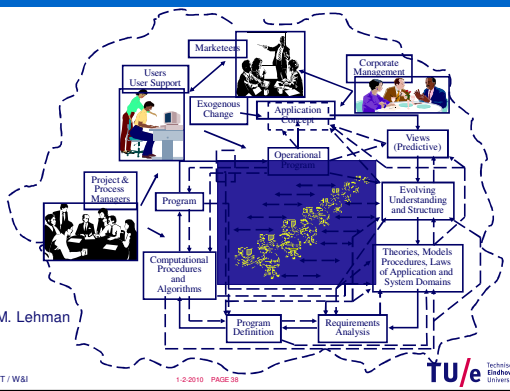
8. Feedback system

- 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, DSL
- 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

Assignment 1

- Individual
- Deadline: February 8, 23:59

- Think of examples of software systems with components of/close to pure S-, P- and E-type.
- Briefly describe the systems and the components.
- Explain why do you think that these components are S-, P- or E-type.

- Your description should not exceed 2 A4 pages.

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
- Humans and Evolving Software
- Co-evolution and Inconsistencies
- Controlling Evolution: Refactoring, Reengineering and Model Transformations

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

Software Evolution @ TU/e

- What does evolve?
  - Repositories for models – Zvezdan Protic, MvdB
- What can we learn from evolution so far?
  - Statistical approaches – Serguei Roubtsov, AS, MvdB
- How evolvable are our systems?
  - SQuAVisiT – Serguei Roubtsov, Martin vd Vlist, Peter Schachtschabel
- How to make the artefacts more evolvable?
  - Reverse Engineering – Dennie van Zeeland, MvdB, AS
- How is the evolution implemented?
  - Model transformations – Marcel van Amstel, Luc Engelen, MvdB
  - Migration coordination – Erik de Vink, Suzana Andova

and also YOU!