

Introduction to Software Evolution

Olivier Liechti
HEIG-VD
olivier.liechti@heig-vd.ch



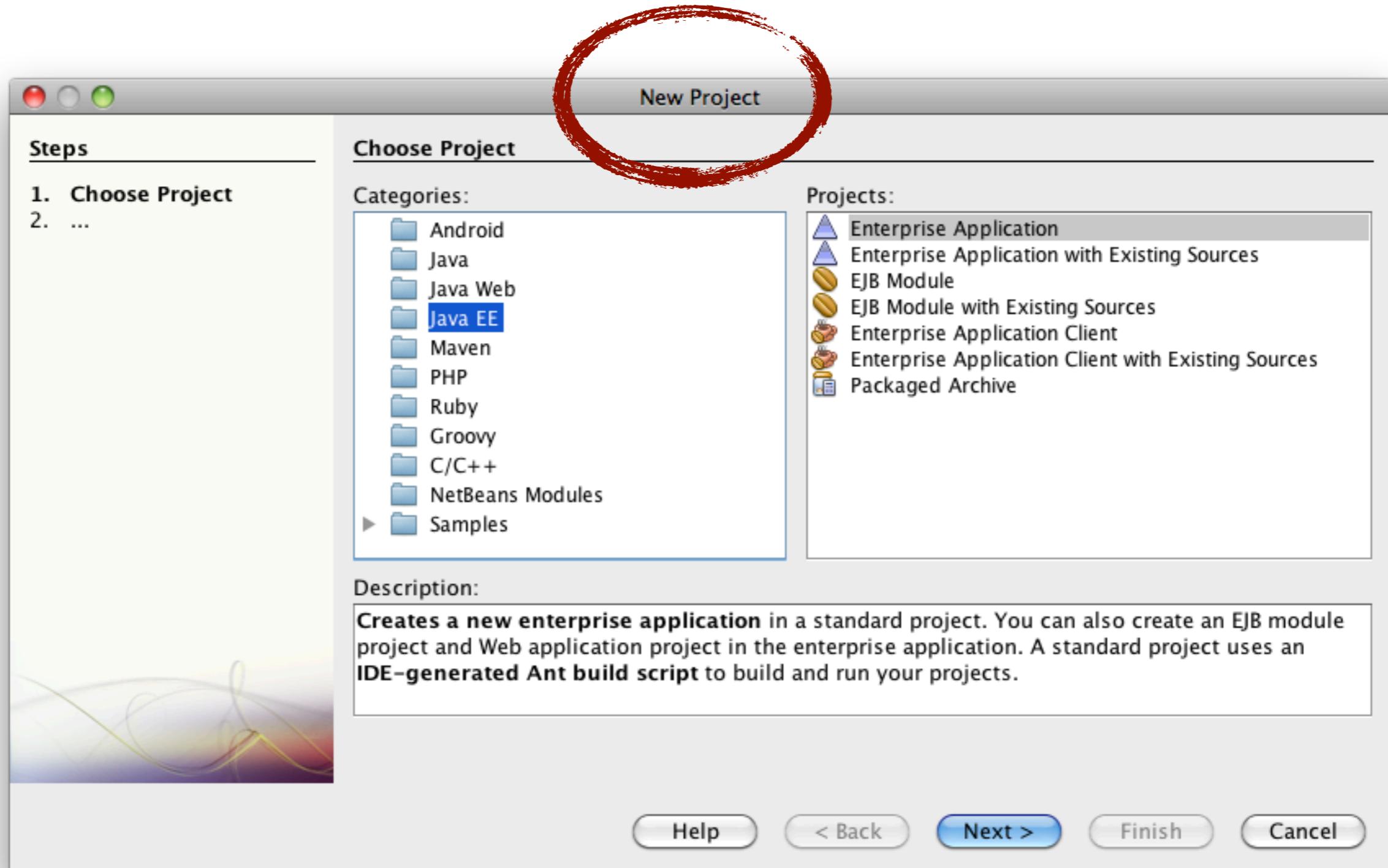
MASTER OF SCIENCE
IN ENGINEERING

RE-Planning

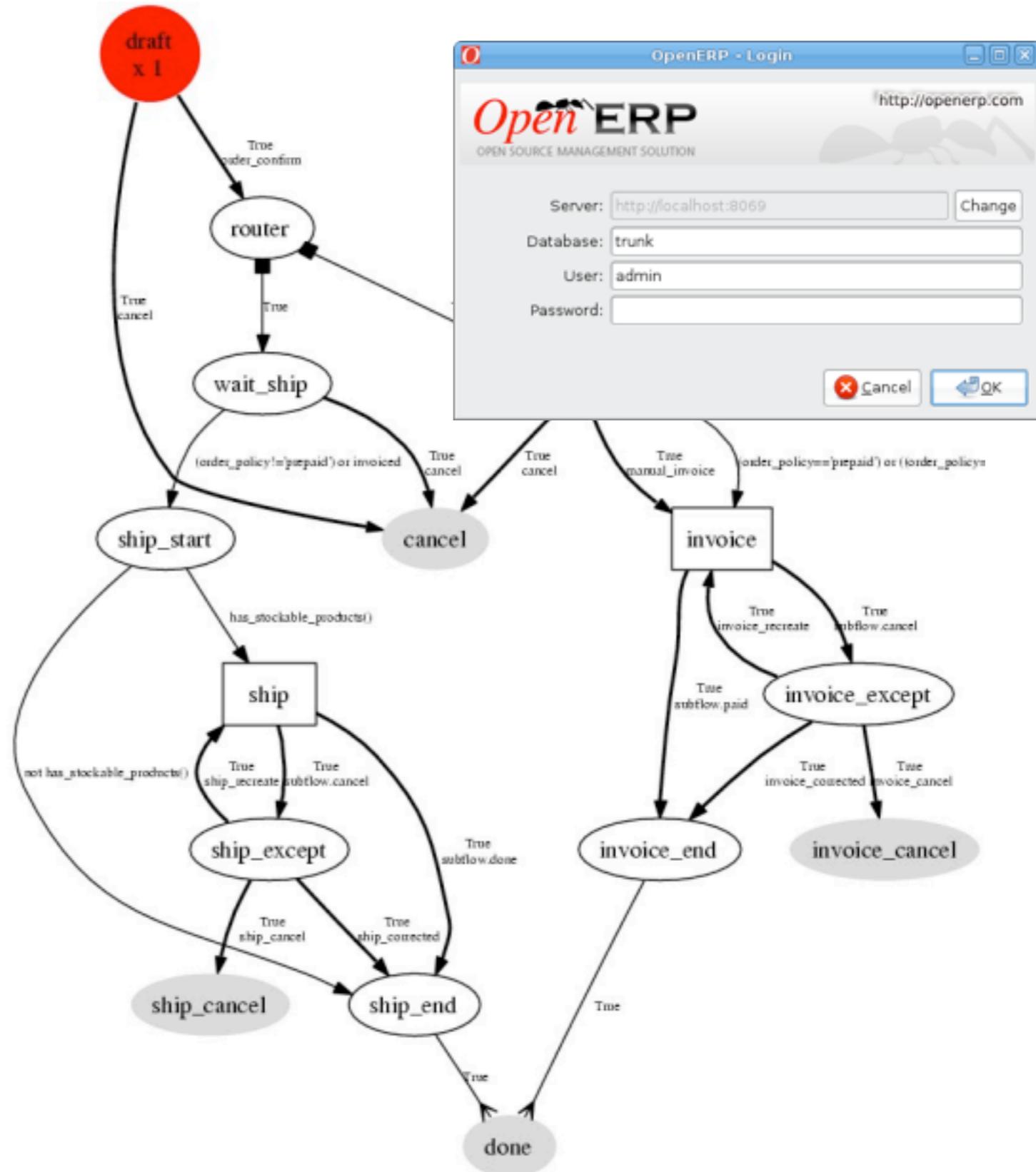
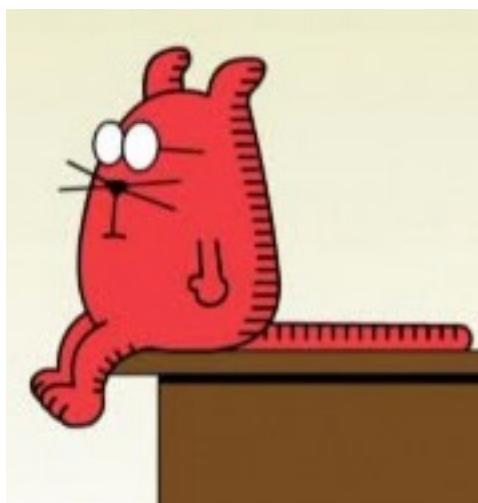
Date		Theory	Practice
20.09.12	agile	Introduction to agile & scrum	
27.09.12	agile	Agile development tools	Agile development tools
04.10.12	agile	User Stories	Guest speaker (Lotaris)
11.10.12	agile	The Product Owner	Presentations Agile Development Tools
18.10.12	agile	Guest speaker (Octo)	Guest speaker (Octo)

Date		Continuous Delivery	Lab
01.11.12	agile		
08.11.12	architecture	Behavior Driven Development (BDD)	BDD lab
15.11.12	architecture	Development in and for the Cloud	Guest speaker (Cloudbees)
22.11.12	architecture	BDD lab	BDD lab
29.11.12	evolution	Software Evolution	Guest speaker (SonarSource)
06.12.12	evolution	BDD lab	BDD lab
13.12.12	evolution	OO Reengineering	BDD lab
20.12.12	architecture	Design of (REST) APIs	BDD lab

10.01.13	architecture	BDD Presentations	BDD Presentations
17.01.13			
24.01.13			



I want to be able
to send e-invoices
to mobile phone
users!



How was the ERP system designed?

How can I extend the ERP?

Am I the first to do something like that?

Where is the documentation?

How can I be sure that I did not break anything?

How do I release my new feature?

Who has the knowledge?



Agenda

- **Introduction**

- The scientific and engineering fields of software evolution

- **Foundations**

- Laws of Software Evolution
 - Software Aging
 - Roadmap

- **Methods & Tools**

- Dealing with Legacy Systems
 - Re-engineering

Introduction



Research on Software Evolution

Google scholar

software evolution

Search

Advanced Scholar Search
Scholar Preferences

Scholar

Articles and patents

anytime

include citations

Results 1 - 10 of about 2,080,000. (0.15 sec)

Architecture-based runtime software evolution

P Oreizy, N Medvidovic, RN Taylor - ... conference on Software ..., 1998 - portal.acm.org
ABSTRACT Continuous availability is a critical requirement for an important class of software systems. For these systems, runtime system evolution can mitigate the costs and risks associated with shutting down and restarting the system for an update. We present an architecture- ...
Cited by 114 Related articles - BL Direct - All 19 versions - Import into BibTeX

[psu.edu](#) [PDF]

[PDF] Programs, life cycles, and laws of software evolution

MM Lehman... - Proceedings of the IEEE, 1980 - ipd.bth.se
PROCEEDINGS OF THE IEEE, VOL. 68, NO. 9, SEPTEMBER 1980 Programs, Life Cycles and Laws of Software Evolution MEIR M. LEHMAN, senior member, ieee Abstract—By classifying programs according to their relationship to the environment in which they are executed.
Cited by 540 Related articles - All 2 versions - Import into BibTeX

[bth.se](#) [PDF]

[PDF] Evolution in software engineering: A survey

MW Godfrey, Q Tu - 16th IEEE International Conference on Software ..., 2000 - Citeseer
Most studies of software evolution have been performed on systems developed within a single company using traditional management techniques. With the widespread availability of several large software systems that have been developed using an "open source" development ...
Cited by 335 Related articles - View as HTML - All 47 versions - Import into BibTeX

[psu.edu](#) [PDF]

Metrics and laws of software evolution-the nineties view

... , JF Ramil, PD Wernick, DE Perry, ... - Software Metrics ..., 1997 - doi.ieeecomputersociety.org
Imperial College of Science, Technology and Medicine London SW7 2BZ tel: +44 (0)171 594 8214 fax: +44 (0) 171 594 82 15 e-mail: (mml.icfl_ndw@doc.ic.ac.uk URL: http://www-dse.doc.ic.ac.uk/~mml/feast/ ... Wroclaw Institute of Informatics Warsaw University Warsaw ...
Cited by 11 Related articles - All 36 versions - Import into BibTeX

[psu.edu](#) [PDF]

Software maintenance and evolution: a roadmap

KH Bennett, VT Rajlich - ... of the conference on The future of Software ..., 2000 - portal.acm.org
Keith Bennett has been a full Professor since 1986, and a former Chair, both within the Department of Computer Science at the University of Durham. For the past fourteen years he has worked on methods and tools for program comprehension and reverse engineering, based on ...
Cited by 263 Related articles - All 22 versions - Import into BibTeX

[psu.edu](#) [PDF]

Laws of software evolution

MM Lehman - Lecture Notes in Computer Science, 1996 - Springer
Abstract. Data obtained during a 1968 study of the software process [8] led to an investigation of the evolution of OS/360 [13] and, over a period of twenty years, to formulation of eight Laws of Software Evolution. The FEAST project recently initiated (see sections 4 - 6 ...
Cited by 237 Related articles - BL Direct - All 32 versions - Import into BibTeX

[psu.edu](#) [PDF]

Research on Software Evolution

Google scholar [Advanced Scholar Search](#) [Scholar Preferences](#)

Scholar [Articles and Citations](#) [anytime](#) [Include citations](#) Results 1 - 10 of about 1,230,000. (0.15 sec)

Software aging

DL Parnas - ... of the 16th international conference on Software ..., 1994 - portal.acm.org
ABSTRACT Programs, like people, get old. We can't prevent aging, but we can understand its causes, take steps to limit its effects, temporarily reverse some of the damage it has caused, and prepare for the day when the software is no longer viable. A sign that the Software
[Cited by 460](#) - [Related articles](#) - [All 12 versions](#) - [Import into BibTeX](#)

A methodology for dealing with the problem of software aging

S Garg, Avan Moorsel, K ... - ... on Software ..., 1998 - doi.ieeecomputersociety.org
Sachin Garg , Aad van Moorsel Lucent Technologies Bell Laboratories 600 Mountain Avenue Murray Hill, NJ 07974, USA f sgarg,aad g @research.bell-labs.com ... Kalyanaraman Vaidyanathan, Kishor S. Trivedi Center for Advanced Computing & Communication
[Cited by 142](#) - [Related articles](#) - [All 8 versions](#) - [Import into BibTeX](#)

[PDF] Proactive management of software aging

V Castelli, RE Harper, P Heidelberger, SW ... - IBM Journal of ..., 2001 - Citeseer
Software failures are now known to be a dominant source of system outages. Several studies and much anecdotal evidence point to "software aging" as a common phenomenon, in which the state of a software system degrades with time. Exhaustion of system resources, data ...
[Cited by 165](#) - [Related articles](#) - [View as HTML](#) - [BL Direct](#) - [All 12 versions](#) - [Import into BibTeX](#)

Modeling and analysis of software aging and rejuvenation

KS Trivedi, K Vaidyanathan, K ... - ANNUAL ..., 2000 - doi.ieeecomputersociety.org
Software systems are known to suffer from outages due to transient errors. Recently, the phenomenon of "software aging", one in which the state of the software system de-grades with time, has been reported. To counteract this phe-nomenon,a proactive approach of fault management, ...
[Cited by 78](#) - [Related articles](#) - [BL Direct](#) - [All 18 versions](#) - [Import into BibTeX](#)

[ncsu.edu \[PS\]](#)

[psu.edu \[PDF\]](#)

[psu.edu \[PDF\]](#)

Concepts

Legacy Systems

Agile processes

Life Cycle

Maintenance

Software Aging

Software Evolution

Reverse Engineering

Program Comprehension

Re-engineering

Program Transformation

Mining Software Repositories

Metrics

Research dimensions

Basic research

How do we model a software system and its evolution?

Empirical studies and validation

Industrial, large scale systems

Software Evolution

Methods and tools-oriented research

How do we support program comprehension and program transformation?

Research on Software Evolution

- **Pioneers:**

- Keith H. Bennett
- Meir M. Lehman
- Bennet P. Lientz, Lientz
- David Lorge Parnas
- Vaclav T. Raijlich
- E. Burton Swanson Swanson

- **Some of the people in the community:**

- Serge Demeyer, Universiteit Antwerpen (Belgium)
- Stephane Ducasse, INRIA (France)
- Harald C. Gall, Software Evolution and Architecture lab, University of Zürich (Switzerland)
- Tudor Girba, did his Ph.D. at the Software Composition Group at University of Bern (Switzerland)
- Michele Lanza, University of Lugano (Switzerland)
- Tom Mens, Software Engineering Lab, Université de Mons (Belgium)
- Oscar Nierstrasz, Software Composition Group, University of Bern.



Publish Date: March 10, 2008
Print ISBN: 3540764399

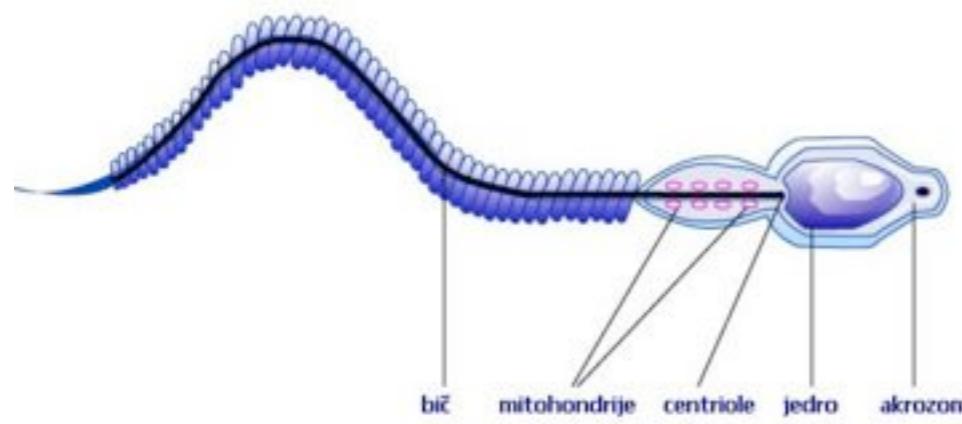
History and Challenges of Software Evolution

- **1968:** first conference on Software Engineering, organized by the NATO Science Committee, with the goal to establish **sound engineering principles** in order to obtain reliable, efficient and economically viable software.
- **1970:** Royce proposes the **waterfall life-cycle process** for software development. Maintenance is seen as the final phase of the software lifecycle (with only bug fixes and minor adjustments). The model had a strong and long influence on the industrial practice of software development.
- **Late 1970's:** first attempt towards a more evolutionary process model. Identification of new activities, such as impact analysis and change propagation. In the same period, formulation of "**Laws of software evolution**" by Lehman.
- **1990's:** widespread acceptance of software evolution, formalization of evolutionary processes (Gilb's evolutionary development, Boehm's spiral model, Bennet and Rajich's staged model).
- **Software evolution is a crucial ingredient of agile software development (iterative and incremental development, embracing change!)**

History and Challenges of Software Evolution

- **Two dimensions of Software evolution**

- **What and Why?** Software evolution as a **scientific discipline**, which studies the nature of the software evolution **phenomenon**, and seeks to understand its driving factor, its impact.
- **How?** Software engineering as an **engineering discipline**, which studies more **pragmatic aspects** that aid software developers and project managers in their day-to-day tasks. Focus on **technology, methods, tools** and activities that provide a means to direct, implement and control software evolution.



Software Maintenance

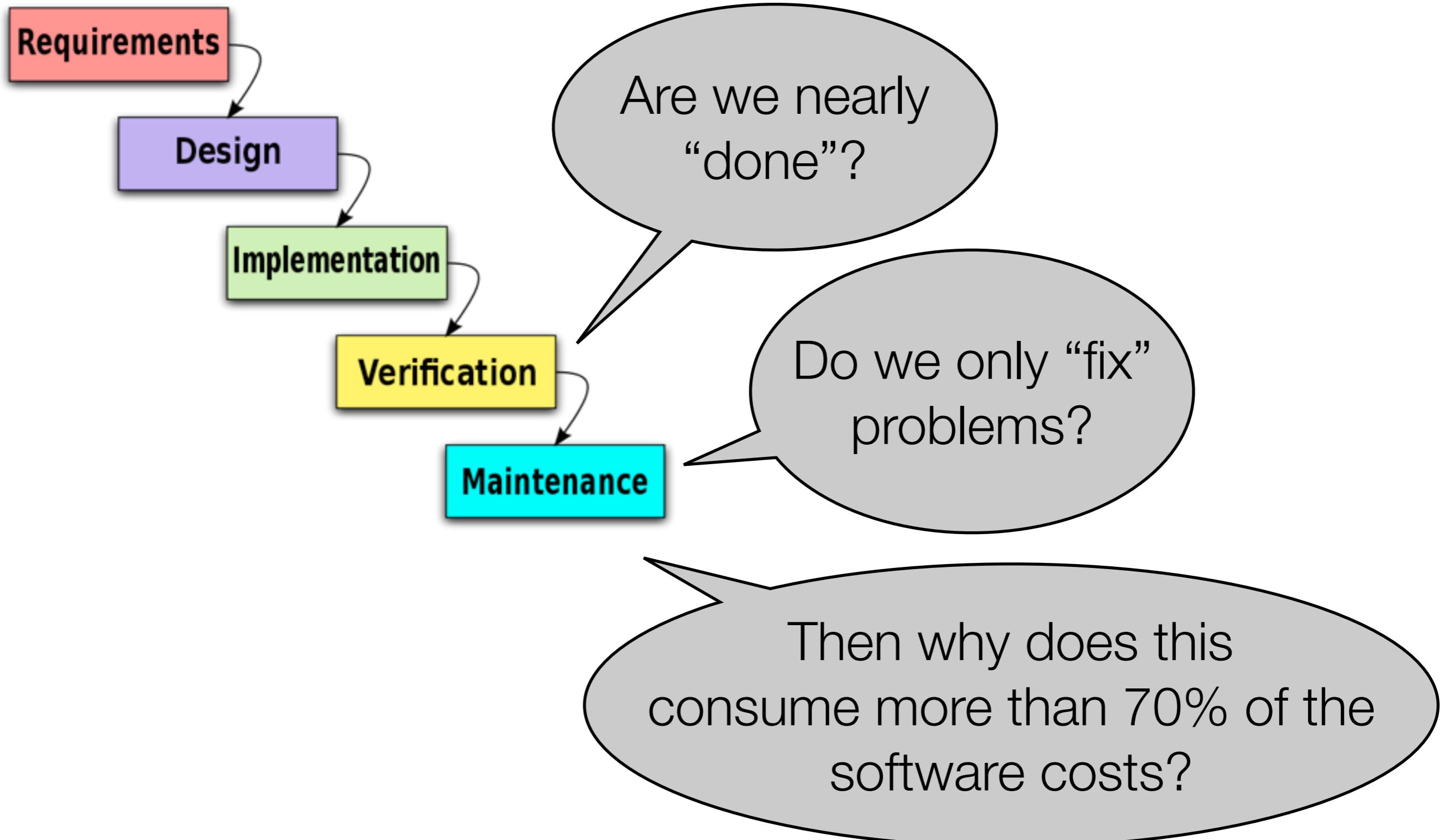
- In many engineering disciplines, **physical products** are designed, built and delivered to users.
- Think about a **bridge**, a **building**, a **car** or a **vacuum cleaner**.
- Maintenance is about making sure that the product **stays operational** over the years.
- Since we are talking about physical products, we are thinking about fixing defects, changing **worn-out components**, do some cleaning, adding oil, etc.
- When we talk about maintenance, the **functionality of the product stays the same**.



http://www.flickr.com/photos/laenulfean/474217855/sizes/m/#cc_license

*Is software maintenance only
about fixing defects?*

What does it mean to “maintain” software?



Software Maintenance

- Maintenance is **costly** - between 70% to 80% of the software costs are spent on maintenance.
- Classification of software maintenance activities:
 - **Corrective:** errors need to be fixed.
 - **Preventive:** prevent problems in the future (fix design issues).
 - **Adaptive:** something has changed in the environment.
 - **Perfective:** improve system qualities, e.g. performance.



Management
Applications

H. Morgan
Editor

Characteristics of Application Software Maintenance

B. P. Lientz, E. B. Swanson,
and G. E. Tompkins

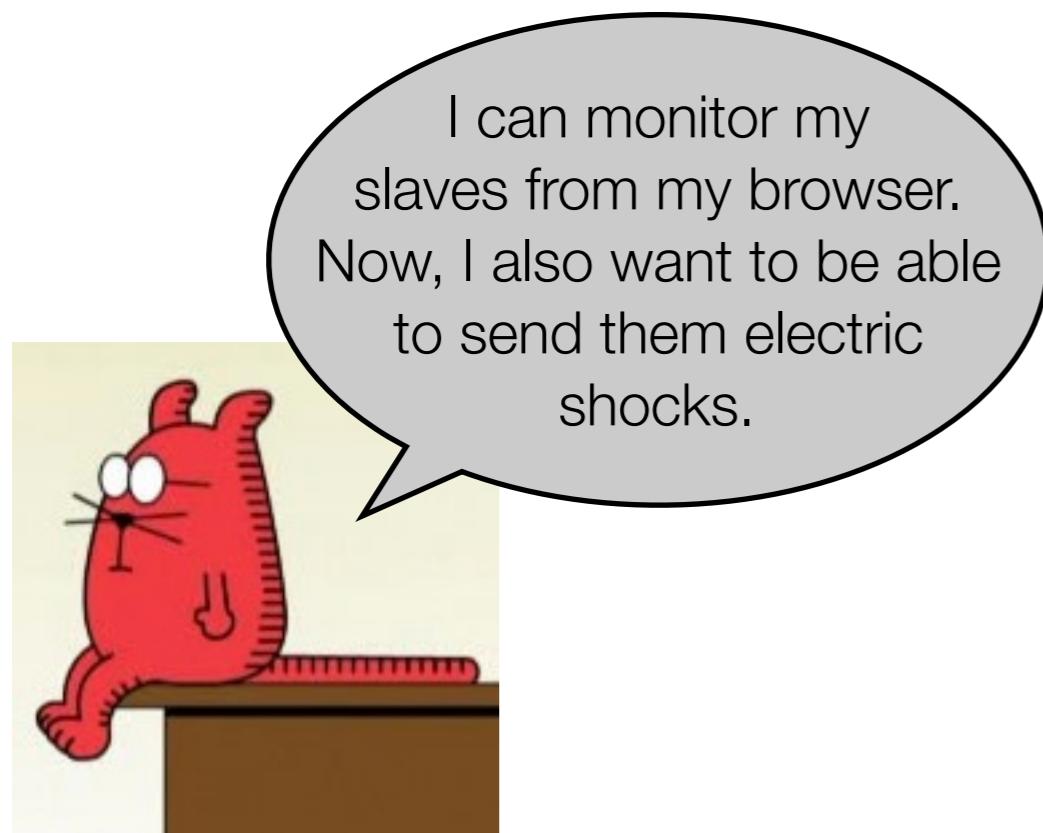
University of California at Los Angeles

Maintenance and enhancement of application software consume a major portion of the total life cycle cost of a system. Rough estimates of the total systems and programming resources consumed range as high as 75-80 percent in each category. However, the area has been given little attention in the literature. To analyze the problems in this area a questionnaire was developed and pretested. It was then submitted to 120 organizations. Respondents totaled 69. Responses were analyzed with the SPSS statistical package. The results of the analysis indicate that: (1) maintenance and enhancement do consume much of the total resources of systems and programming groups; (2) maintenance and enhancement tend to be viewed by management as at least somewhat more important than new application software development; (3) in maintenance and enhancement, problems of a management orientation tend to be more significant than those of a technical orientation; and (4) user demands for enhancements and extension constitute the most important management problem area.

Software Maintenance



Moving towards Servlet 3.0 means that we will have less worries about scalability...



Let's replace this JPA query with a native SQL query to gain a 15% performance improvement!

Software Maintenance

Corrective

Preventive

I can monitor my
slaves from my browser.
Now, I also want to be able
to do X.

Adaptive

Perfective

Moving towards
what we
about
S.

Let's replace this
JPA query with a native
SQL query to gain a 15%
performance
ment!

Lehman's “Laws” of Evolution

- Propose a **theoretical model** to reason about software systems and their interaction with the socio-economic environment.
- Maintenance is costly, but maintenance is not only about bug fixing!
- Propose a classification of software: S-Programs, P-Programs and E-Programs.
- Conduct **quantitative studies** on large scale industrial projects, (collect metrics)
- Derive “**Laws of Evolution**” that are generally applicable.

THE TOTAL U.S. expenditure on programming in 1977 is estimated to have exceeded \$50 billion, and may have been as high as \$100 billion. This figure, which represents more than 3 percent of the U.S. GNP for that year, is already an awesome figure. It has increased ever since in real terms and will continue to do so as the microprocessor finds ever wider application. Programming effectiveness is clearly a significant component of national economic health. Even small percentage improvements in productivity can make significant financial impact. The potential for saving is large.

Economic considerations are, however, not necessarily the main cause of widespread concern. As computers play an ever larger role in society and the life of the individual, it becomes more and more critical to be able to create and maintain effective, cost-effective, and timely software. For more than two decades, however, the programming fraternity, and through them the computer-user community, has faced serious problems in achieving this [1]. As the application of microprocessors extends ever deeper into the fabric of society the problems will be compounded unless very basic solutions are found and developed.

“Programs, Life Cycles, and Laws of Software Evolution”, Proceedings of the IEEE, Vol. 68, No. 9, September 1980.

Different Types of Software Systems

- **S-Programs.** Programs that can be completely and formally specified.
 - e.g. a program that sorts an array.
- **P-Programs.** Programs that can be completely specified, but which makes an approximation of the real world.
 - e.g. a program that plays chess against a human player.
- **E-Programs.** Programs that mechanize a human or societal activity. The program becomes a part of the world it models!
 - e.g. an ERP system.

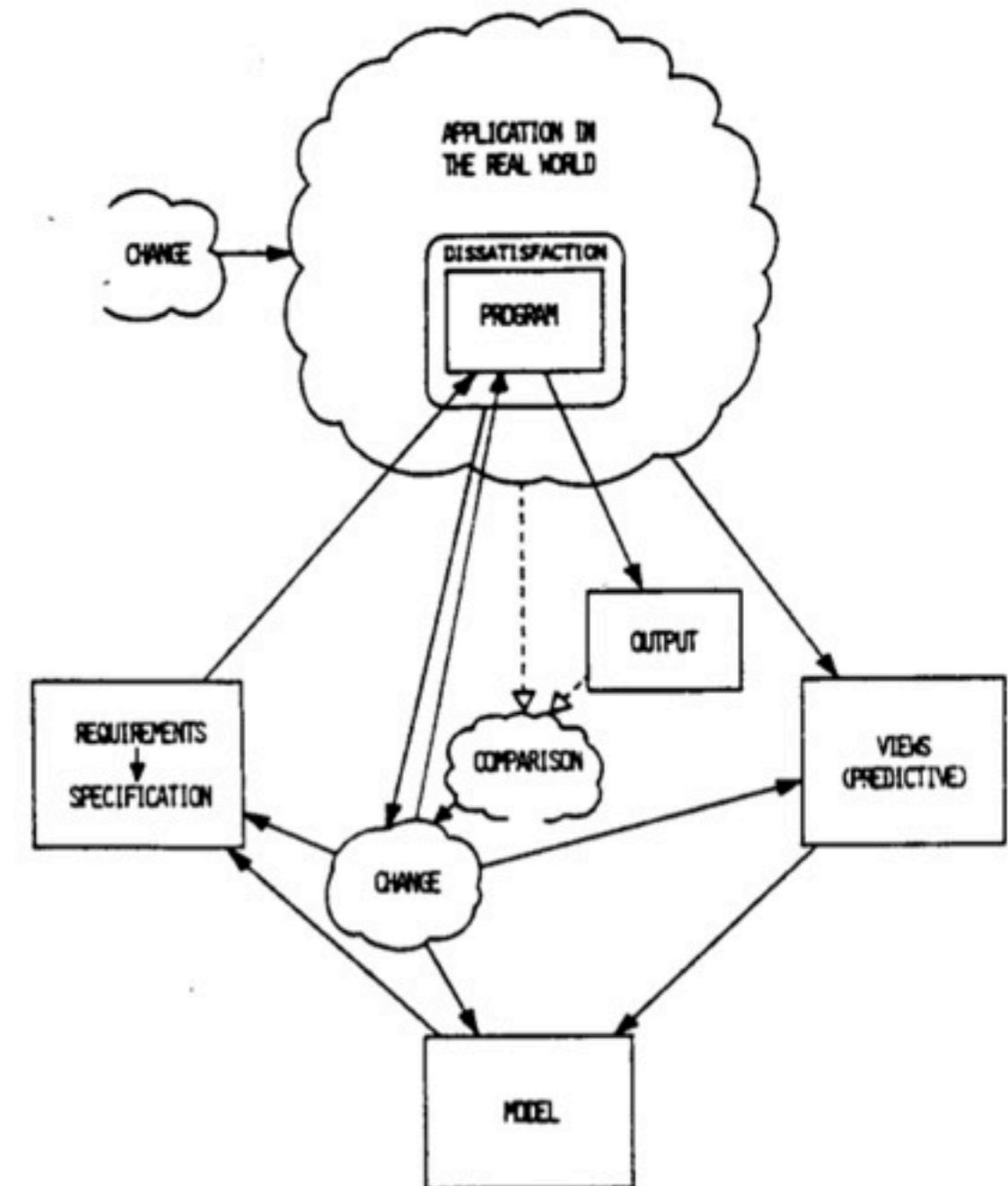
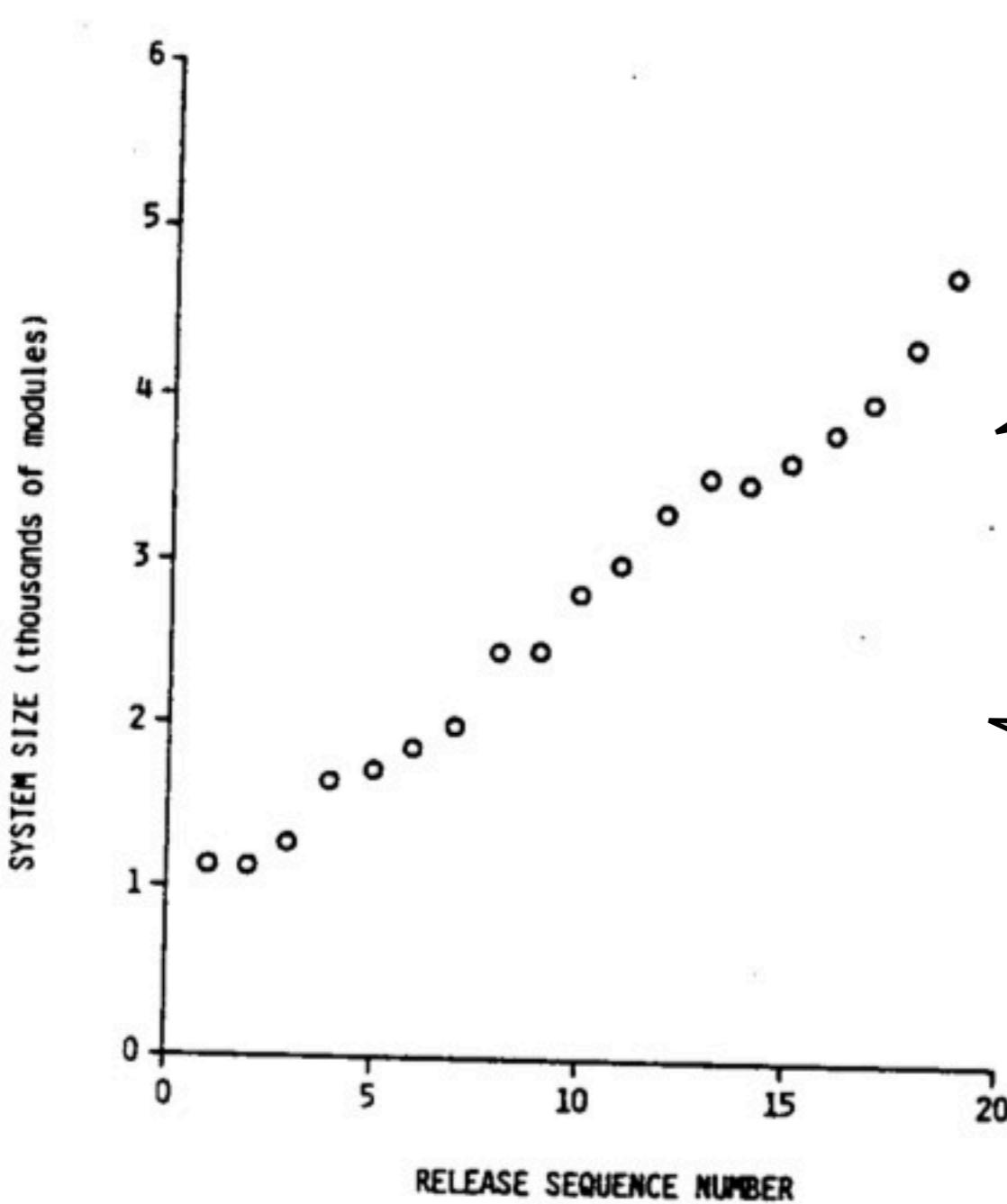


Fig. 4. E-programs.

Quantitative Studies

- **Collection of data** on a large scale, industrial project (IBM OS 360)
- **Analysis over**
 - Elapsed time
 - Release numbers
- **Metrics**
 - System size (number of modules)
 - Incremental growth
 - Modules changed (indicates complexity)
 - Interval
- **Statistical analysis shows trends in metrics evolution.**

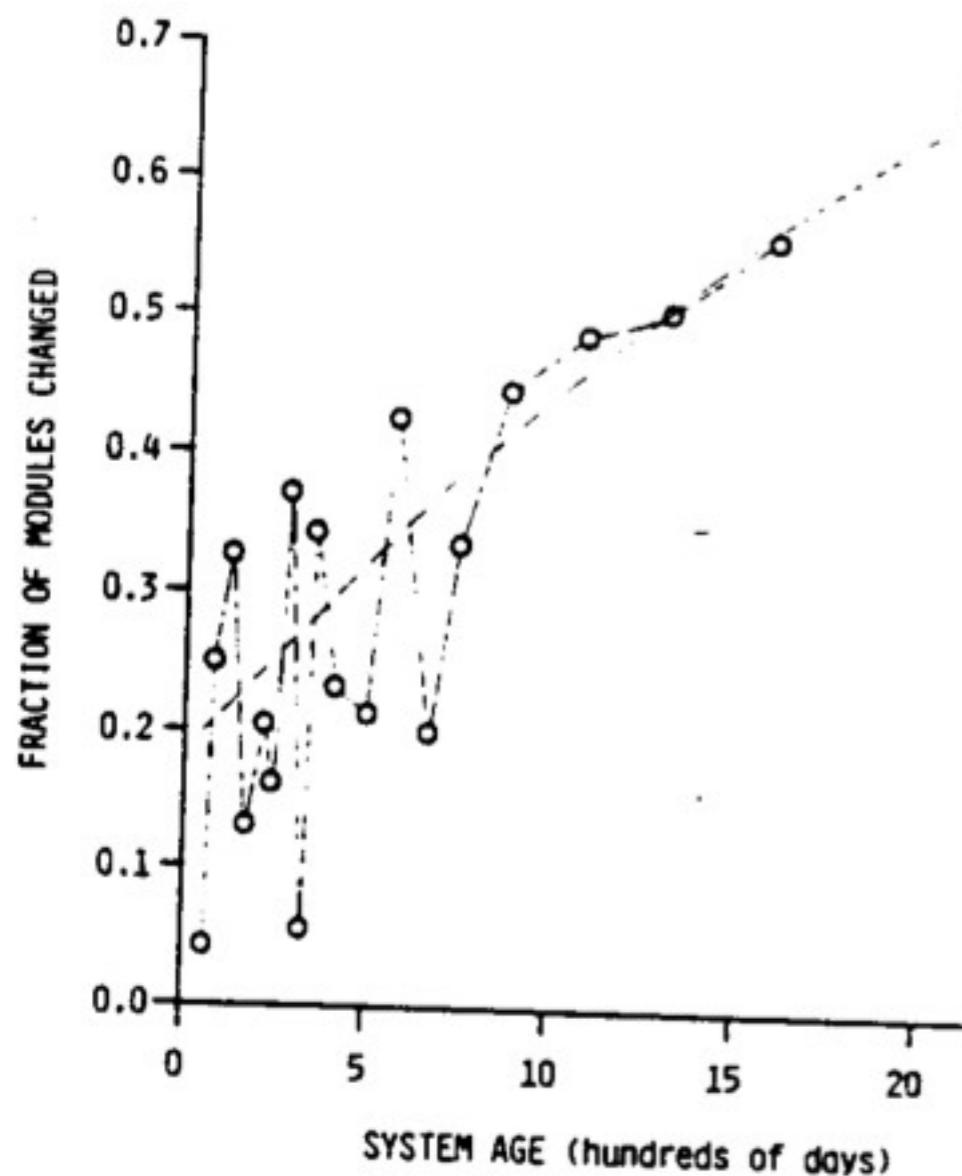
Quantitative Studies



The system grows in a predictable way. This reflects the constant evolution!

So... we're not only fixing bug then?

Quantitative Studies



The number of modules that must be changed for a release reflects the complexity of the system.

So... it becomes more and more complex then?

Lehman's “Laws” of Evolution

Software systems have to **evolve**, otherwise they gradually become useless.

If you don't take specific actions, software will become more **complex** and more difficult to adapt.

TABLE I
LAWS OF PROGRAM EVOLUTION

I. Continuing Change

A program that is used and that as an implementation of its specification reflects some other reality, undergoes continual change or becomes progressively less useful. The change or decay process continues until it is judged more cost effective to replace the system with a recreated version.

II. Increasing Complexity

As an evolving program is continually changed, its complexity, reflecting deteriorating structure, increases unless work is done to maintain or reduce it.

III. The Fundamental Law of Program Evolution

Program evolution is subject to a dynamics which makes the programming process, and hence measures of global project and system attributes, self-regulating with statistically determinable trends and invariances.

IV. Conservation of Organizational Stability (Invariant Work Rate)

During the active life of a program the global activity rate in a programming project is statistically invariant.

Conservation of Familiarity (Perceived Complexity)

During the active life of a program the release content (changes, additions, deletions) of the successive releases of an evolving program is statistically invariant.

Lehman's “Laws” of Evolution (nineties view)

If you don't take
specific actions, software
quality will decline!

No.	Brief Name	Law
I 1974	Continuing Change	<i>E</i> -type systems must be continually adapted else they become progressively less satisfactory.
II 1974	Increasing Complexity	As an <i>E</i> -type system evolves its complexity increases unless work is done to maintain or reduce it.
III 1974	Self Regulation	<i>E</i> -type system evolution process is self regulating with distribution of product and process measures close to normal.
IV 1980	Conservation of Organisational Stability (invariant work rate)	The average effective global activity rate in an evolving <i>E</i> -type system is invariant over product lifetime.
V 1980	Conservation of Familiarity	As an <i>E</i> -type system evolves all associated with it, developers, sales personnel, users, for example, must maintain mastery of its content and behaviour [leh80a] to achieve satisfactory evolution. Excessive growth diminishes that mastery. Hence the average incremental growth remains invariant as the system evolves.
VI 1980	Continuing Growth	The functional content of <i>E</i> -type systems must be continually increased to maintain user satisfaction over their lifetime.
VII 1996	Declining Quality	The quality of <i>E</i> -type systems will appear to be declining unless they are rigorously maintained and adapted to operational environment changes.
VIII 1996 (first stated 1974, formalised as law 1996)	Feedback System	<i>E</i> -type evolution processes constitute multi-level, multi-loop, multi-agent feedback systems and must be treated as such to achieve significant improvement over any reasonable base.

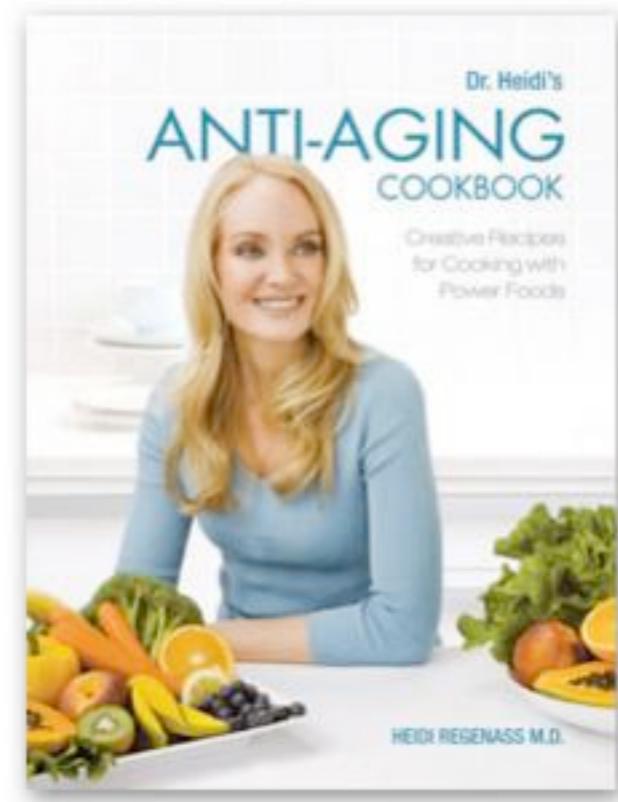
Software Aging

“Programs, like people, get old. We can’t prevent aging, but we can understand its causes, take steps to limit its effects, temporarily reverse some of the damage it has caused, and prepare for the day when the software is no longer viable.”

David Lorge Parnas, 1994

Software Aging

```
cmd C:\Windows\system32\cmd.exe
C:\Users\Softpedia\Desktop\PDF-Chart-Creator-Command-Line-Tool\v1-33>\PDFChart.exe -log result.log -options PDF-Chart-Creator-Example-Bar-Horizontal-Chart.txt -openpdf
C:\Users\Softpedia\Desktop\PDF-Chart-Creator-Command-Line-Tool\v1-33>\PDFChart.exe -log result.log -options PDF-Chart-Creator-Example-Bar-Vertical-Chart.txt -openpdf
C:\Users\Softpedia\Desktop\PDF-Chart-Creator-Command-Line-Tool\v1-33>\PDFChart.exe -log result.log -options PDF-Chart-Creator-Example-Pie-Spoke-Chart.txt -openpdf
C:\Users\Softpedia\Desktop\PDF-Chart-Creator-Command-Line-Tool\v1-33>\PDFChart.exe -log result.log -options PDF-Chart-Creator-Example-Pie-Legend-Chart.txt -openpdf
C:\Users\Softpedia\Desktop\PDF-Chart-Creator-Command-Line-Tool\v1-33>\PDFChart.exe -log result.log -options PDF-Chart-Creator-Example-Line-Chart-Standard.txt -openpdf
```



The causes of software aging

- **Lack of movement**

- Caused by the failure of the product's owners to modify it to meet **changing needs**.
- Unless software is frequently updated, its users will **become dissatisfied** and they will change to a new product as soon as the benefits outweigh the costs of retraining and converting.

- **Ignorant surgery**

- Caused by the **changes** that are made to software.
- Changes made by people who do not understand the original design concept almost always cause the **structure of the program to degrade**.
- Software that has been repeatedly modified in this way becomes very expensive to update.

The costs of software aging

- The **symptoms** of software aging mirror those of human aging:
 - Owners of aging software find it increasingly **hard to keep up** with the market and lose customers to newer products (“weight gain”)
 - Aging software often **degrades in its performance** as a result of a gradually deteriorating structure.
 - Aging software often becomes “**buggy**” because of errors introduced when changes are made.

Reducing the costs of software aging

- **Preventive medicine**

- What can we do to delay the decay and limit its effects?
- Design for change
- Keep records - documentation
- Second opinion - reviews

- **Software geriatrics**

- What can we do to treat software aging that has already occurred?
- Stopping the deterioration
- Retroactive documentation
- Retroactive incremental modularization
- Amputation
- Major surgery - restructuring

Software Maintenance and Evolution: A Roadmap

A major challenge for the research community is to develop a good **theoretical understanding** and underpinning for maintenance and evolution, which scales to **industrial** applications. **Most computer science research has been of benefit to the initial development of software.** Type theory and configuration management have in different ways made major contributions to maintenance. Many others claim to do so, but reliable empirical evidence is lacking.

The **staged model of software lifecycle** was introduced in [BENN99] and is summarized in Figure 1. It represents the software lifecycle as a **sequence of stages**, with initial development being the first stage. Its key contribution is to **separate the 'maintenance'** phase into an **evolution** stage followed by a **servicing** and **phase-out** stages.

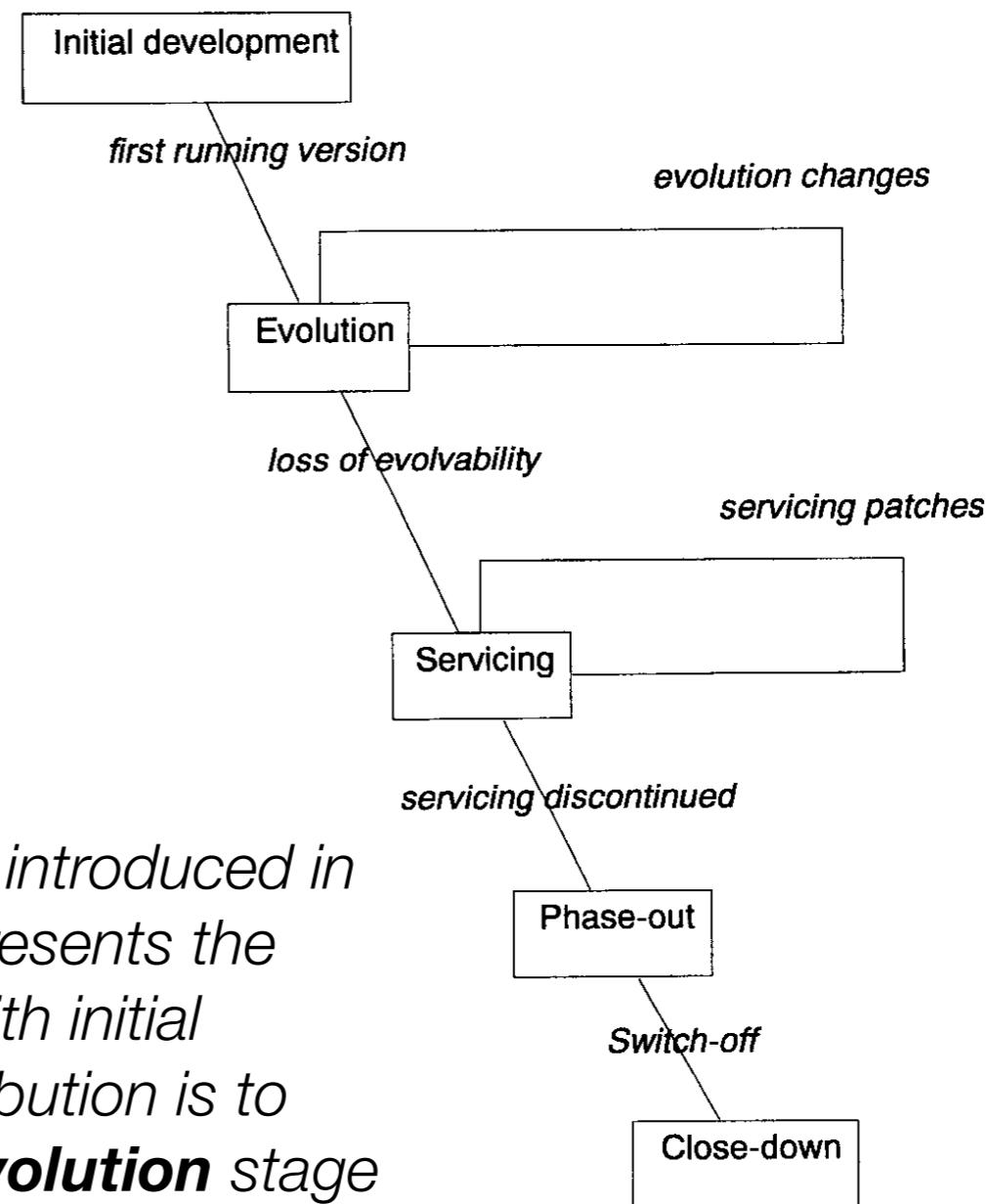


Figure 1. The simple staged model

Software Maintenance and Evolution: A Roadmap

Each stage has very different technical solutions, processes, staff needs and management activities.

An amplification of the staged model is the **versioned staged model**. The software team produces versions of the software during an **extended phase of evolution**, but the versions are no longer evolved, only serviced.

All substantial changes in the functionality are implemented in the future versions.

If a version becomes outdated and the users need a new functionality, they have to replace their version with a new one.

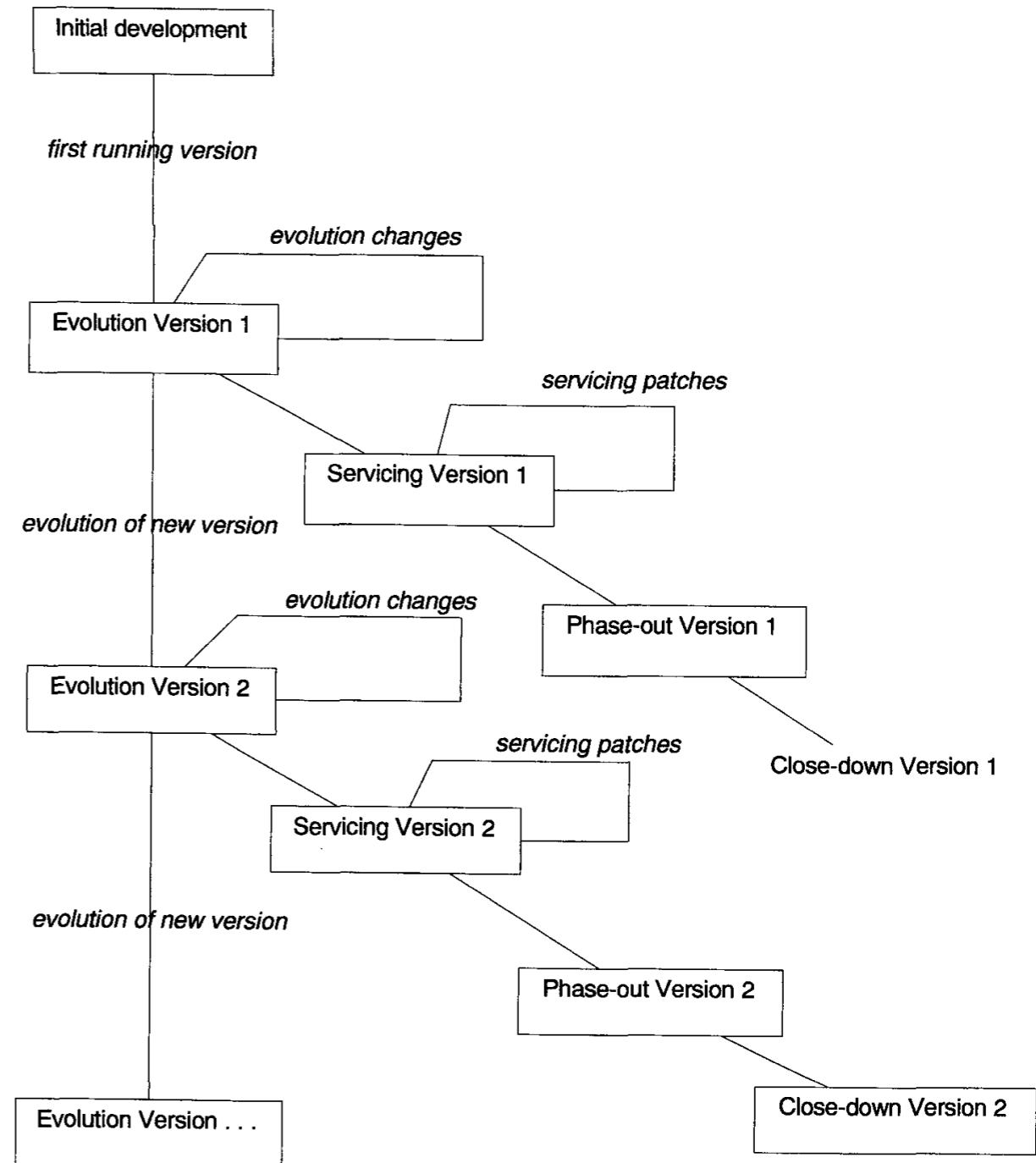


Figure 2. The versioned staged model

Software Maintenance and Evolution: A Roadmap

- **Initial development stage**

- **Research challenge:** find ways of developing software, so that it can be changed more easily and reliably in subsequent phases.
- Two key outcomes of this phase: i) **architecture** and ii) **team knowledge**

- **Evolution stage**

- **Goal:** implement possibly major changes without being able a priori to predict how user requirements will evolve.
- **Management issue:** keep original architects and designers in the team - or risk to move into the servicing stage quickly.

Software Maintenance and Evolution: A Roadmap

- **Servicing stage**
 - **Goal:** implement **tactical changes**, at minimum cost.
 - **Big change in terms of project management:** only minor corrections/enhancements should be done, senior staff members are no longer required, accurate cost prediction is needed. It is possible to outsource the servicing (how do we define SLAs?)
 - **Research challenge:** program comprehension. We need tools to understand the impact of changes, to display the program structure, to do regression testing. We also need tools for configuration management and version control of large distributed systems.
- **Phase-out and close down stages**

Methods & Tools



Overview

*The documentation is missing or obsolete, and the original developers have departed. Your team has limited understanding of the system, and unit tests are missing for many, if not all, of the components. When you fix a bug in one place, another bug pops up somewhere else in the system. Long rebuild times make any change difficult. All of these are **signs of software that is close to the breaking point.***

*Many systems can be upgraded or simply thrown away if they no longer serve their purpose. **Legacy software, however, is crucial for operations and needs to be continually available and upgraded.***

How can you reduce the complexity of a legacy system sufficiently so that it can continue to be used and adapted at acceptable cost?

Legacy System

- **Legacy:** a sum of money, or a specified article, given to another by will; anything handed down by an ancestor or predecessor (Oxford English Dictionary)
- **Legacy system**

- A software system that you have **inherited**.
- A software system that is still **valuable**.
- A software system that **may not be easy to deal with...**



Dealing with Legacy Systems

- **Typical problems**

- Original developers have left or are not available.
- Extensive patches and modifications have been made.
- Documentation is not up-to-date or even missing.
- Architecture may be not be optimal (improper layering, lack of modularity, etc.)

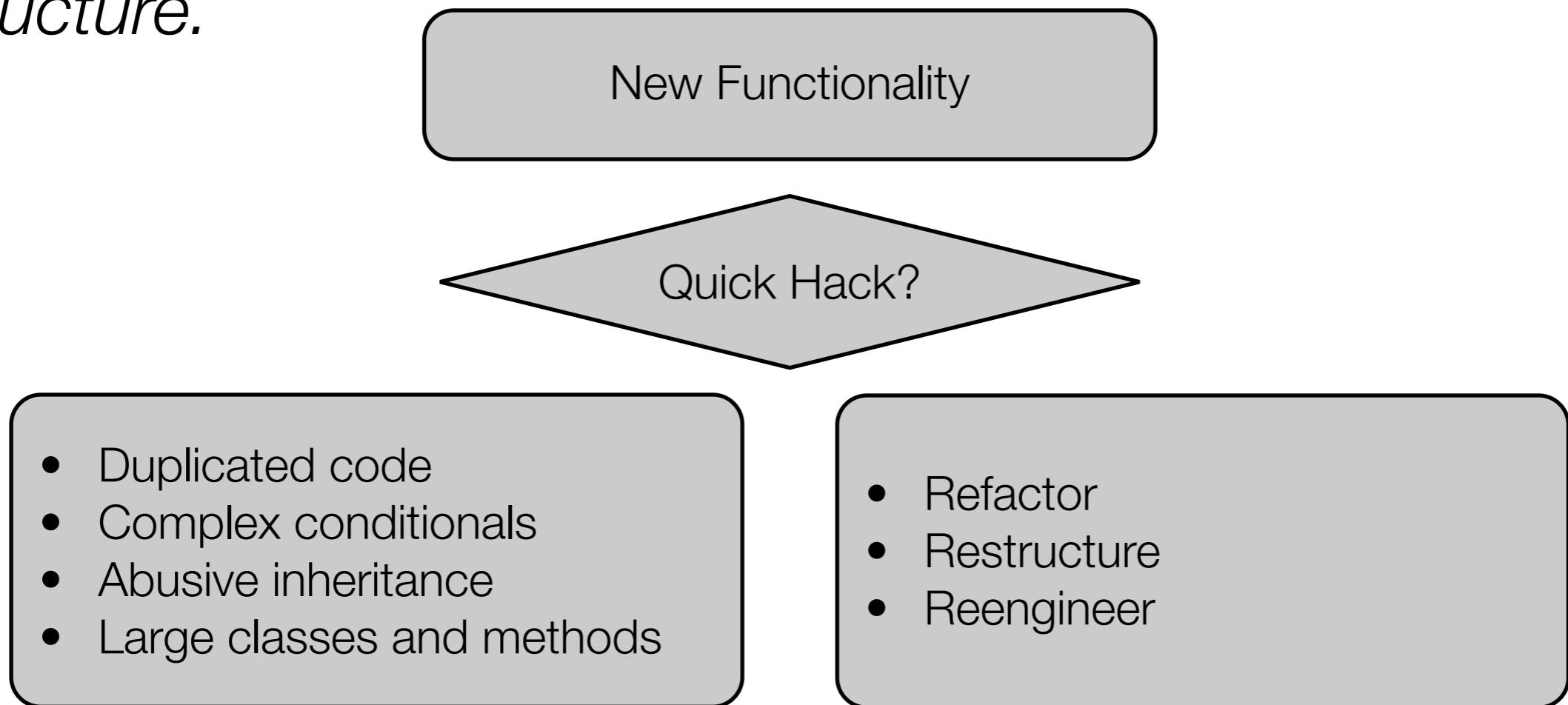


Evolution is more and more difficult...

Dealing with Legacy Systems

New or changing requirements will gradually degrade original design...

...unless extra development effort is spent to adapt the structure.



Take a loan on your software

Invest for the future

Terminology

- **Forward Engineering**

- traditional process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a system

- **Reverse Engineering**

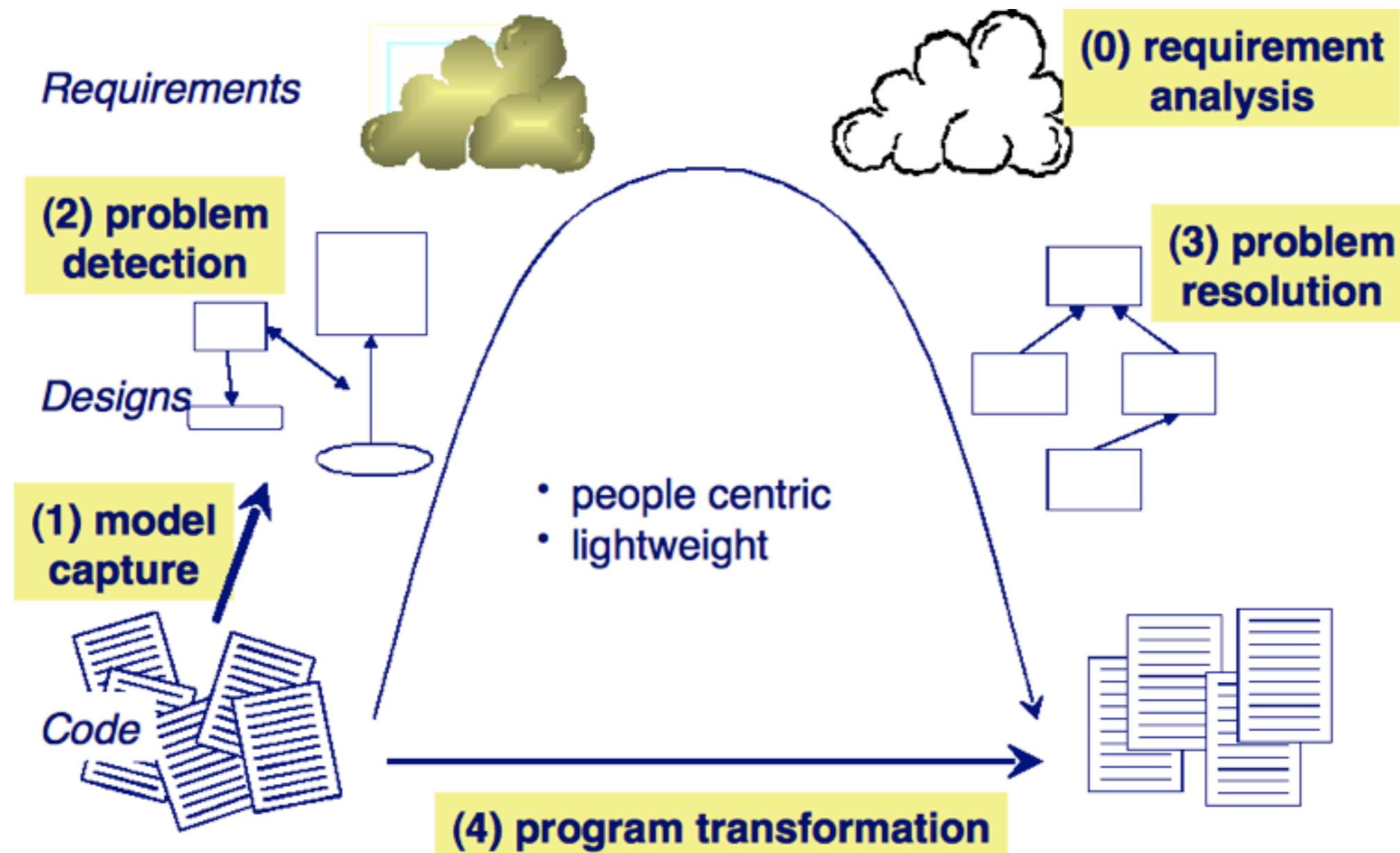
- process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction.

- **Reengineering**

- examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form.

Chikofsky and Cross, 1993

The Reengineering Life-Cycle



Goals of Reengineering

- **Unbundling**
 - split a monolithic system into parts that can be separately marketed
- **Performance**
 - “first do it, then do it right, then do it fast” – experience shows this is the right sequence!
- **Port to other Platform**
 - the architecture must distinguish the platform dependent modules
- **Design extraction**
 - to improve maintainability, portability, etc.

Goals of Reverse Engineering

- **Cope with complexity**
 - need techniques to understand large, complex systems
- **Recover lost information**
 - extract what changes have been made and why
- **Detect side effects**
 - help understand ramifications of changes
- **Synthesize higher abstractions**
 - identify latent abstractions in software
- **Facilitate reuse**
 - detect candidate reusable artifacts and components

Chikofsky and Cross, 1993

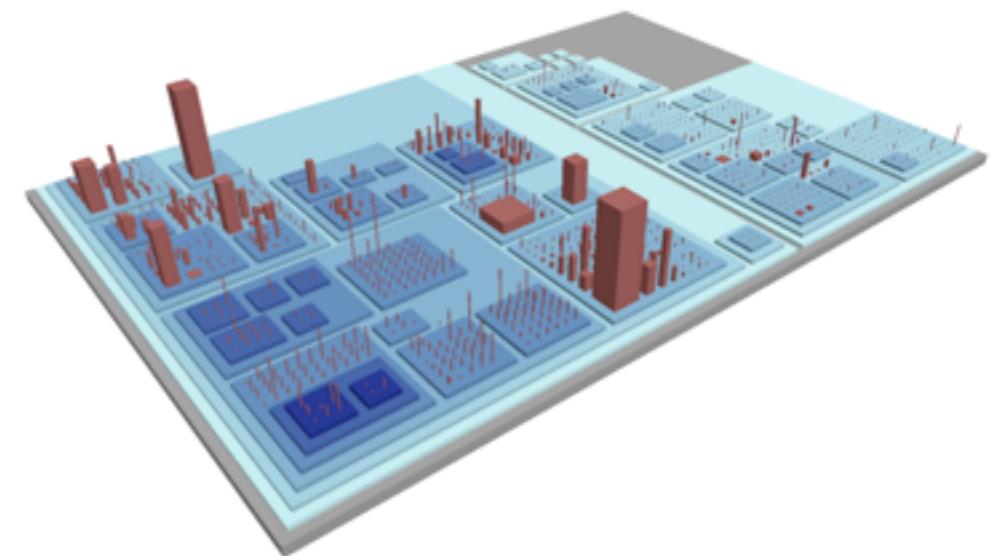
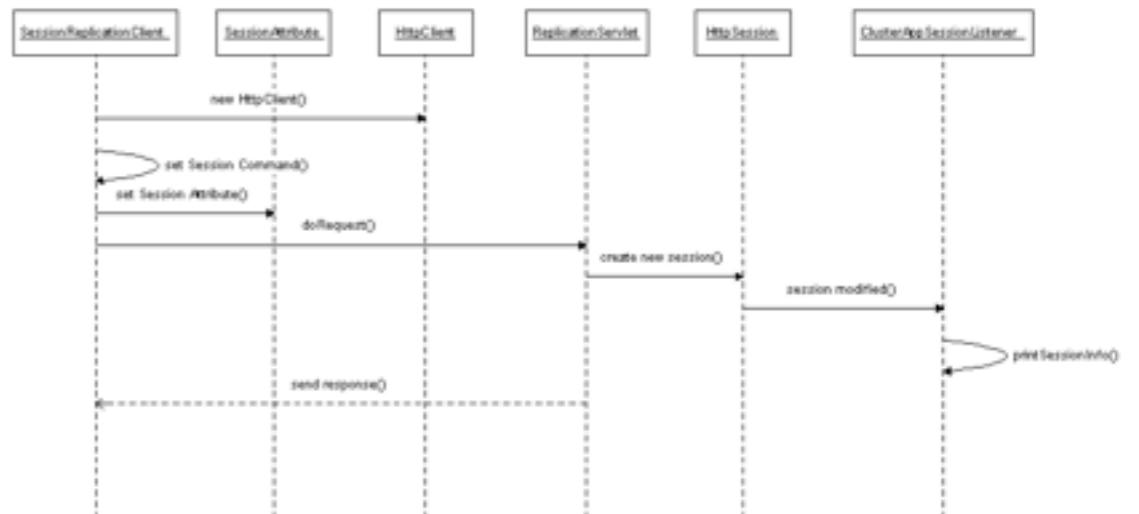
Reverse Engineering Techniques

- **Redocumentation**

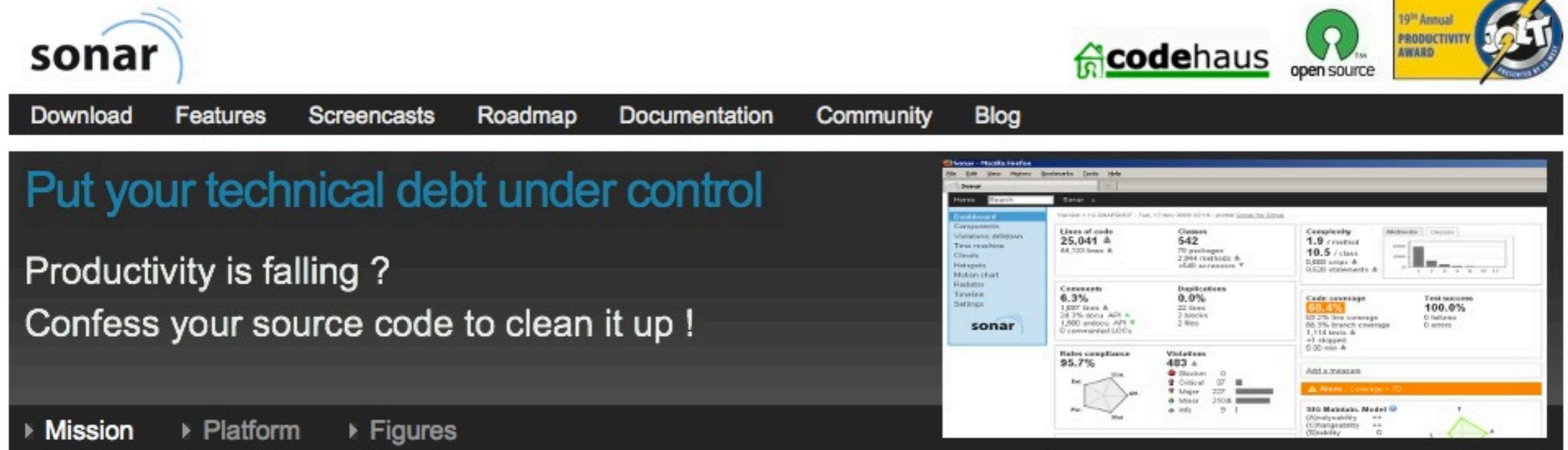
- pretty printers
- diagram generators
- cross-reference listing generators

- **Design recovery**

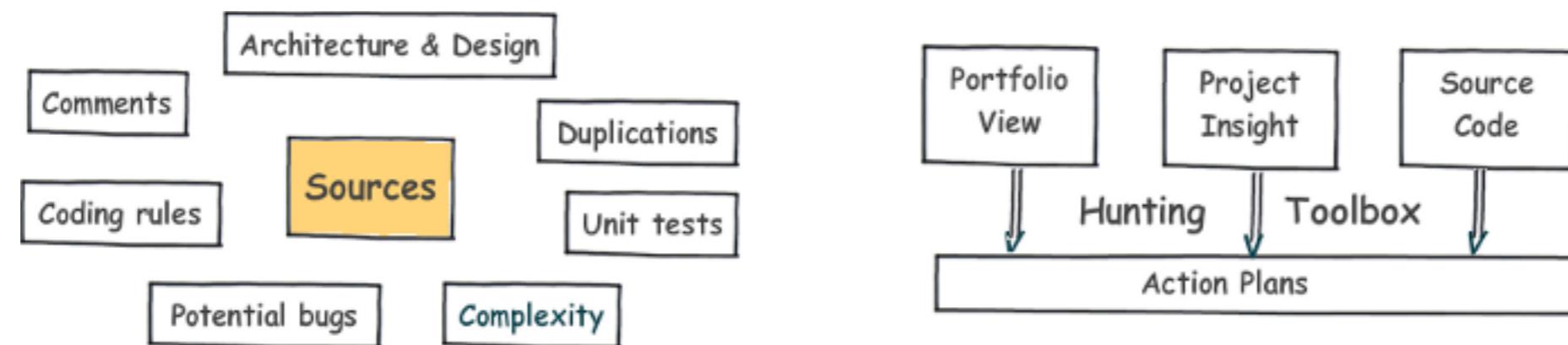
- software metrics
- browsers, visualization tools
- static analyzers
- dynamic (trace) analyzers



Tool: Sonar



The Sonar website features a top navigation bar with links for Download, Features, Screencasts, Roadmap, Documentation, Community, and Blog. Below this, a large banner with the text "Put your technical debt under control" and "Productivity is falling ? Confess your source code to clean it up !" is displayed. A navigation menu at the bottom includes Mission, Platform, and Figures. To the right of the banner is a detailed dashboard showing various metrics: Classes of code (25,041), Comments (6.3%), Duplications (0.0%), Code coverage (66.4%), and Rules compliance (95.7%). The dashboard also includes sections for Complexity, Test success, and a Risk Analysis chart.



The diagram illustrates the Sonar architecture and workflow. On the left, a cluster of boxes represents the "Sources" component, containing sub-boxes for Architecture & Design, Comments, Duplications, Coding rules, Unit tests, Potential bugs, and Complexity. On the right, a flowchart shows the process: Portfolio View, Project Insight, and Source Code feed into Hunting, which leads to Action Plans. Hunting also feeds into a central "Toolbox" box, which then feeds into Action Plans.

<http://sonar.codehaus.org/>

Sonar: technical debt

- Concept invented by Ward Cunningham, also described by Martin Fowler
 - <http://www.youtube.com/watch?v=pqeJFYwnkjE>
 - <http://martinfowler.com/bliki/TechnicalDebt.html>

You have a piece of functionality that you need to add to your system.

You see two ways to do it, one is **quick** to do but is **messy** - you are sure that it will make further changes harder in the **future**. The other results in a **cleaner** design, but will take **longer** to put in place.

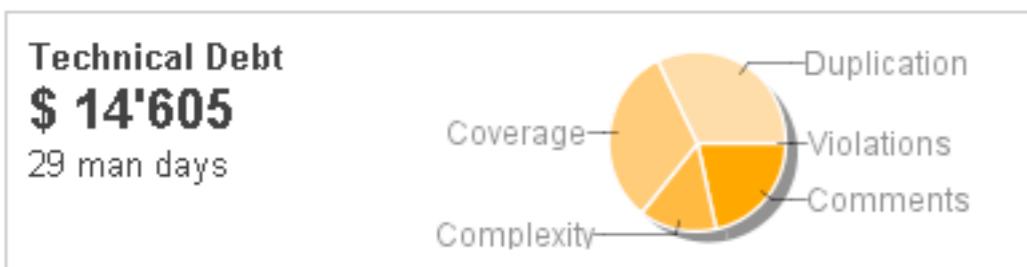
Technical Debt is a wonderful **metaphor** to help us think about this problem. Doing things the quick and dirty way sets us up with a technical debt, which is **similar to a financial debt**.

Like a financial debt, the technical debt incurs **interest payments**, which come in the form of the extra effort that we have to do in future development because of the quick and dirty design choice. We can choose to continue paying the interest, or we can **pay down** the principal by **refactoring** the quick and dirty design into the better design. **Although it costs to pay down the principal, we gain by reduced interest payments in the future.**

Sonar: technical debt

- Concept invented by Ward Cunningham, also described by Martin Fowler
 - <http://www.youtube.com/watch?v=pqeJFYwnkjE>
 - <http://martinfowler.com/bliki/TechnicalDebt.html>

Debt(in man days) = cost_to_fix_duplications + cost_to_fix_violations +
cost_to_comment_public_API + cost_to_fix_uncovered_complexity +
cost_to_bring_complexity_below_threshold



Duplications = cost_to_fix_one_block * duplicated_blocks

Violations = cost_to_fix_oneViolation * mandatory_violations

Comments = cost_to_comment_one_API * public undocumented_api

Coverage = cost_to_cover_one_of_complexity * uncovered_complexity_by_tests (80% of coverage is the objective)

Complexity = cost_to_split_a_method * (function_complexity_distribution >= 8) + cost_to_split_a_class * (class_complexity_distribution >= 60)

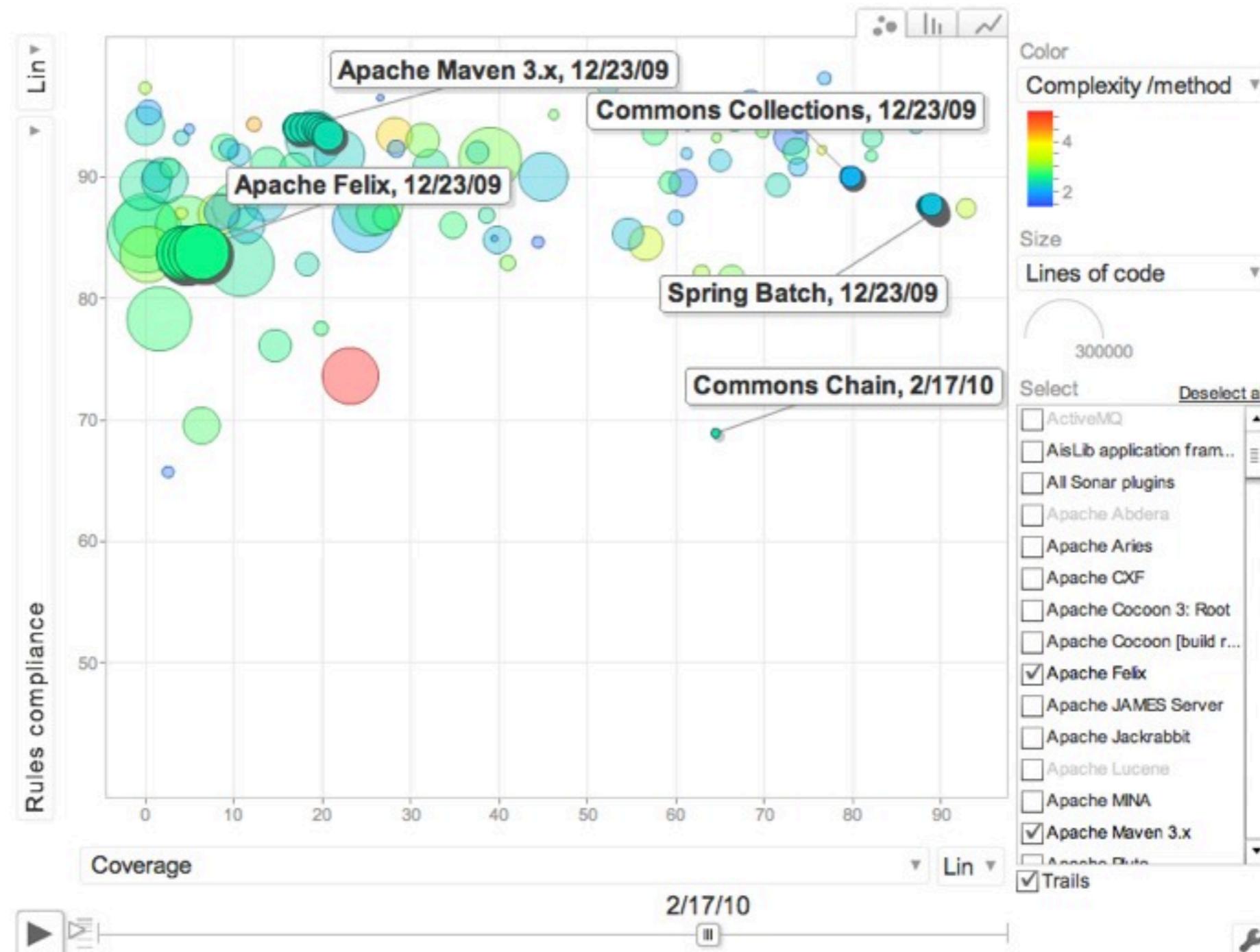
The metaphor also explains why it may be sensible to do the quick and dirty approach.

Just as a business incurs some debt to take advantage of a **market opportunity** developers may incur technical debt to hit an important deadline. The all too common problem is that development organizations let their debt get **out of control** and spend most of their future development effort paying crippling interest payments.

Sonar on open source projects



Sonar on open source projects



Sonar on open source projects

Name		Lines of code	Technical Debt ratio	Coverage	Duplicated lines (%)	Build time
AisLib application framework		12,187	9.6%	38.6%	1.1%	2010-03-13
Tapestry 5 Project		57,213	6.7%	51.8%	0.2%	2010-03-15
Commons Collections		20,901	10.1%	79.8%	3.3%	2010-03-14
MicroEmulator		28,344	11.7%		2.3%	2010-03-14
Apache Jackrabbit		206,604	16.6%	26.4%	4.6%	2010-03-14
Utils		18,585	14.9%	2.8%	3.8%	2010-03-15
javagit		6,127	11.1%	61.2%	0.3%	2010-03-14
Wicket Parent		104,703	9.3%	44.7%	1.1%	2010-03-15
Commons Chain		3,901	14.4%	64.5%	21.9%	2010-03-14
Commons IO		6,779	5.2%	82.0%	2.3%	2010-03-14
Commons SCXML		7,331	9.8%	69.8%	7.2%	2010-03-14
Sonar		31,558	6.2%	68.6%	0.0%	2010-03-17

Sonar on Sonar



Sonar on Sonar

Priority Category

Priority	Category	Count
Blocker		3
Critical		37
Major		271
Minor		242
Info		20

Rule

Rule	Count
Header	3
Security - Array is stored directly	15
Bad practice - Class defines compareTo(...) and uses Object.equals()	6
Dodgy - Redundant nullcheck of value known to be non-null	3
Bad practice - equals() method does not check for null argument	2
Bad practice - equals method fails for subtypes	2

Sonar :: Server 8 org.sonar.server.startup 5 CustomBarRenderer 2

Sonar :: Plugin API 5 org.sonar.server.charts.deprecated 2 GwtPublisher 2

Sonar :: Batch 1 org.sonar.api.database.model 2 MeasureData 2

Sonar :: Plugins :: CPD 1 org.sonar.api.resources 2 RulesManager 1

Sonar :: Plugins :: GWT 1 org.sonar.plugins.cpd 1 RuleLoader 1

Sonar :: Plugins :: QualityProfile 1 org.sonar.api.rules 1 QualityProfileLoader 1

Path: Any priority » Security - Array is stored directly [clear](#) »

[org.sonar.server.startup.GwtPublisher](#) [New window](#)

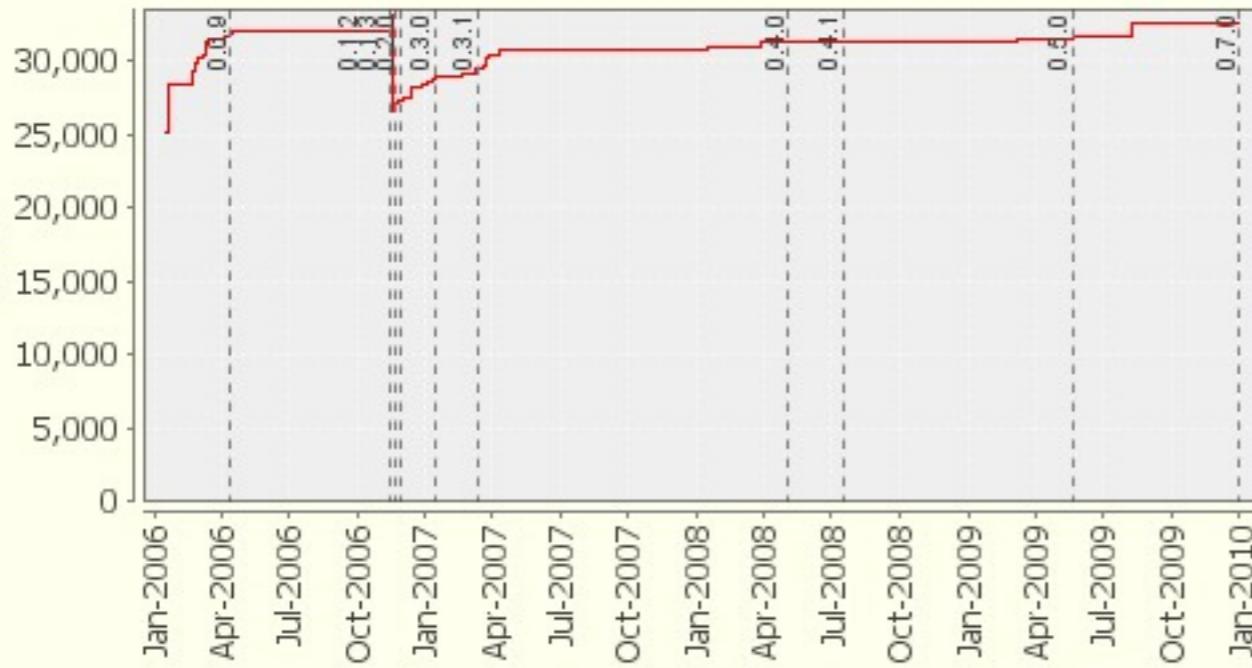
Blame	Coverage	Dependencies	Duplications	LCOM4	Sources	Violations
0	2	2	0	0		

Filter: Security - Array is stored directly (2) Expand:

```
44 private Configuration configuration;
45 private GwtExtension[] extensions = null;
46 private File outputDir = null;
47
48 public GwtPublisher(GwtExtension[] extensions, Configuration configuration) {
    Security - Array is stored directly : The user-supplied array 'extensions' is stored directly.
49     this.extensions = extensions;
50     this.configuration = configuration;
51 }
52
53 protected GwtPublisher(GwtExtension[] extensions, File outputDir) {
    Security - Array is stored directly : The user-supplied array 'extensions' is stored directly.
54     this.extensions = extensions;
55     this.outputDir = outputDir;
56 }
57
58 protected GwtPublisher() {
```

Tool: StatSVN

StatSVN: Lines of Code



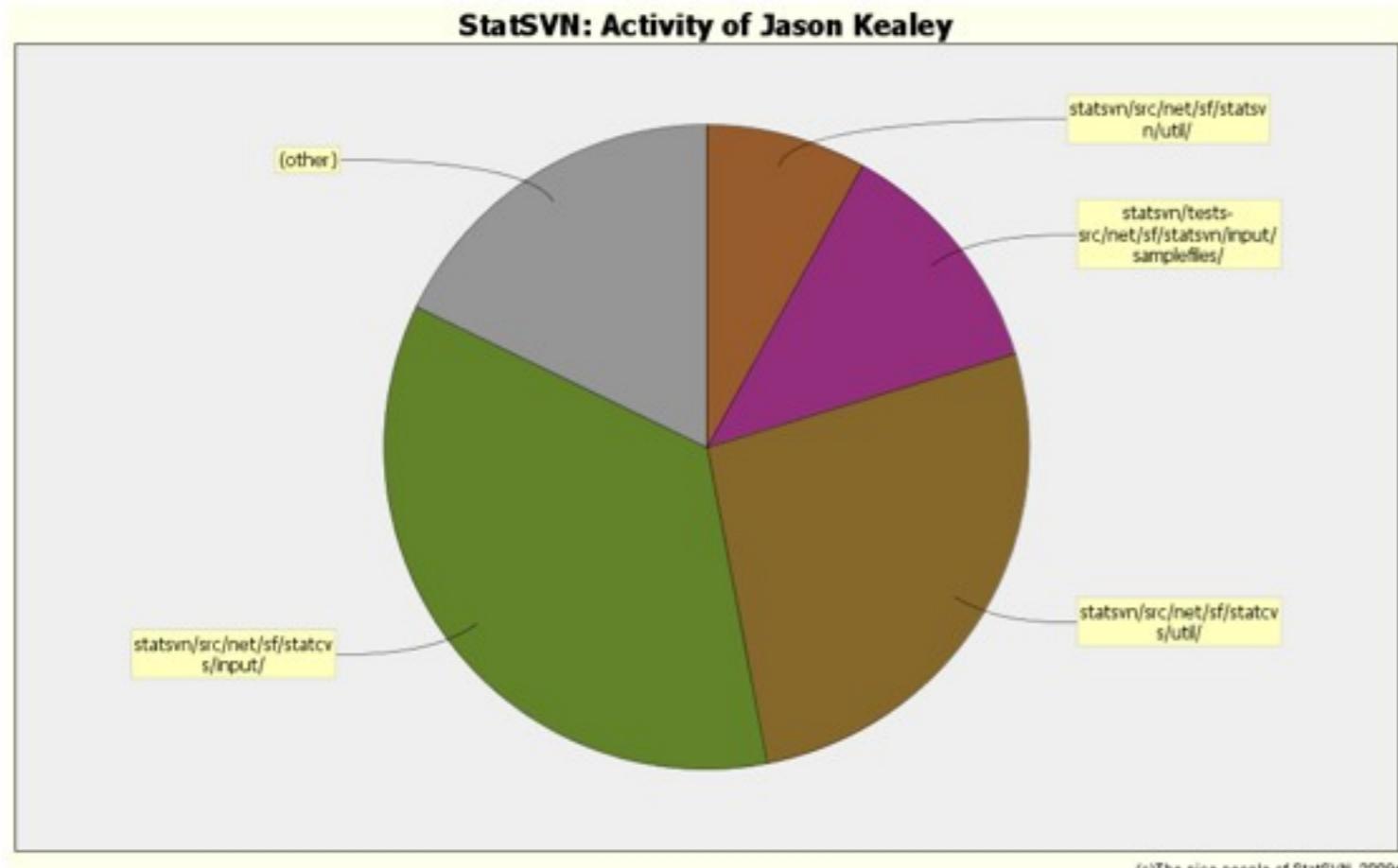
Type	Files	LOC	LOC per file
*.info	1 (0.6%)	18122 (56.6%)	18122.0
*.java	60 (33.5%)	10090 (31.5%)	168.1
*.xml	16 (8.9%)	1686 (5.3%)	105.3
*.properties	15 (8.4%)	526 (1.6%)	35.0
*.bat	44 (24.6%)	474 (1.5%)	10.7
*.prefs	3 (1.7%)	329 (1.0%)	109.6
*.txt	7 (3.9%)	206 (0.6%)	29.4
*.rss	1 (0.6%)	179 (0.6%)	179.0
*.css	1 (0.6%)	146 (0.5%)	146.0
*.html	2 (1.1%)	93 (0.3%)	46.5
Others	5 (2.8%)	164 (0.5%)	32.8
Non-Code Files	24 (13.4%)	0 (0.0%)	0.0
Totals	179 (100.0%)	32015 (100.0%)	178.8

The nice people of StatSVN, 2009+

Files With Most Revisions

File	Revisions
statsvn/site/changes.xml	54
statsvn/src/net/sf/statsvn/Main.java	41
statsvn/project.xml	38
statsvn/src/net/sf/statcvs/input/SvnLogFileParser.java	34
statsvn/site/index.xml	31
statsvn/project.properties	31
statsvn/src/net/sf/statsvn/util/SvnDiffUtils.java	27
statsvn/src/net/sf/statsvn/input/SvnLogFileParser.java	26
statsvn/src/net/sf/statcvs/input/SvnXmlLogFileHandler.java	24
statsvn/.classpath	22
statsvn/build.xml	21
statsvn/src/net/sf/statcvs/Main.java	20
statsvn/src/net/sf/statcvs/input/Builder.java	19
statsvn/src/net/sf/statsvn/input/Builder.java	19
statsvn/src/net/sf/statcvs/util/SvnDiffUtils.java	19
statsvn/src/net/sf/statcvs/input/FileBuilder.java	18
statsvn/src/net/sf/statcvs/util/SvnInfoUtils.java	17
statsvn/src/net/sf/statsvn/input/FileBuilder.java	17
statsvn.bat	17
statsvn/src/net/sf/statcvs/output/ConfigurationOptions.java	16

Tool: StatSVN



Number of Developers: 4

Author	Author Id	Changes	Lines of Code	Lines per Change
Jason Kealey	jkealey	379 (16.1%)	14310 (40.1%)	37.7
Benoit Xhenseval	benoitx	1620 (68.7%)	14014 (39.3%)	8.6
Jean-Philippe Daigle	jpdraigle	213 (9.0%)	3758 (10.5%)	17.6
Gunter Mussbacher	gunterm	147 (6.2%)	3576 (10.0%)	24.3
Totals		2359 (100.0%)	35658 (100.0%)	15.1

<http://www.statsvn.org/>

Developer of the Month

Month	Author	Lines	Tweet This
January 2010	Benoit Xhenseval	18	TWEET THIS
December 2009	Jason Kealey	16	TWEET THIS
August 2009	Jason Kealey	3386	TWEET THIS
June 2009	Benoit Xhenseval	20	TWEET THIS
May 2009	Benoit Xhenseval	293	TWEET THIS
April 2009	Benoit Xhenseval	17	TWEET THIS
March 2009	Benoit Xhenseval	599	TWEET THIS
July 2008	Benoit Xhenseval	2	TWEET THIS
June 2008	Benoit Xhenseval	742	TWEET THIS
May 2008	Benoit Xhenseval	12	TWEET THIS
April 2008	Benoit Xhenseval	153	TWEET THIS
March 2008	Benoit Xhenseval	1657	TWEET THIS
January 2008	Jason Kealey	175	TWEET THIS
July 2007	Benoit Xhenseval	16	TWEET THIS
June 2007	Jean-Philippe Daigle	43	TWEET THIS
May 2007	Jason Kealey	1	TWEET THIS
April 2007	Benoit Xhenseval	557	TWEET THIS
March 2007	Benoit Xhenseval	1387	TWEET THIS
February 2007	Benoit Xhenseval	141	TWEET THIS
January 2007	Benoit Xhenseval	1205	TWEET THIS
December 2006	Benoit Xhenseval	1730	TWEET THIS
November 2006	Benoit Xhenseval	5310	TWEET THIS
August 2006	Jason Kealey	27	TWEET THIS
April 2006	Jason Kealey	1354	TWEET THIS
March 2006	Jason Kealey	4862	TWEET THIS
February 2006	Jason Kealey	3309	TWEET THIS
January 2006	Jean-Philippe Daigle	3292	TWEET THIS

References

- **Mining Software Repositories community**
 - <http://msr.uwaterloo.ca/msr2010/>
- **Tools**
 - **Sonar:** <http://sonar.codehaus.org>
 - **StatSVN:** <http://www.statsvn.org>
 - **Bugzilla:** <http://www.bugzilla.org>
- **Visualization**
 - <http://code.google.com/apis/charttools>
 - <http://prefuse.org>, <http://flare.prefuse.org>
 - <http://processing.org>