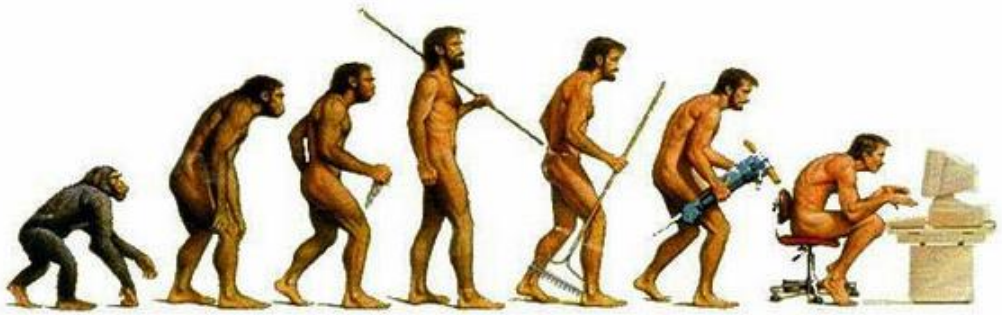


Software Evolution

Lecture 1: Introduction

Your teachers for this course:
Thomas van Binsbergen (coordinator),
Damian Frölich (teacher),
Georgia Samaritaki (teacher)

The plan for today

1. Course overview:
   - Review objectives,
   - Components & grading,
   - Schedule of lectures and assignments.
2. Introduction in the field of Software Evolution
3. A more detailed look at the assignments
4. Rascal / Series 0 Introduction

Motivation

- Software Evolution
  - a course about studying existing software
  - Why?
- Software
  - A lot of software exists
  - Impacts our daily lives
  - Continuous change to software and to daily life in which it is used

Some quotes:

In 2020 only 30% software projects are “new”, the rest is “maintenance.”
(source: Software Productivity Research)

0.1 - 10 bugs per 1,000 lines of code.
Exercise: Who says this, is it true?

If you start working as a software engineer, the chance of having to work with existing software exceeds the chance of starting from scratch!

Objectives

Exit qualification: *"The graduate masters the methods and techniques needed to analyze an existing software system and to enable it to evolve given changing requirements."*

Our objectives are three-fold:

1. Acquire an understanding and appreciation of the challenges posed by *software maintenance* and *software evolution*.
2. Learn about *quality* of architecture and source code and how it affects software maintenance and evolution.
3. Select and also construct *software analysis* and *software transformation* tools to help obtain insight in software quality and to help improve software quality.

Course Goals and Activities

- Objectives
  - **Master concepts.** Learn basic concepts and "laws" of software evolution
  - **Analyze code.** Learn how to measure or detect evolution in real software (i.e., code)
  - **Build tools.** Learn building tools that help maintaining existing software
  - **Read, judge, analyze, synthesize, write.** Learn to do scientific reading and reporting
- Approach
  - **Reading.** Study selected papers and book chapters
  - **Attend Lectures.** Introduce a topic and discuss
  - **Writing.** Write an annotated bibliography on the papers you read
  - **Practical Lab.** Build tools for software analysis

Teachers and guest lecturers

Thomas van Binsbergen

Damian Frölich

Georgia Samaritaki

Martin Bor

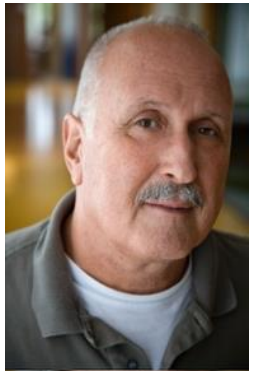
Tijs van der Storm

Magiel Bruntink

Vadim Zaytsev

Lina Ochoa Venegas

Kudos for developing and maintaining this course



Paul Klint

Jurgen Vinju

Magiel Bruntink

Vadim Zaytsev

Thomas <–> Software Evolution

- BSc/MSc Computer Science at Utrecht University
  - Advanced functional programming, compiler construction
  - Thesis: Scheduling of Linear Ordered Attribute Grammar

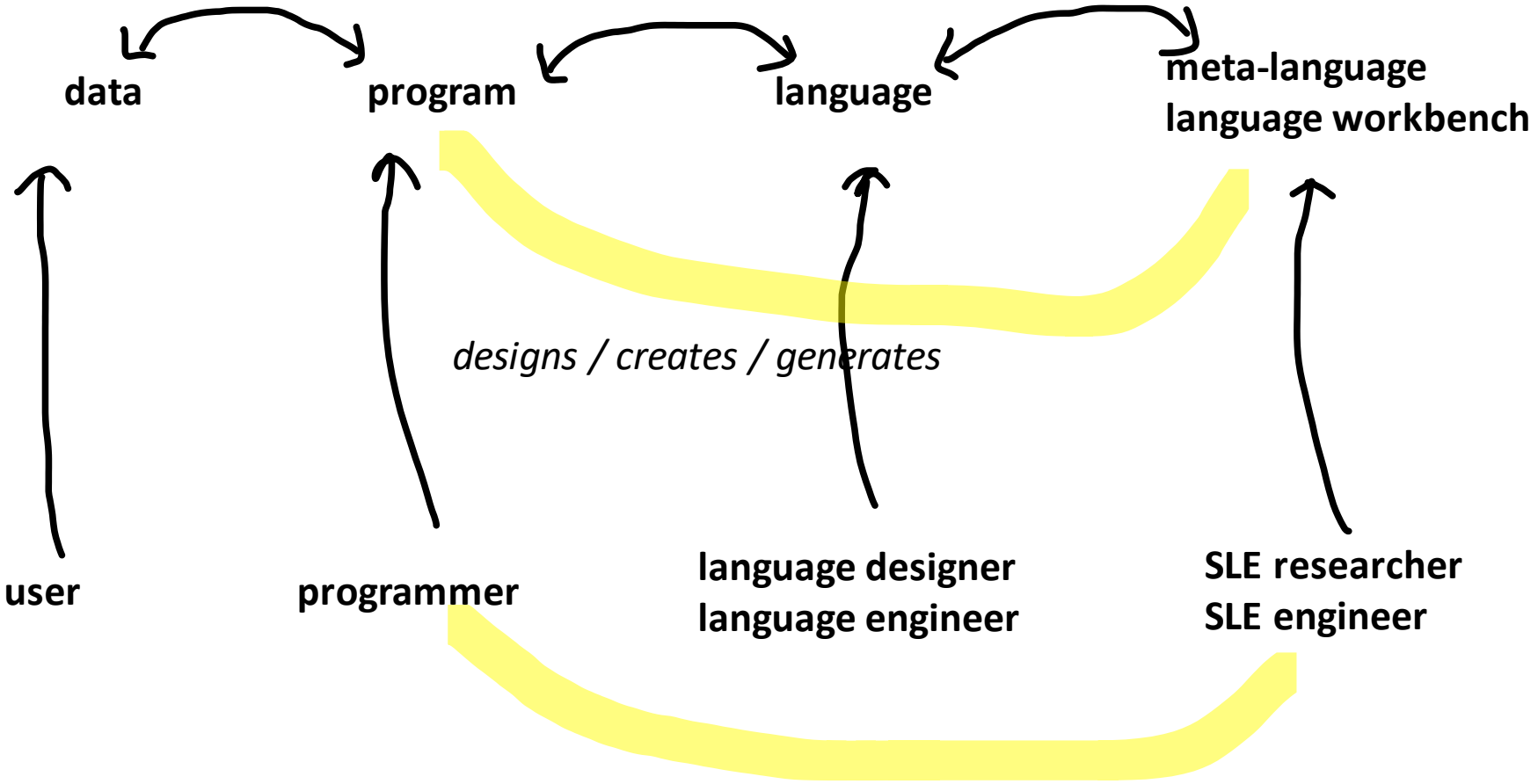
- PhD at Royal Holloway, University of London
  - Generalised top-down (GLL) parsing and parser combinators
  - Fundamental Constructs (funcons) of programming languages
  - Thesis: Executable Formal Specification of Programming Languages with Reusable Components

- Postdoc at CWI Software Analysis & Transformation (SWAT) group
  - Designing and implementing a DSL for normative specifications (laws, regulations, contracts) -- eFLINT
  - Study the foundations of incremental (REPL interpreters) and exploratory programming (computational notebooks)

- Assistant Professor at UvA Complex Cyber Infrastructure group
  - Specification and configuration of norm-aware data exchange systems
  - DSLs to manage the complexity of (virtual) network infrastructures
  - Techniques for reuse and composition in language definition and implementation

eflint

Software (language) engineering

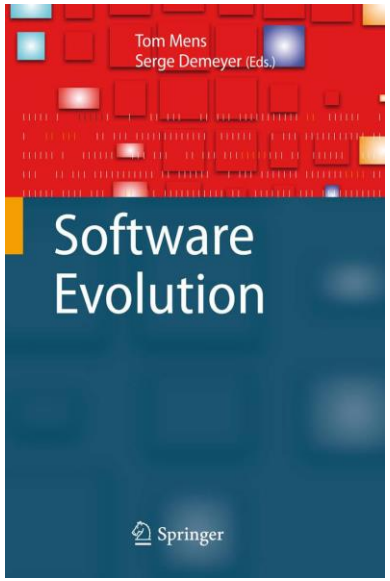


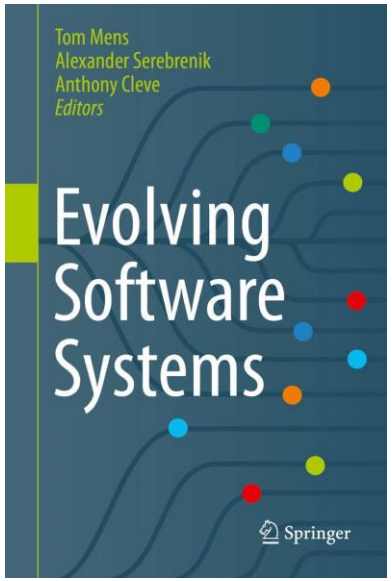
Where are you in this picture?

And this course?

Software Evolution: Overview

- Software cloning and duplication
- Defect, failure and bug prediction
- Re-engineering, restructuring and refactoring
- Database evolution
- Migration of legacy systems
- Service oriented architectures
- Testing
- Service-oriented software development
- Software architecture evolution
- Reverse-engineering, program understanding, program comprehension
- Incremental change, impact analysis, change propagation
- Evolution process
- Software visualization
- Component-based software development
- Open source software
- Software repository mining
- Software transformation, model transformation, graph transformation
- Model evolution, meta-model evolution
- Software measurement, software quality
- Software requirements
- Search based software engineering
- Socio-technical networks, Web 2.0
- Web-based systems
- Dynamic adaptation run-time evolution
- Software product line engineering
- Software ecosystems, biological evolution





Software Evolution. Editors: Tom Mens and Serge Demeyer. Springer, 2008

Evolving Software Systems. Editors: Tom Mens, Alexander Serebrenik, Anthony Cleve. Springer 2014

Journals and Conferences

- ACM
  - TOPLAS: Transactions on Programming Languages and Systems
  - TOSEM: Transactions on Software Engineering and Methodology
  - TWEB: Transactions on the Web
- Elsevier
  - JSS: Journal on Systems and Software
- Inderscience
  - IJWET: International Journal of Web Engineering and Technology
- IEEE
  - TSE: Transactions on Software Engineering
- Kluwer
  - ASE: Automated Software Engineering
- Rinton Press
  - JWE: Journal of Web Engineering
- Springer
  - SoSyM: Software and Systems Modeling
  - EMSE: Empirical Software Engineering
- Wiley
  - SPE: Software: Practice and Experience

- BENEVOL: Belgium-Netherlands Software Evolution Workshop
- CSMR: European Conference on Software Maintenance and Reengineering
- ICPC: International Conference on Program Comprehension
- ICSM: International Conference on Software Maintenance
- SCAM: International Working Conference on Source Code Analysis & Manipulation
- SEAMS: International Conference on Software Engineering for Adaptive and self-Managing Systems
- SLE: International Conference on Software Language Engineering
- MSR: Working Conference on Mining Repositories
- WCRE: Working Conference on Reverse Engineering

Why all these reading suggestions?

Course overview

- Two full-day semi-formal sessions 9-17 Monday and Tuesday
  - Monday (guest) lecture 11-13 (occasional Tuesday)
  - Grading/presentation sessions: Week 4 and Week 8
  - 6 ETCS / 8 weeks ~= 20 p/week

- Guided self-study and group-study
  - Ask each other questions and engage in discussions (in lab or slack)
    - Share ideas, not solutions!
  - Suggests interesting papers, projects, code bases, etc.

- Where to go with questions?
  1. Canvas and the course Reader
  2. Ask a colleague during lab (or on Slack)
  3. Ask teacher during lab (or on Slack)
  4. E-mail me: l.t.vanbinsbergen@uva.nl
  5. Rascal bugs? See course Reader

Course Schedule: Lectures

w	topic	lecturer
1a	Introduction to Software Evolution	Thomas van Binsbergen
1b	Meta-Programming and Rascal	Tijs van der Storm
2	Software Metrics – Software Improvement Group	Magiel Bruntink
3	Clone Detection and Management	Damian Frölich
4	Semantics and Equality	Thomas van Binsbergen
5	Legacy Software and Renovation – Raincode	Vadim Zaytsev
6	Breaking Changes	Lina Ochoa Venegas
7	No lecture	--
8	Series 2 presentations	You

a: Monday 11:00, b: Tuesday 11:00

Course Schedule: Assignments

w	Reading	Practical Lab	Deadlines
1	Klint et al. 2009, Klint et al. 2019, Mens 2008, Herraiz et al. 2013	Series 0 and 1	Series 0
2	Heitlager et al. 2007, Baggen et al.2012	Series 1	
3	Koschke 2008 Kapser and Godfrey 2006	Series 1	Series 1 (Sunday 23:59)
4	(see Reader)	Series 2 Grading sess.	
5	(see Reader)	Series 2	
6	(see Reader)	Series 2	
7	no new reading	Series 2	Series 2 (Sunday 23:59)
8	no new reading	Presentations	Ann Bibliography (Friday 23:59)

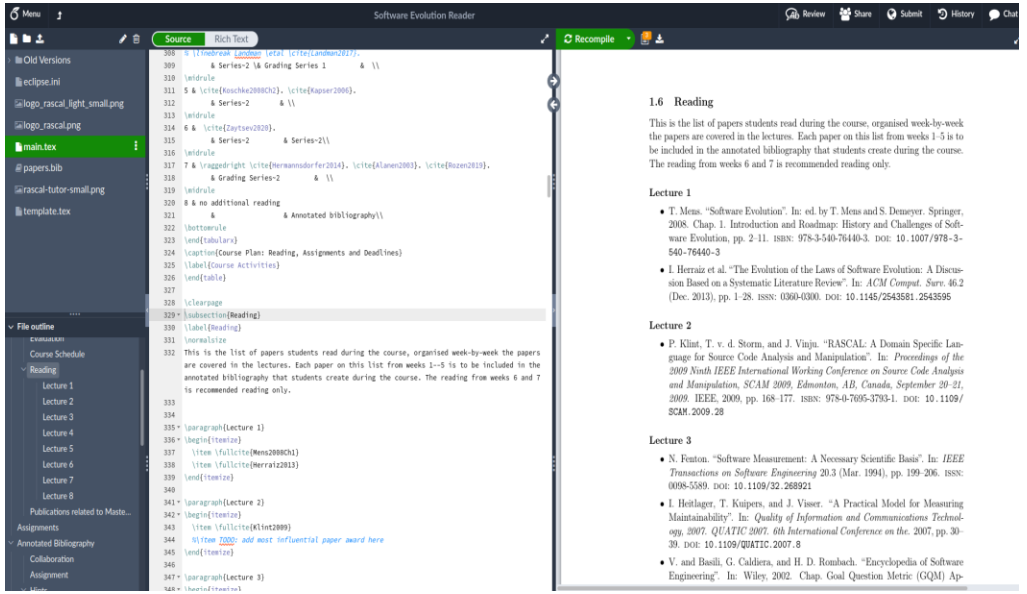
Assignments and Grading

- Series 0: Rascal Basics
  - Introductory level
  - Individual
  - mandatory but not graded

- Annotated Bibliography
  - Academic exercise
  - Individual
  - 1/3 of your grade

- Series 1: Software Metrics
  - Intermediate level
  - In teams of two
  - 1/3 of your grade

- Series 2: Clone Detection
  - Advanced level
  - In teams of two
  - 1/3 of your grade



The Plan for Today

1. Course overview
2. Introduction in the field of Software Evolution
3. A more detailed look at the assignments
4. Rascal / Series 0 Introduction

Part of this Week's Reading

- T. Mens, *Introduction and Roadmap: History and Challenges of Software Evolution*, in *S*oftware Evolution, Springer, 2008.

- Herraiz, et al. The Evolution of the Laws of Software Evolution: A Discussion Based on a Systematic Literature Review. ACM Comput. Surv., 46(2):28:1– 28:28, December 2013.

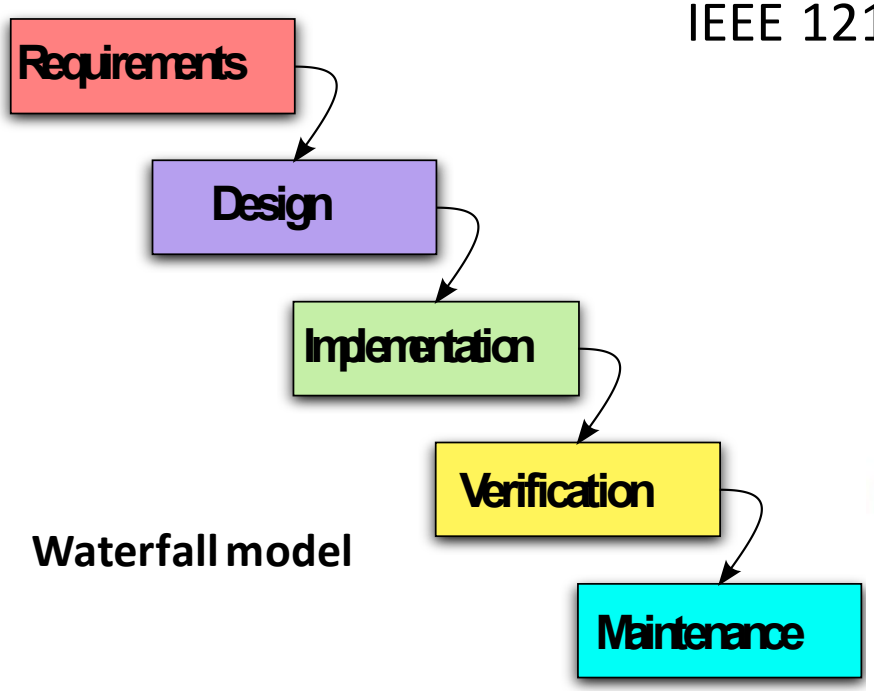
Why does software evolve?

?

Maintenance vs Evolution

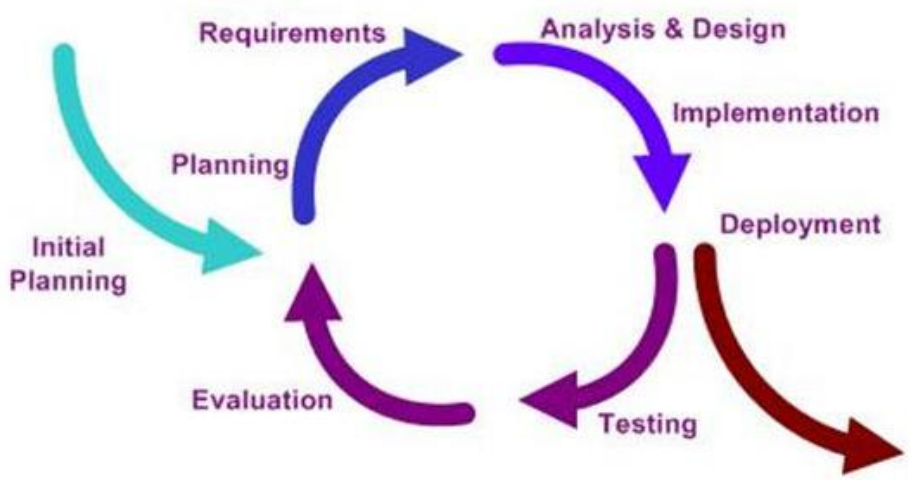
- *software maintenance:* Modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment

Standard for Software Maintenance
IEEE 1219, 1993



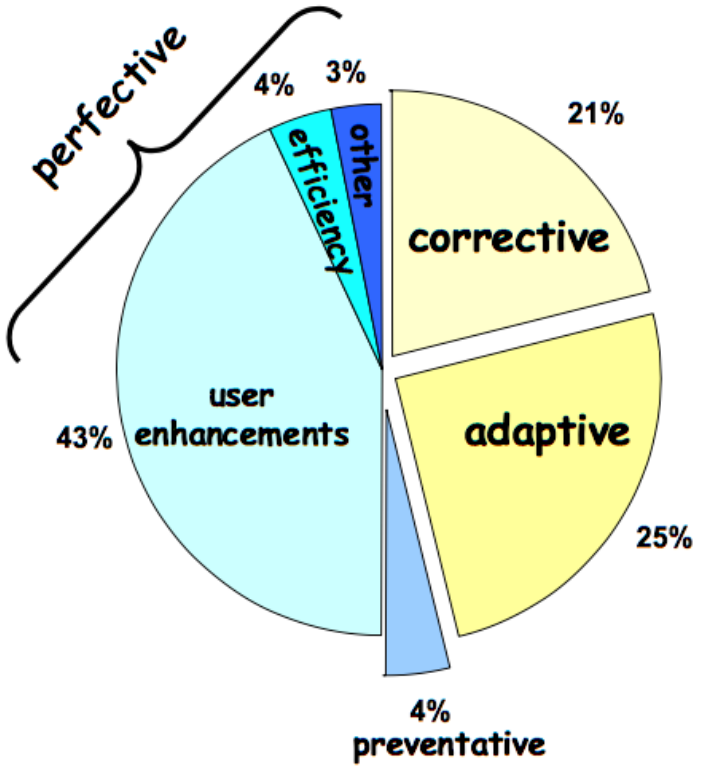
Waterfall model

Iterative / Agile



Maintenance Categories

- ***Corrective*** – diagnosing and fixing errors, possibly ones found by users
- ***Adaptive*** – modifying the system to cope with changes in the software environment
- ***Perfective*** – implementing new or changed user requirements which concern functional enhancements to the software
- ***Preventive*** – increasing software maintainability or reliability to prevent problems in the future

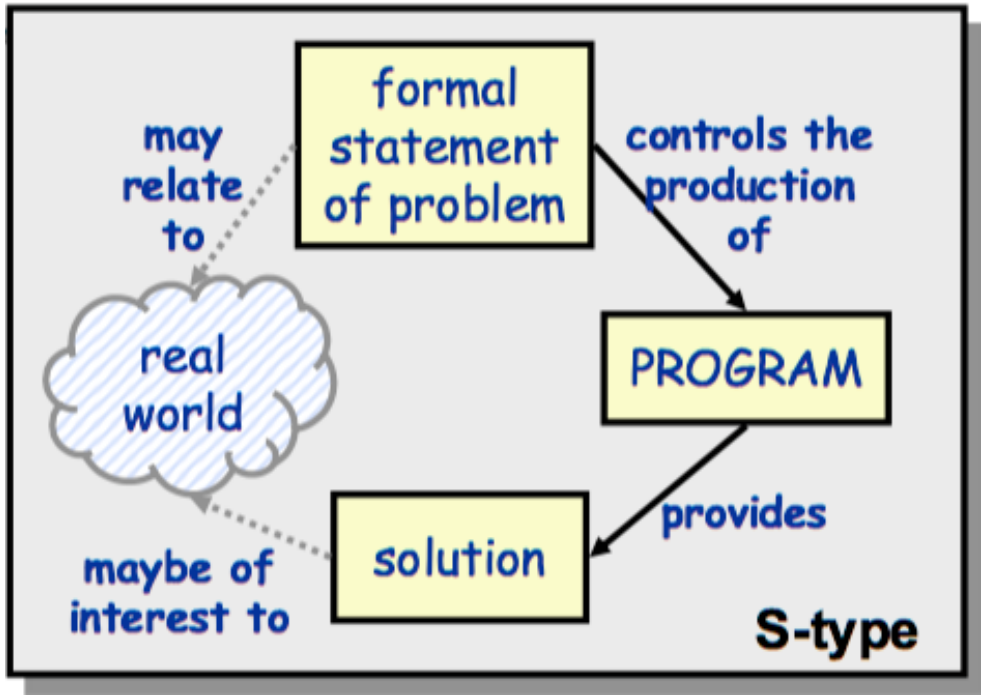


B.P.Lientz, E.B.Swanson, *Software Maintenance Management, A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations*, 1980.

Steve Easterbrook, Software Maintenance. http://www.cs.toronto.edu/~sme/CSC444F/slides/L20-SoftwareMaintenance.pdf

S-Type Programs “Specifiable”

- Problem formally defined by a specification
- Automated acceptance is possible:
  “Is the program correct according to its specification?”
- The software does not evolve because a change to the specification is a new program

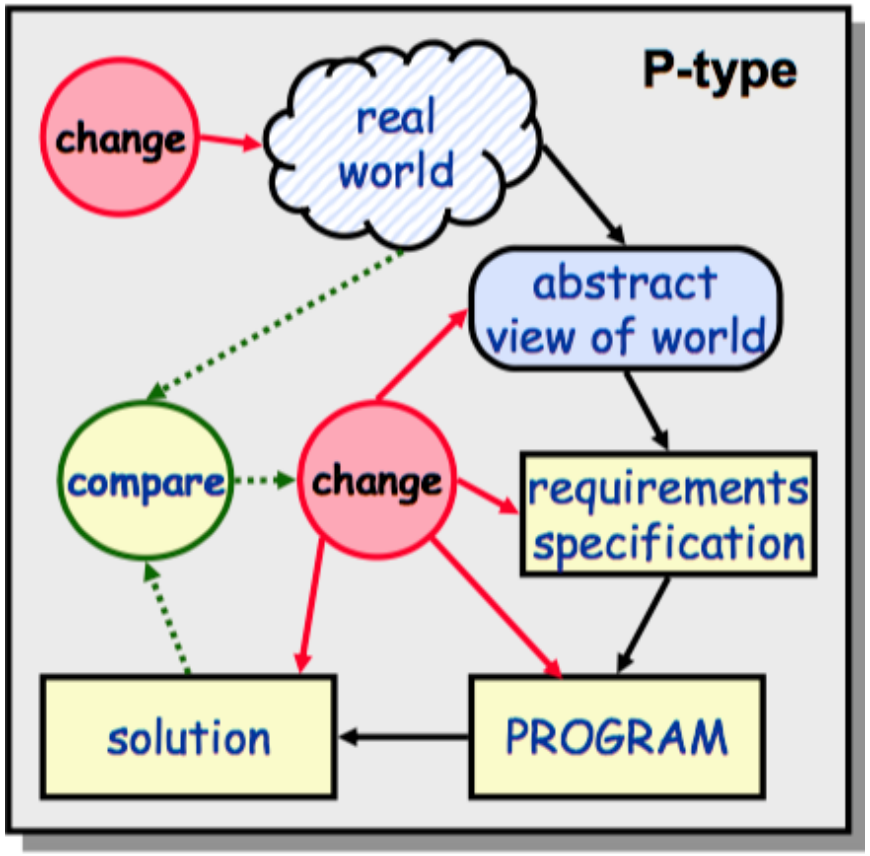


M.M.Lehman, *Programs, Life Cycles and Laws of Software Evolution*, IEEE 68(9), 1980.

Steve Easterbrook, Software Maintenance. http://www.cs.toronto.edu/~sme/CSC444F/slides/L20-SoftwareMaintenance.pdf

P-type programs “Problem Solving”

- Program models a real-world task imperfectly
- Qualitative acceptance:
  “Is the program an acceptable solution to the problem?”
- Software is likely to evolve continuously
  - Because the solution is never perfect, and can be improved
  - Because the real-world changes and hence the problem changes

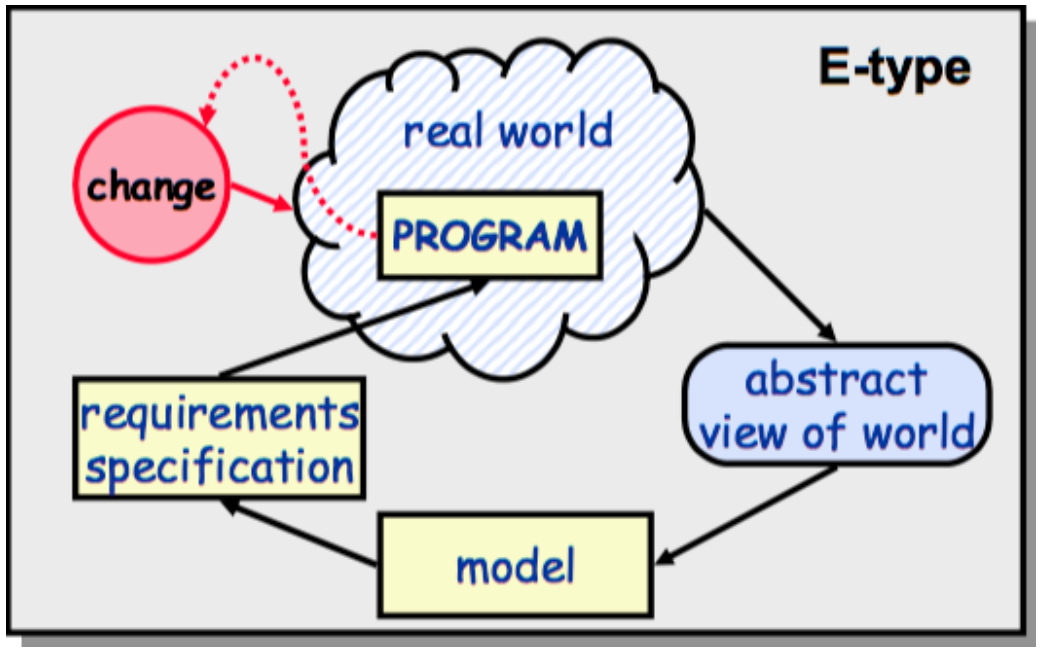


M.M.Lehman, *Programs, Life Cycles and Laws of Software Evolution*, IEEE 68(9), 1980.

Steve Easterbrook, Software Maintenance. http://www.cs.toronto.edu/~sme/CSC444F/slides/L20-SoftwareMaintenance.pdf

E-Type Programs “Embedded”

- Program forms a solution that is part of the real world and affects the real world
- Acceptance requires prediction and judgement about program effects
- Such systems are <u>inherently evolutionary</u> because changes to the software and the world affect each other



M.M.Lehman, *Programs, Life Cycles and Laws of Software Evolution*, IEEE 68(9), 1980.

Steve Easterbrook, Software Maintenance. http://www.cs.toronto.edu/~sme/CSC444F/slides/L20-SoftwareMaintenance.pdf

The Laws Of Software Evolution

- First formulated by two researchers, Lehman and Belady, in 1976.

- Based on a case study of IBM OS/360 they formulated 3 laws:
  1. **continuing change:**
     "Software systems must be continuously changed to adapt to the environment."
  2. **increasing entropy:**
     "Changes increase the complexity of software."
  3. **smooth growth:**
     "Software evolution can be studied using statistical methods."

- The laws have been studied, validated and changed over the years: now there are 8 laws.

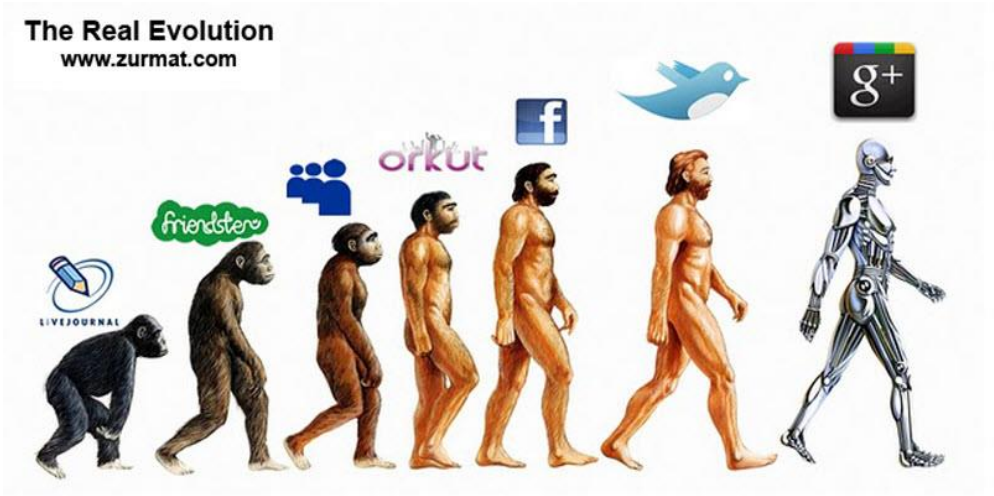
- In order to study software evolution we require a categorization of software -> SPE scheme

Lehman's Laws (1/8)

I. Law of Continuing Change

- E-type system rots,
  unless adapted it becomes progressively less useful
- the process never stops
  (until the system is replaced)

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



M.M.Lehman, *Programs, Life Cycles and Laws of Software Evolution*, IEEE 68(9), 1980.

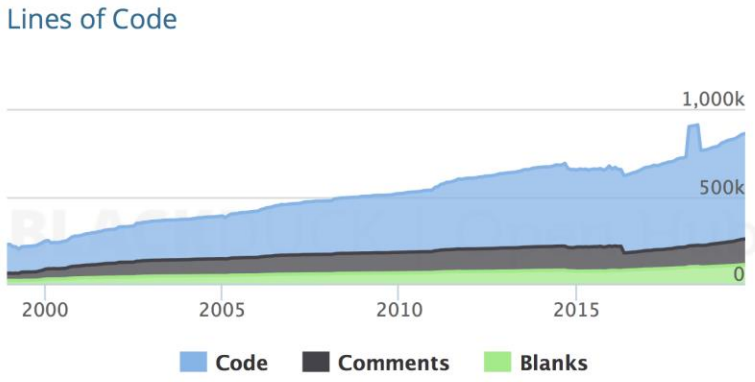
Lehman's Laws (2/8)

II. Law of Increasing Complexity

- as an E-system evolves it becomes more complex
- changing means complicating
- (unless mitigated by changes/adaptations)

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

OpenSSL on BlackDuck OpenHub:



M.M.Lehman, *Programs, Life Cycles and Laws of Software Evolution*, IEEE 68(9), 1980.

Lehman's Laws (3/8)

III. Fundamental Law of Program Evolution Software evolution

- "Software evolution is self-regulating with statistically determinable trends."

III. Law of self regulation

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

- Note
  - The 3rd law changed a few times… 1974, 1980, 1996
  - partially empirically validated

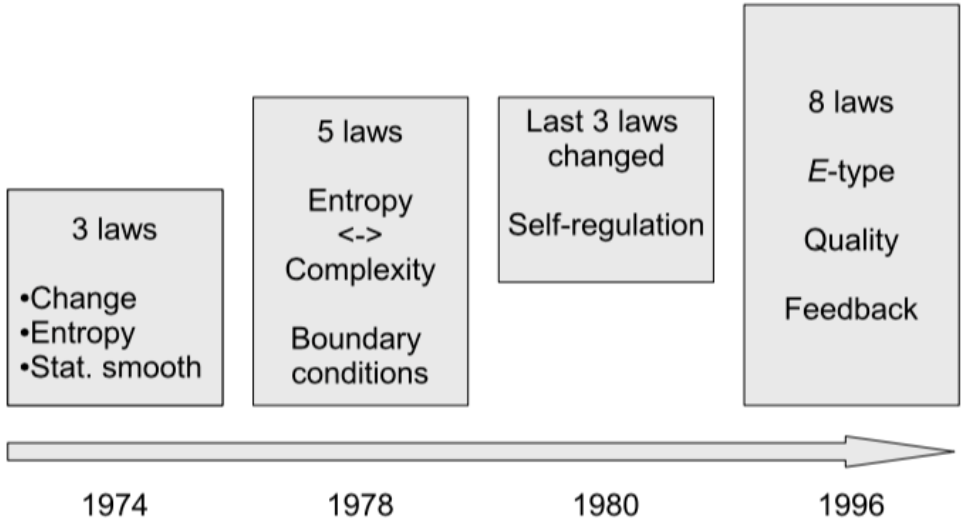


Diagram of the evolution of the laws of software evolution from Herraiz et al.

Herraiz, et al. The Evolution of the Laws of Software Evolution. ACM Comput. Surv., 46(2):28:1– 28:28, December 2013.

Lehman's Laws (4/8)

IV. Law of Conservation of Organizational Stability

- E-system development activity is invariant
- throughout lifetime
- (does not depend on resources / manpower)

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

OpenSSL on BlackDuck OpenHub:

M.M.Lehman, *Programs, Life Cycles and Laws of Software Evolution*, IEEE 68(9), 1980.

Lehman's Laws (5/8)

V. Law of Conservation of Familiarity

- E-system changes per release is invariant
- throughout its lifetime
- (too little: bored; too much: overwhelmed)

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

- Can be viewed as a more specific version of Law IV on the Conservation of Organizational Stability.

M.M.Lehman, *Programs, Life Cycles and Laws of Software Evolution*, IEEE 68(9), 1980.

Lehman's Laws (6/8)

VI. Law of Continuing Growth

- E-system must add features over time
- to keep users satisfied
- (expectations creep)

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

- Can be viewed as a more specific version of Law I on Continuing Change.

M.M.Lehman, J.F.Ramil, P.D.Wernick, D.E.Perry, W.M.Turski, Metrics and Laws of Software Evolution — The Nineties View, METRICS, 1997.

Lehman's Laws (7/8)

VII. Law of Declining Quality

- E-system perceived quality declined
- internal as well as external
- (unless constantly maintained)

"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."

- Is a corollary to Laws I, VI and II
  - Because a system must be adapted continuously, and because changes result in complexity, it follows that quality will decrease / less satisfaction / harder to evolve

M.M.Lehman, J.F.Ramil, P.D.Wernick, D.E.Perry, W.M.Turski, Metrics and Laws of Software Evolution — The Nineties View, METRICS, 1997.

Lehman's Laws (8/8)

VIII. Law of Feedback System

- E-system evolution processes are feedback systems
  - multi-level
  - multi-loop
  - multi-agent

- Can be viewed as a more elaborate version of Law III on Self Regulation.

M.M.Lehman, J.F.Ramil, P.D.Wernick, D.E.Perry, W.M.Turski, Metrics and Laws of Software Evolution — The Nineties View, METRICS, 1997.

Main conclusions Herraiz et al.

- On the applicability of the laws
  - Significant evidence only for I and VI
    - (continuous change and growth)
  - Significant counter-evidence for II
    - (increasing complexity)
    - by implication partly also VII
  - Other laws:
    - Not definitively confirmed or denied
    - Significant differences for libre vs nonlibre software
- On the evolution of the laws
  - Only published modifications are by Lehman and collaborators
  - No modifications after 1996
  - Empirical validation first attempted in late 90s, with mixed results since

•Herraiz, et al. The Evolution of the Laws of Software Evolution: A Discussion Based on a Systematic Literature Review. ACM Comput. Surv., 46(2):28:1– 28:28, December 2013.

Exercise

- Pair up with your neighbour for 5 minutes:
  1. Discuss examples related to the software evolution laws from your own experience.
  2. Select the 1 or 2 most compelling examples.

- Discuss as a group

The plan for today

1. Course overview:
2. Introduction in the field of Software Evolution
3. A more detailed look at the assignments
4. Rascal / Series 0 Introduction

Software Evolution – Reader

- The course reader gives a detailed overview on reading and activities.
  - Please check Canvas for updates regularly

- Let’s have a look at the assignments

Assignment: Annotated Bibliography

- For each paper on the reading list
  - Write an concise discussion (2-4 coherent paragraphs in your own words) of the major points of the paper.
  - Strict format to use and page limit
  - See detailed assignment on Canvas.

- Mandatory papers and paper 'tracks' are listed in the Reader
  - Choice 1: Metrics
  - Choice 2: Software language engineering *(pick 1 of 2 tracks)*

- Ask for feedback during lab sessions.

- The annotated bibliography should be delivered in week 8 at Friday.

Assignment: Lab series 0

Goal: Learn to work with Rascal

- Documentation: https://new.rascal-mpl.org/docs/GettingStarted/

Collaboration:

Individual exercises to improve your programming skills.

Assignment:

Available on Canvas as a Module

Grading:

This series is not graded. The goal is to prepare for series 1 and 2.

Deadline:

You should finish Series 0 in the 1st week of the course.

Assignment: Lab series 1

Goal:

Build a tool for calculating software metrics
and the SIG maintainability model.

Collaboration:

In pairs, compose yourself.

Assignment, grading:

Detailed description in the Reader.
Grading is based on written report and demonstration during a Grading Session

Deadline:

The deadline for Series 1 is week 3 of the course (Sunday)
Grading Sessions take place in week 4 during lab hours

Assignment: Lab series 2

Goal:

Build a clone management tool. And be creative!

1. Back-end track: implement and compare clone detection algorithms
2. Front-end track: implement and evaluate maintenance visualizations of clones

Collaboration:

Same pair as Series 1

Assignment, grading:

Detailed description in the Reader.

Grading is based on written report and presentation

Deadline:

The deadline for Series 2 is week 7 of the course (Sunday)

Presentations during lab hours in week 8

Week 1 objectives

- By today: Study course Reader and figure out course expectations and planning
- By today: Installing Rascal and learning Rascal
  (material: Canvas documents + Rascal online documentation)

- By tomorrow: Form groups of two for executing Series 1 & 2
- By tomorrow: Ready Overleaf project for Annotated Bibliography
  - Create account and/or link with UvA address (maybe use Git)
  - Invite Martin and Thomas into your project

- By end of week: finish Series 0, material on Canvas
- By end of week: mandatory reading
- By end of week: start Series 1

The plan for today

1. Course overview:
2. Introduction in the field of Software Evolution
3. A more detailed look at the assignments
4. Rascal / Series 0 Introduction