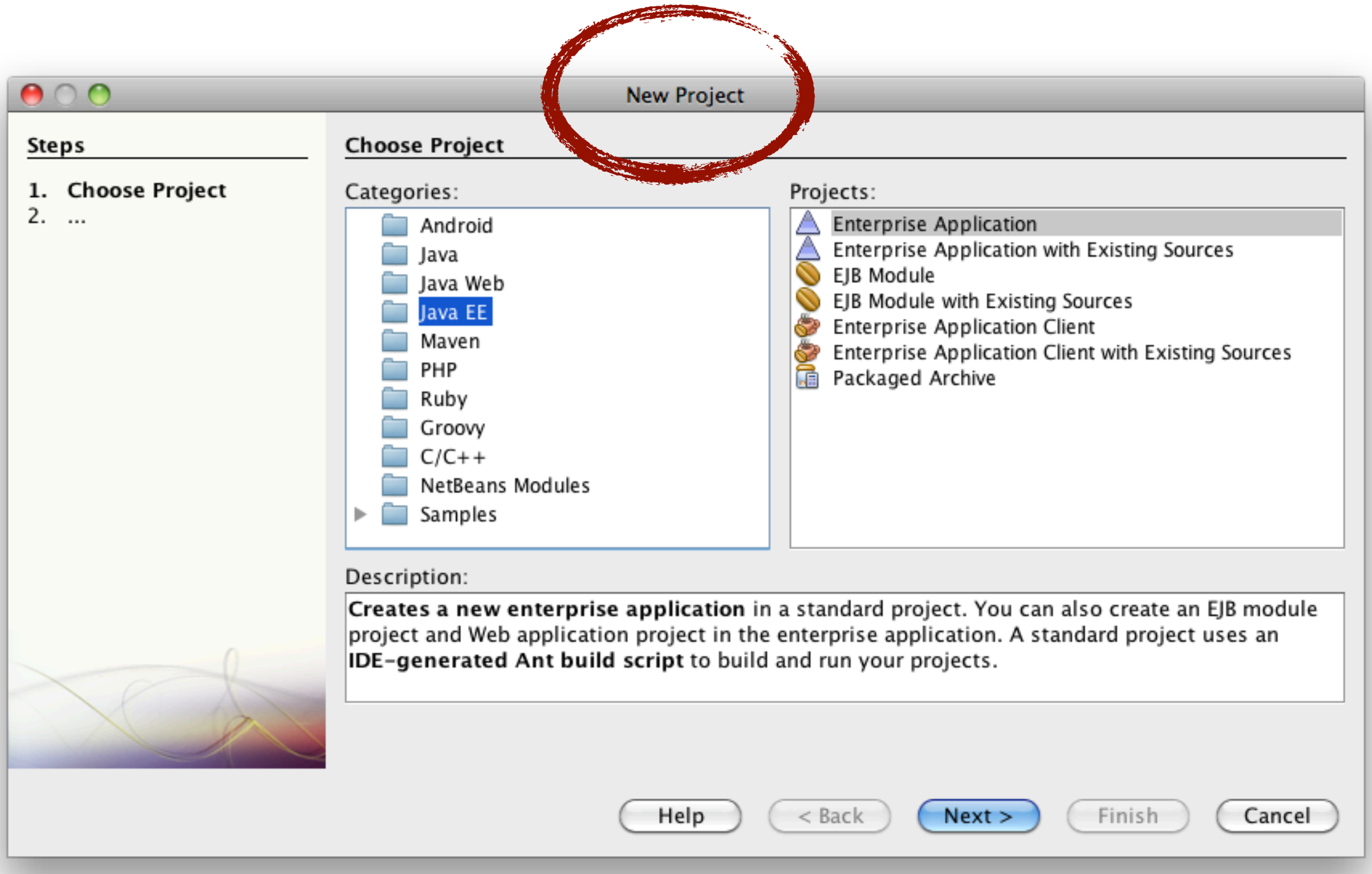
Introduction to Software Evolution

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

MSE | MASTER OF SCIENCE IN ENGINEERING

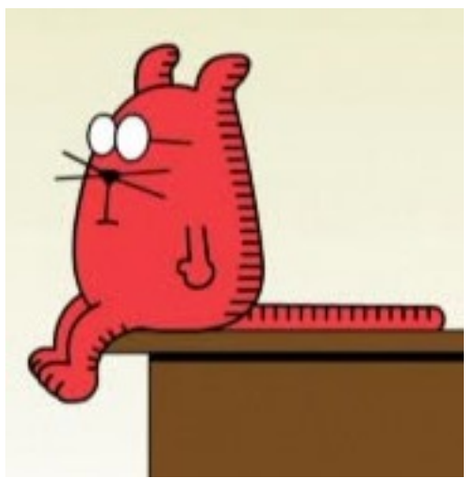
Schedule

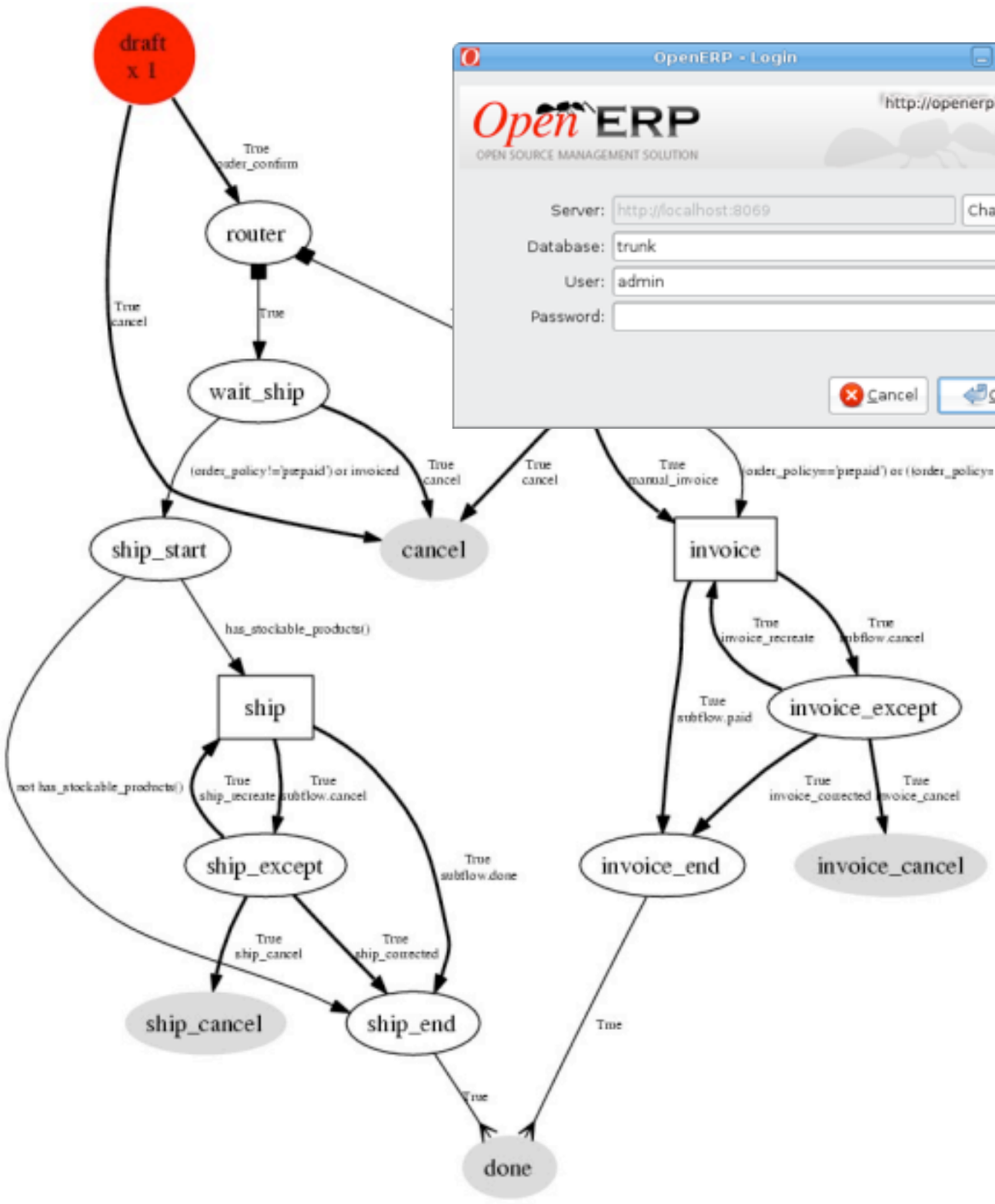
- **Lecture (15h - 15h45)**
  - **Introduction**: the scientific and engineering fields of **software evolution**
  - **Foundations**: Laws of Software Evolution, Software Aging
  - **Tools**: code quality tools, software repository mining, etc.
- **Group work (15h45 - 16h30)**
  - Your experience and perception of agile practices
- **Lab (16h30 - 17h30)**
  - **Introduction to maven,** OR
  - **Maven and Sonar**


New Project
Steps
1. Choose Project
2. ...
Choose Project
Categories:
Android
Java
Java Web
Java EE
Maven
PHP
Ruby
Groovy
C/C++
NetBeans Modules
Samples
Projects:
Enterprise Application
Enterprise Application with Existing Sources
EJB Module
EJB Module with Existing Sources
Enterprise Application Client
Enterprise Application Client with Existing Sources
Packaged Archive
Description:
Creates a new enterprise application in a standard project. You can also create an EJB module project and Web application project in the enterprise application. A standard project uses an IDE-generated Ant build script to build and run your projects.
Help
< Back
Next >
Finish
Cancel

MSE | MASTER OF SCIENCE IN ENGINEERING

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



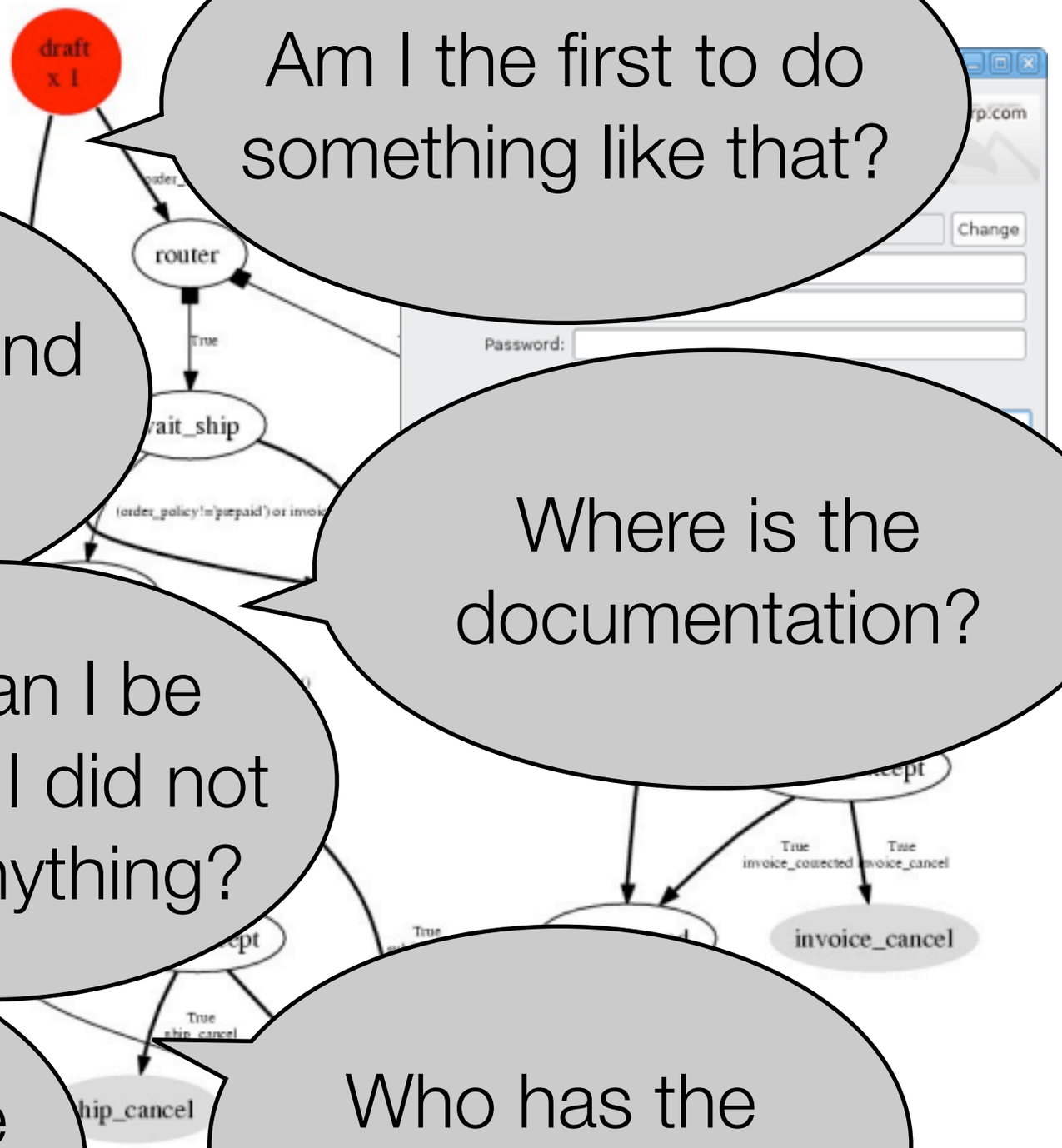


http://doc.openerp.com/developer/2_4_module_development/4_1_module_structure.html

How was the ERP system designed?

How can I extend the ERP?

Am I the first to do something like that?



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

Where is the documentation?

How do I release my new feature?

Who has the knowledge?

http://doc.openerp.com/developer/2_4_module_development/4_1_module_structure.html

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 — psu.edu [PDF]
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 ... - Related articles - BL Direct - All 19 versions - Import into BibTeX

[PDF] **Programs, life cycles, and laws of software evolution** — bth.se [PDF]
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

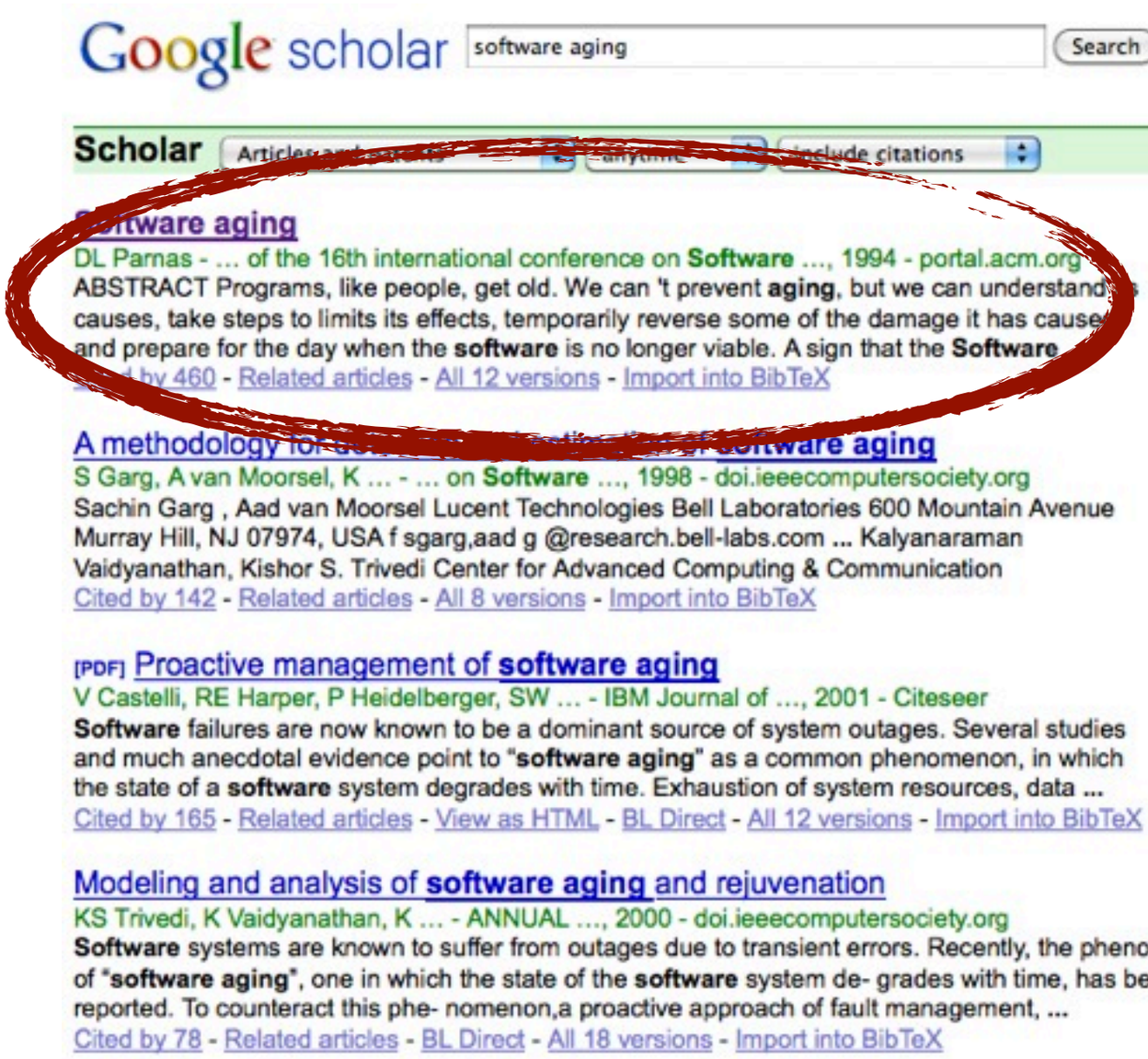
[PDF] **Evolution** in the ... software ... — psu.edu [PDF]
MW Godfrey, Q Tu - 16th IEEE International Conference on **Software** ..., 2000 - Citeseer
Most studies of **software evolution** have been performed on systems developed within a single company using tradi- tional management techniques. With the widespread avail- ability of several large **software** systems that have been de- veloped using an "open source" development ...
Cited by 335 - Related articles - View as HTML - All 47 versions - Import into BibTeX

Metrics and laws of software evolution-the nineties view — psu.edu [PDF]
... , 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) 17 1 594 82 15 e-mail: (mml,jcfl,pdw)@doc.ic.ac.uk URL:http:Nwww-dse.doc.ic.ac.uk/-mml/feast1 ... M Turski Institute of Informatics Warsaw University Warsaw ...
Cited by ... - Related articles - All 36 versions - Import into BibTeX

Software maintenance and evolution: a roadmap — psu.edu [PDF]
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

Laws of software ... invited — psu.edu [PDF]
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 and, over a period of twenty years, to formulation of eight Laws o] So]tware Evo- lution. The FEAST project recently initiated (see sections 4 - 6 ...
Cited by 237 - Related articles - BL Direct - All 32 versions - Import into BibTeX

Research on Software Evolution


Google scholar software aging Search Advanced Scholar Search Scholar Preferences

Scholar Articles and patents 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 limits 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 detection and estimation of **software aging** ncsu.edu [PS]
S Garg, A van 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** psu.edu [PDF]
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 psu.edu [PDF]
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

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

Empirical studies and validation

Basic research

How do we model a software system and its evolution?

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

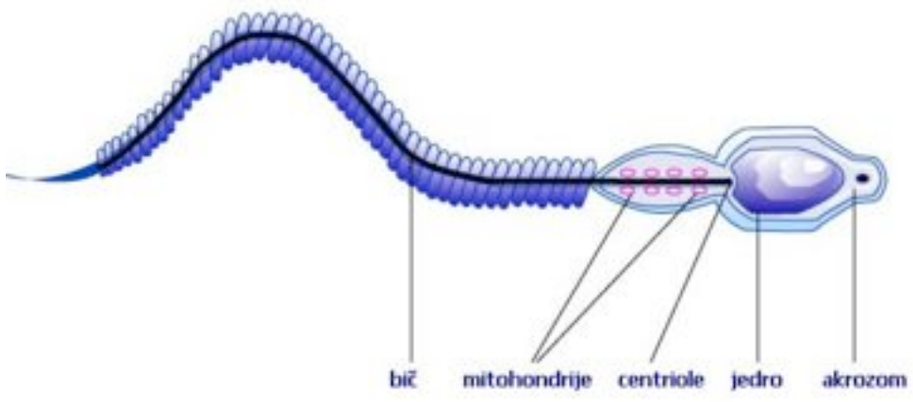
History and Challenges of Software Evolution

- **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!)**

Tom Mens, Introduction and Roadmap: History and Challenges of Software Evolution, Software Evolution, ISBN 978-3-540-76439-7

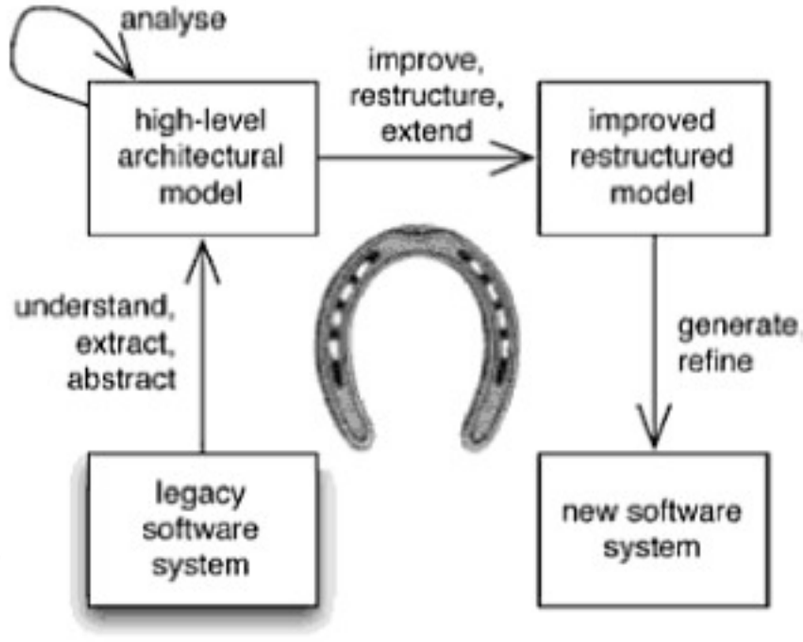
History and Challenges of Software Evolution

- **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.



History and Challenges of Software Evolution

- **Reverse engineering**
  - Activity needed when **trying to understand the architecture or behavior** of a large software system, when the only reliable information is the **source code**.
  - Aims at **building higher-level, more abstract, software models** from the source code.
- **Re-engineering**
  - Activity needed when we are confronted with **legacy systems**. In other words, systems that are still **valuable**, but are **difficult to maintain**.
  - Aims at producing a new system that is more evolvable.



Horseshoe model: reverse engineering, restructuring, forward engineering

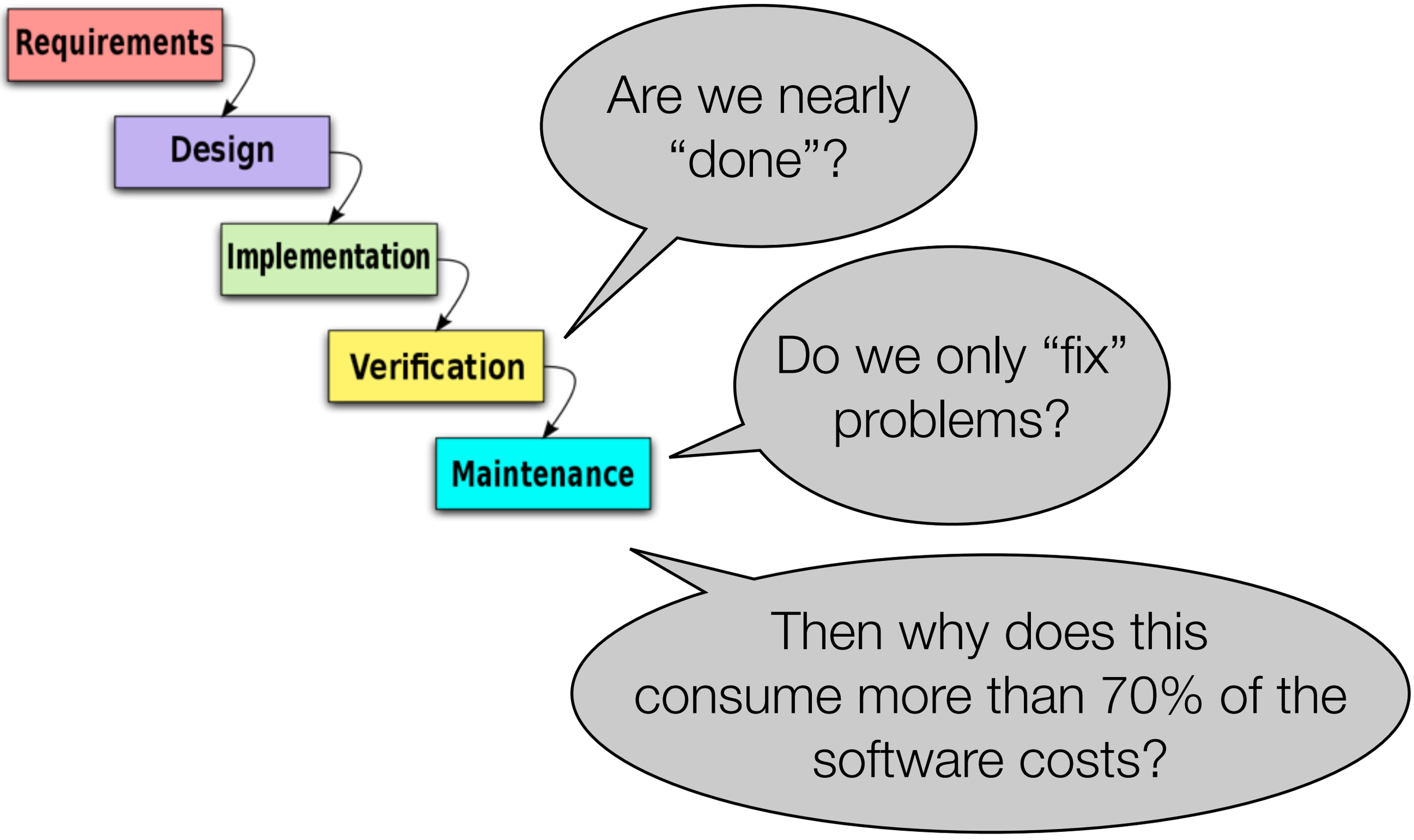
Software Maintenance

- 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.

MSE | MASTER OF SCIENCE IN ENGINEERING

Management Applications
H. Morgan Editor

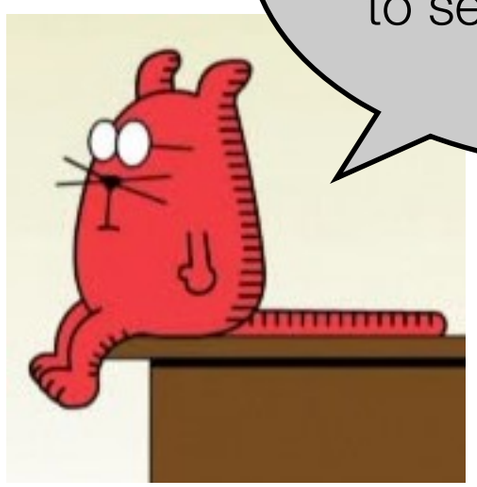
Characteristics of Application Software Maintenance

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.

Communications of the ACM
June 1978 Volume 21 Number 6

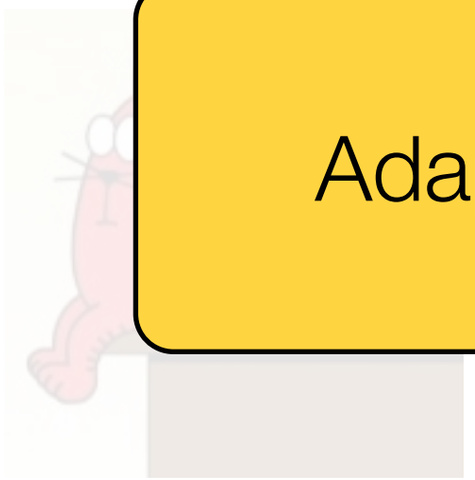
Software Maintenance



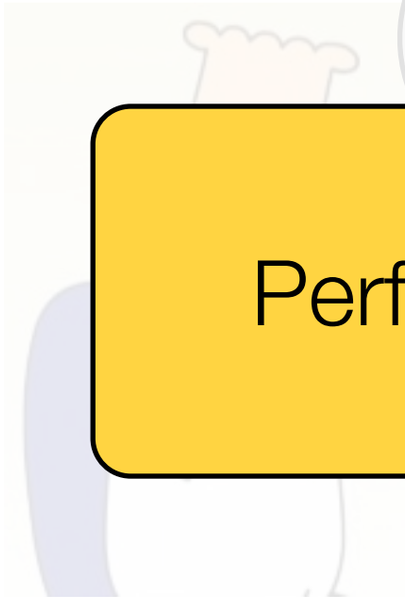
Software Maintenance

Corrective

Preventive



Adaptive



Perfective

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.

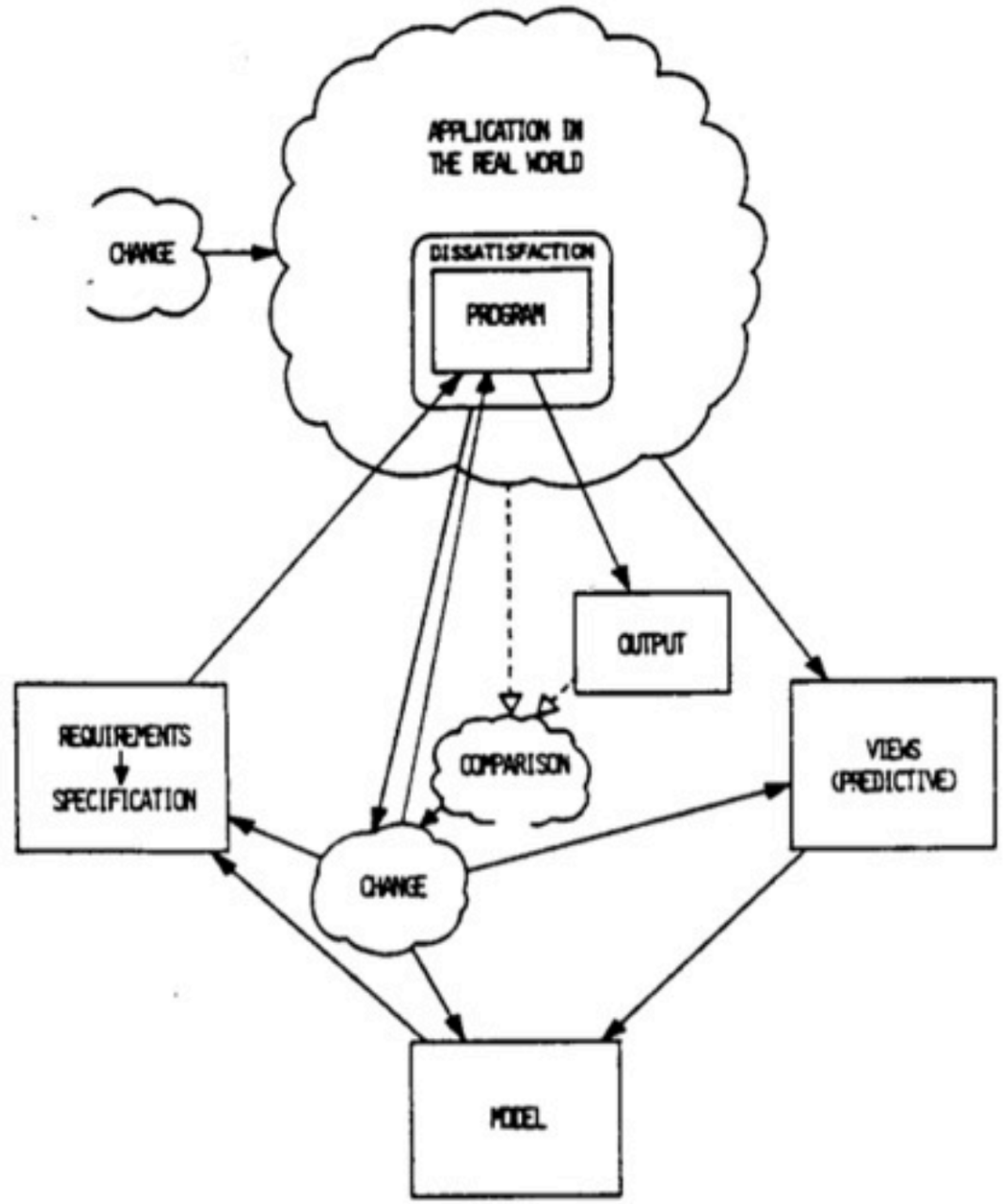
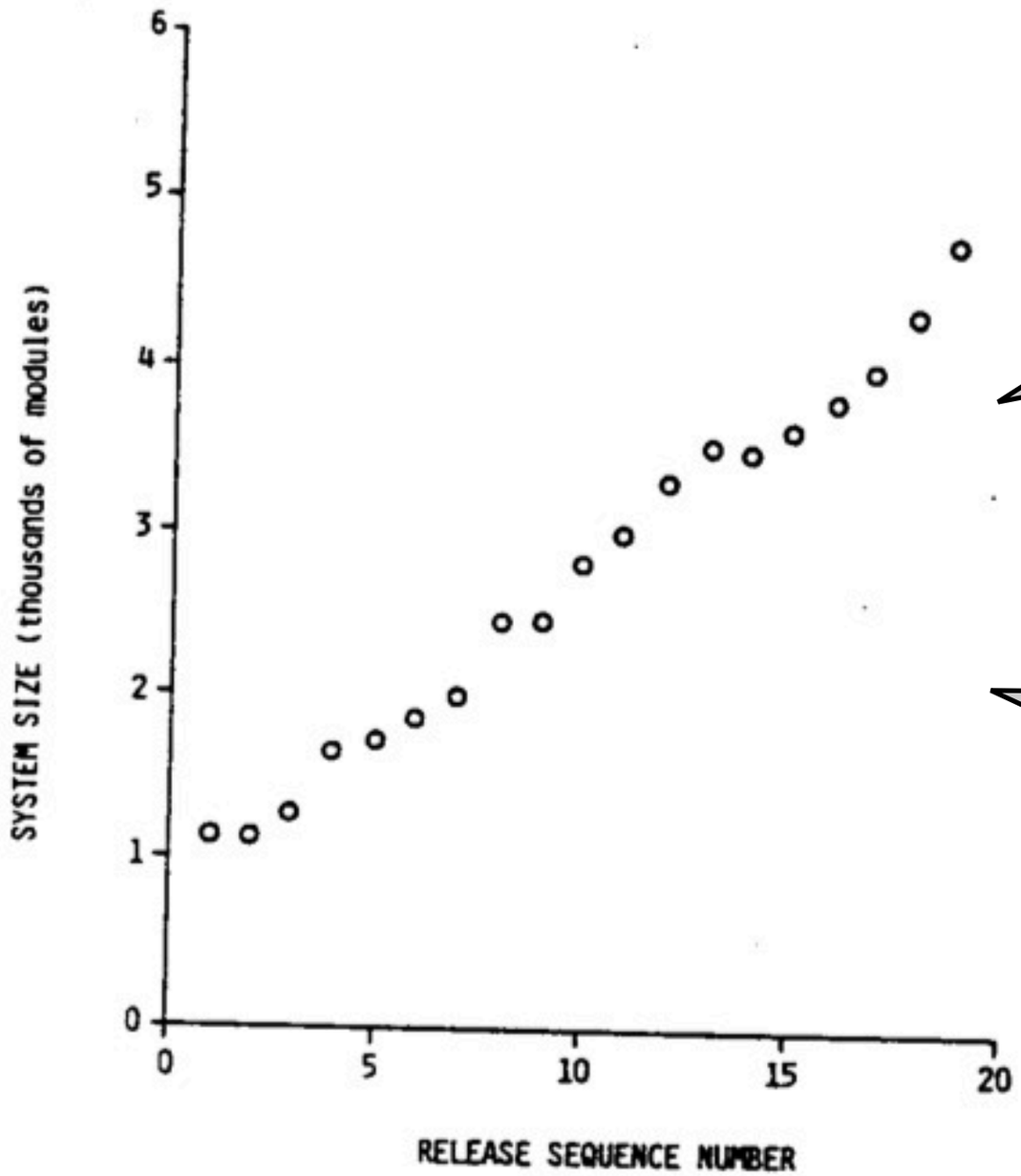


Fig. 4. *E*-programs.

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.**

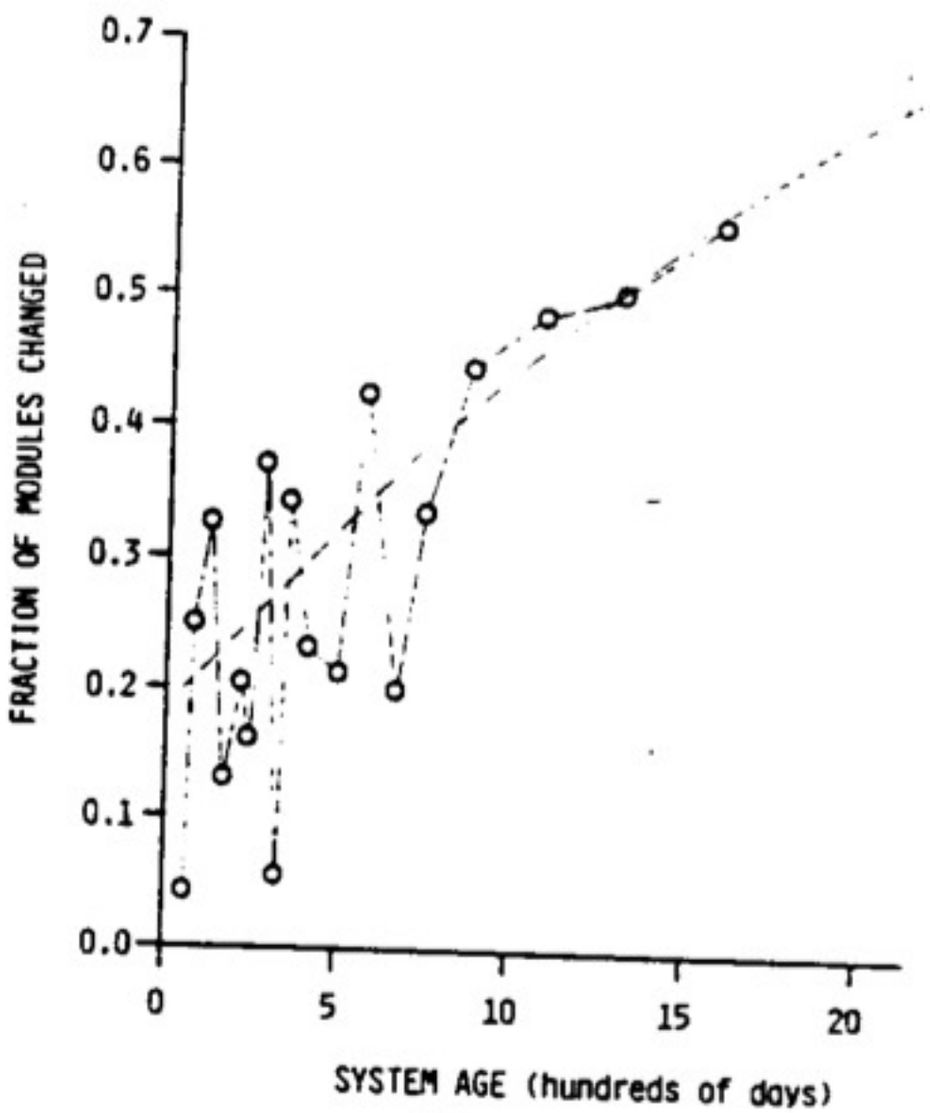
Quantitative Studies



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!

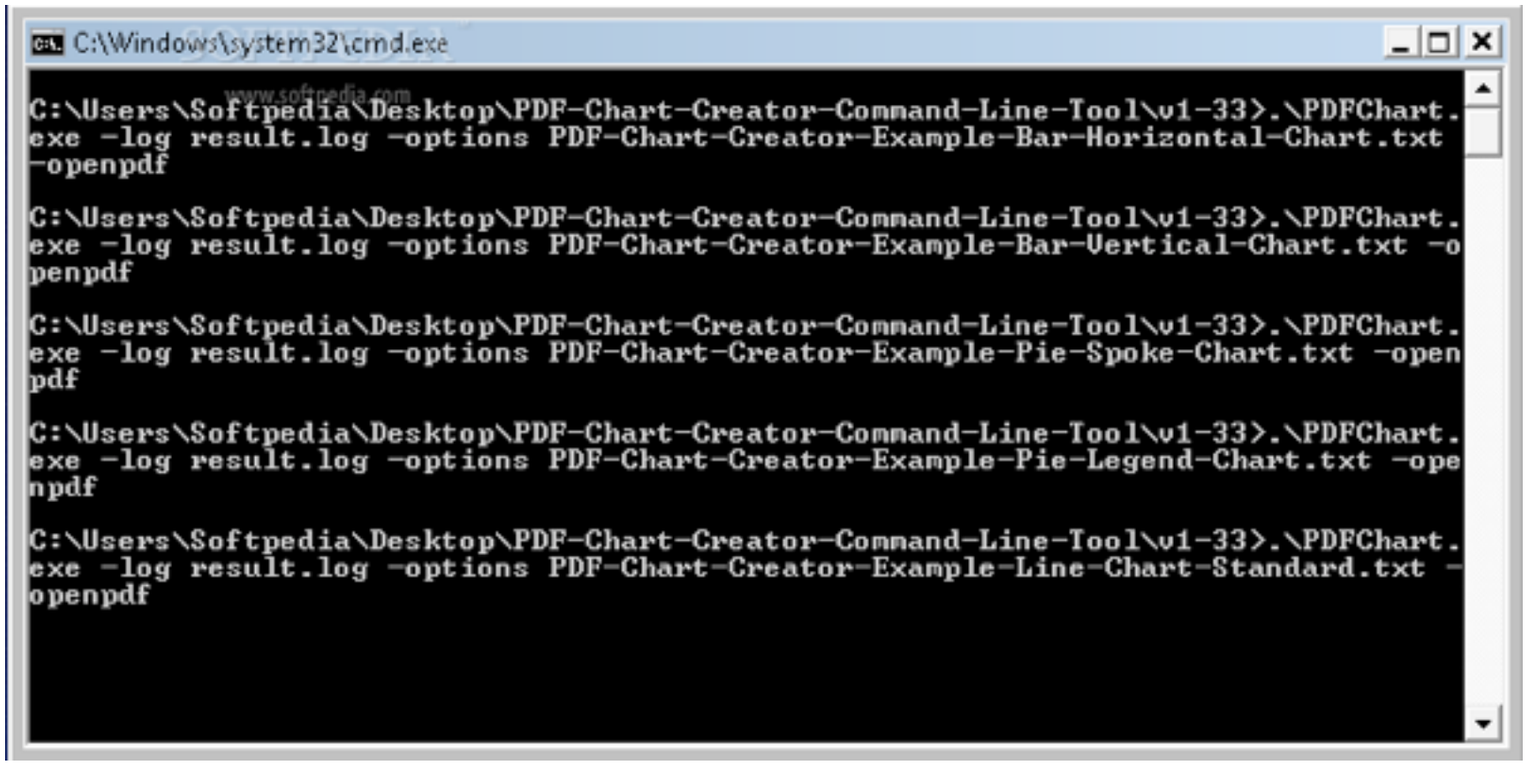
No.	Brief Name	Law
I 1974	Continuing Change	*E*-type systems must be continually adapted else they become progressively less satisfactory.
II 1974	Increasing Complexity	As an *E*-type system evolves its complexity increases unless work is done to maintain or reduce it.
III 1974	Self Regulation	*E*-type 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 *E*-type system is invariant over product lifetime.
V 1980	Conservation of Familiarity	As an *E*-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 *E*-type systems must be continually increased to maintain user satisfaction over their lifetime.
VII 1996	Declining Quality	The quality of *E*-type systems will appear to be declining unless they are rigorously maintained and adapted to operational environment changes.
VIII 1996	Feedback System (first stated 1974, formalised as law 1996)	*E*-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.

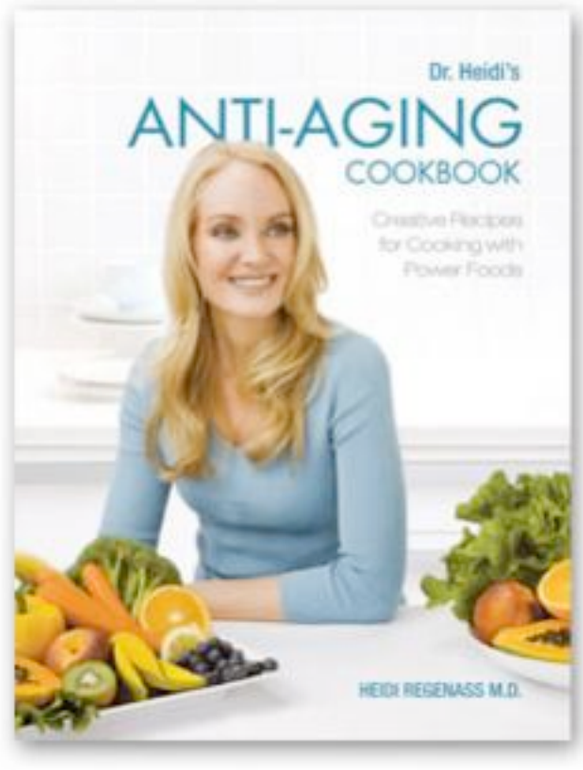
Software Aging

“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





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

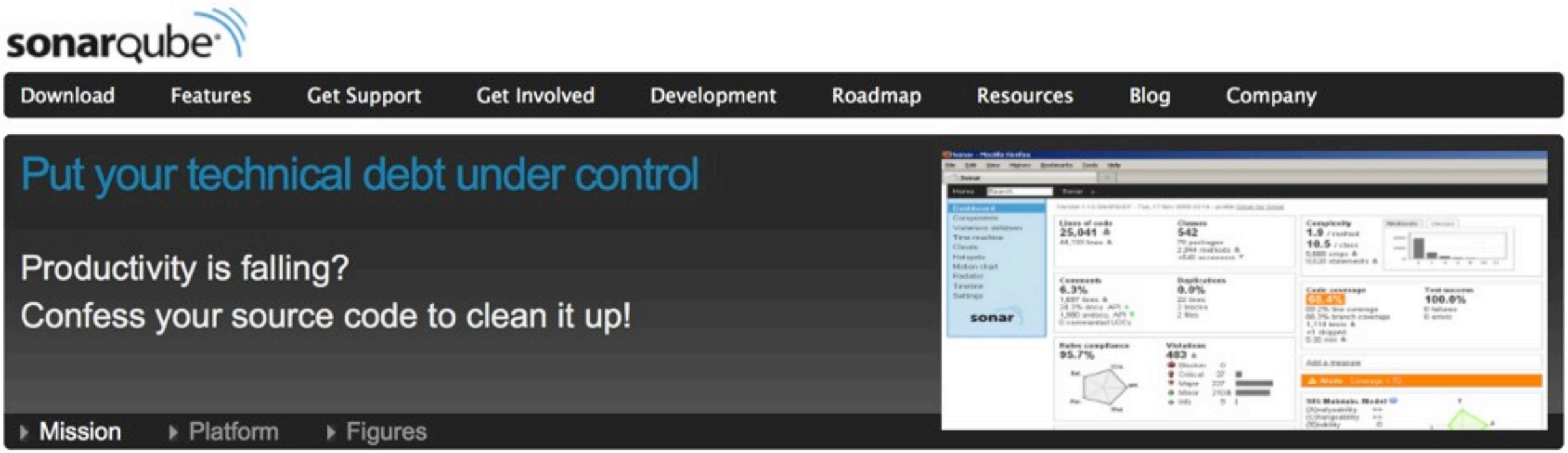
- The **symptoms** of software aging mirror those of human aging:
  - Owners of aging software find it increasingly **hard to keep up** with the market and lose customers to newer products (“weight gain”)
  - Aging software often **degrades in its performance** as a result of a gradually deteriorating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

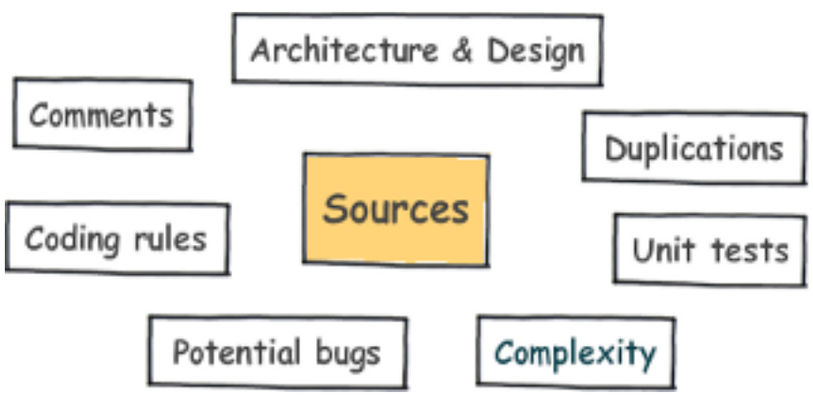
Reducing the costs of software aging

- **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

Tools

Tool: SonarQube





http://www.sonarqube.org/

SonarQube: 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.*

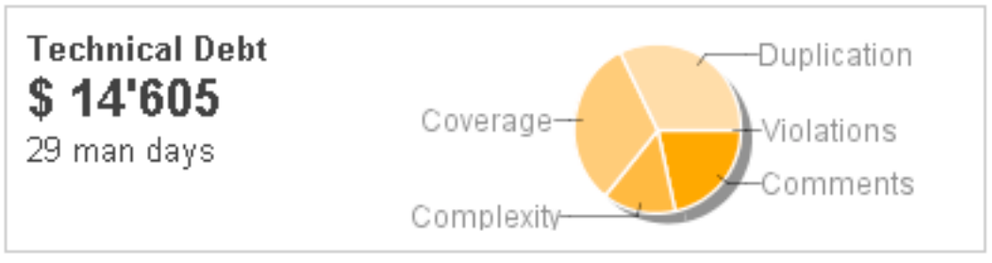
*Technical Debt is a wonderful **metaphor** to help us think about this problem. Doing things the quick and dirty way sets us up with a technical debt, which is **similar to a financial debt**.*

*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.***

SonarQube: 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



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.*

Duplications = cost_to_fix_one_block * duplicated_blocks

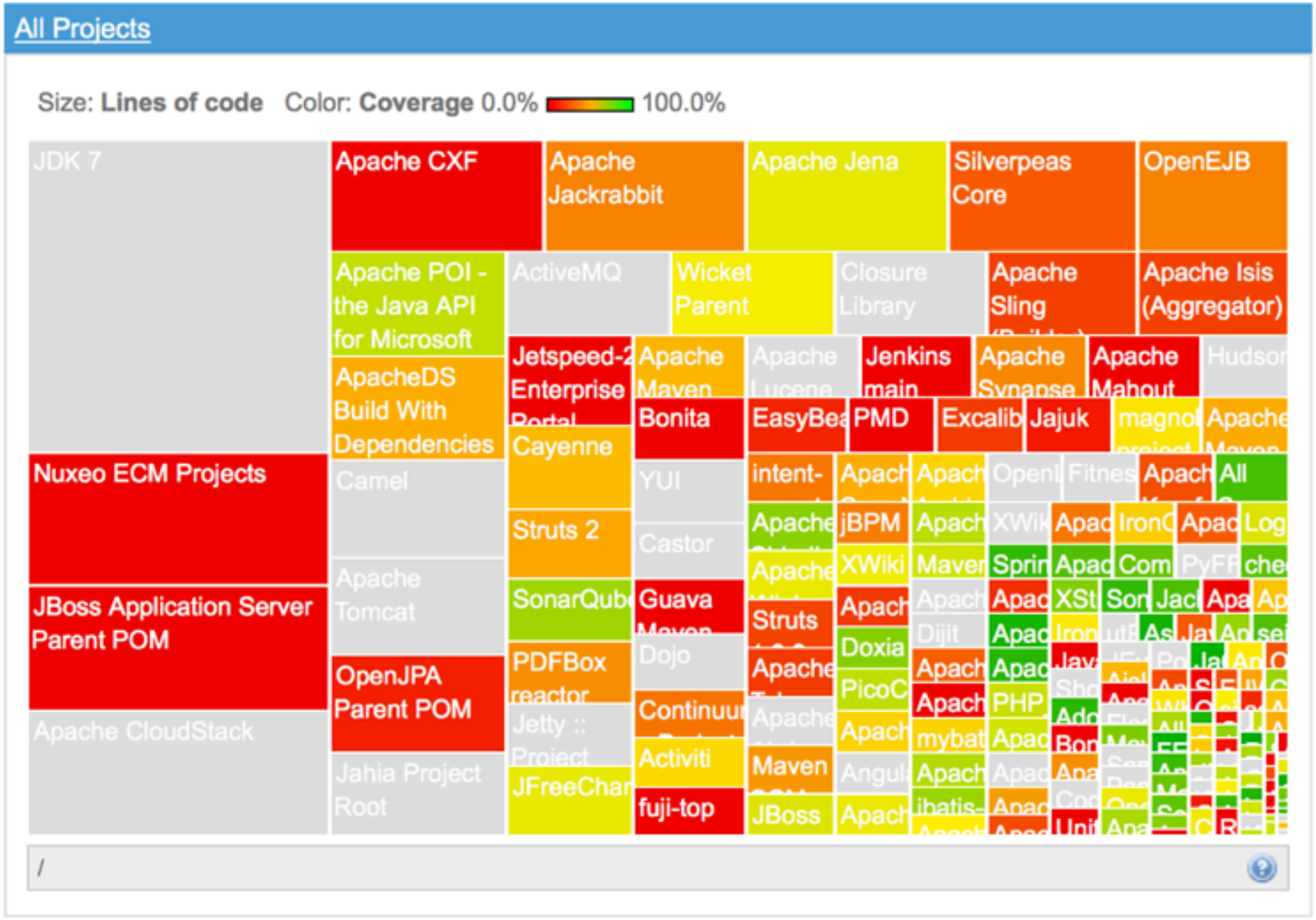
Violations = cost_to fix_one_violation * mandatory_violations

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)

SonarQube on open source projects



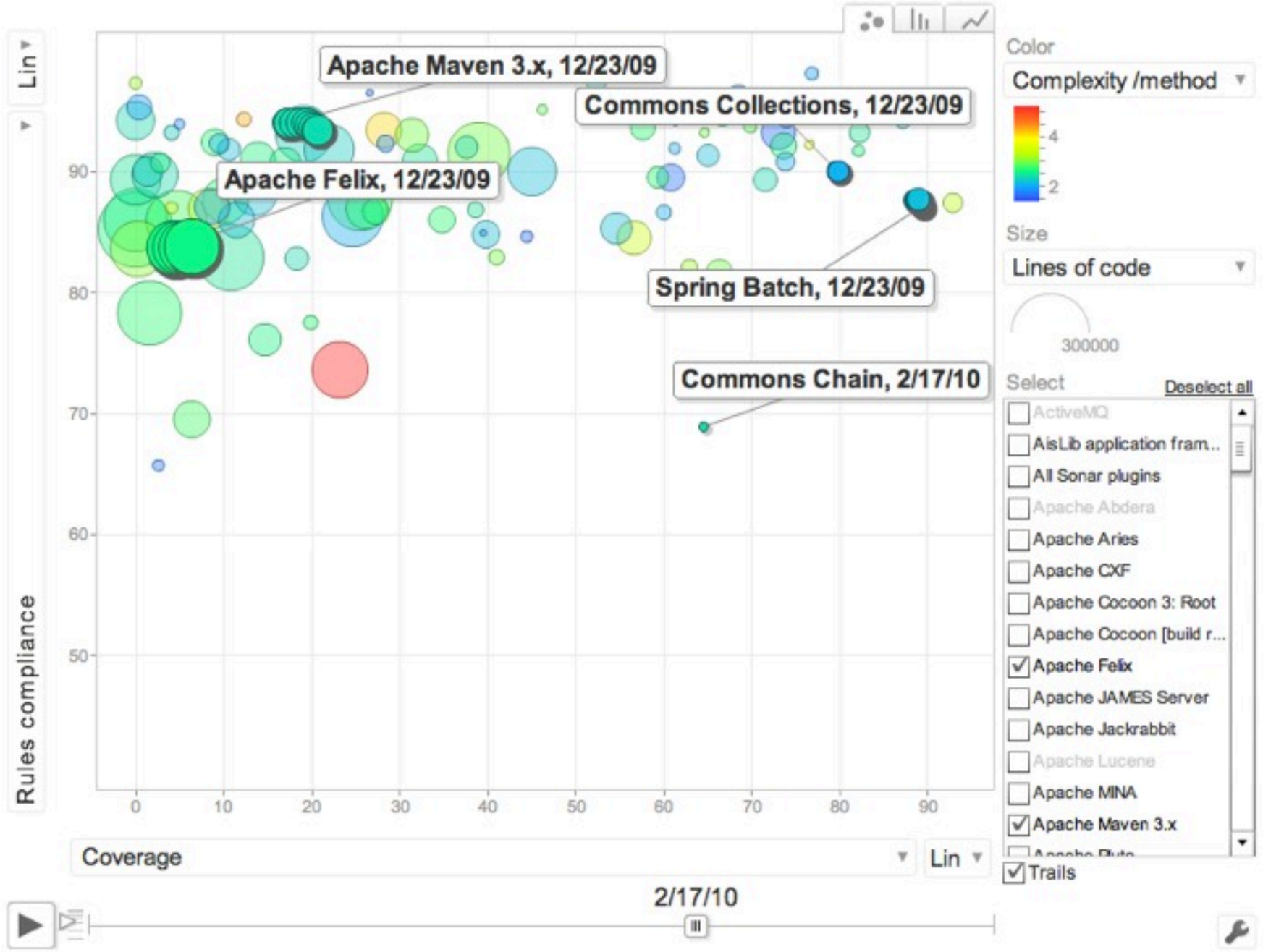
http://nemo.sonarqube.org/

SonarQube 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
	Unitils	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

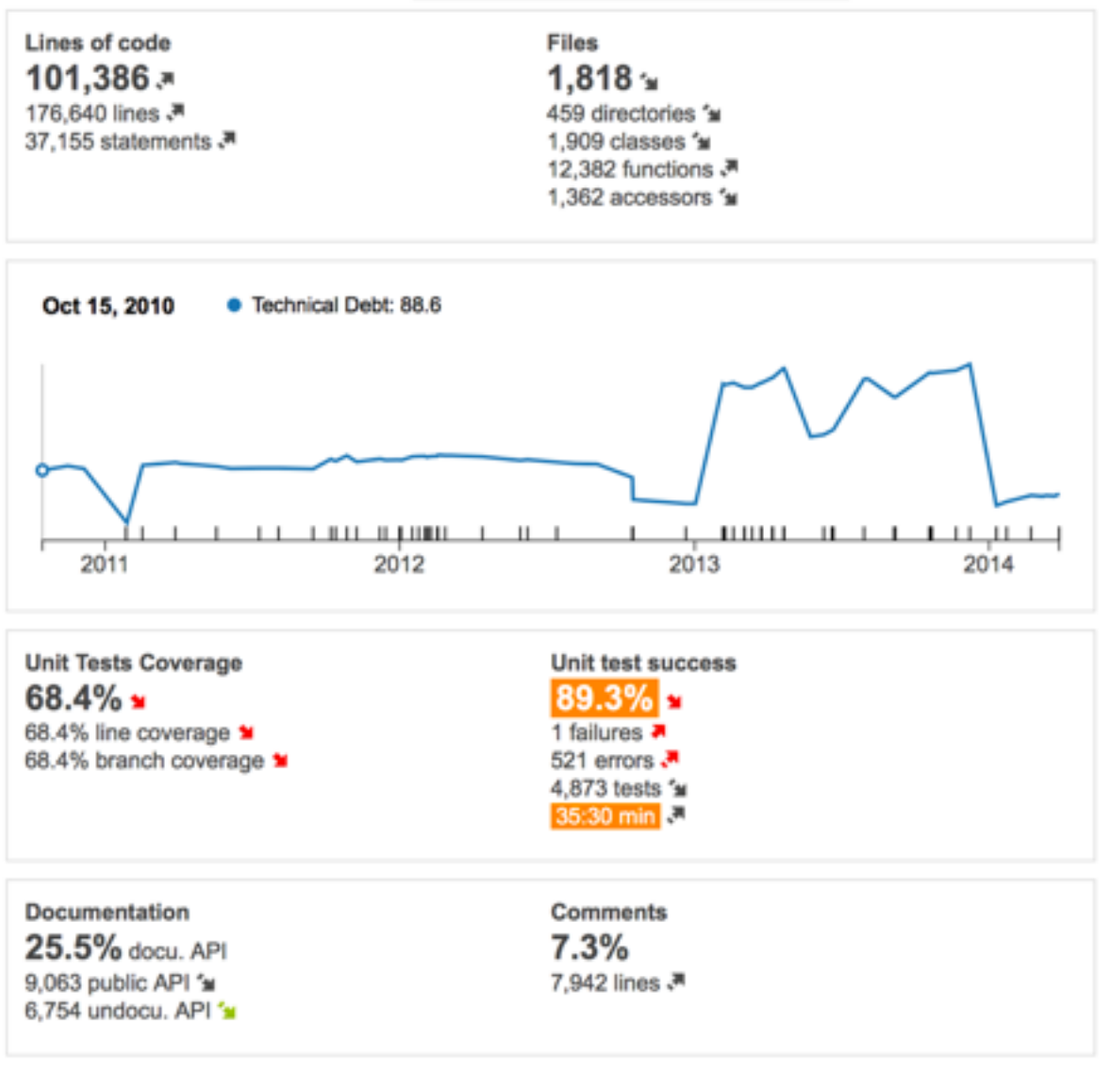
http://nemo.sonarsource.org/

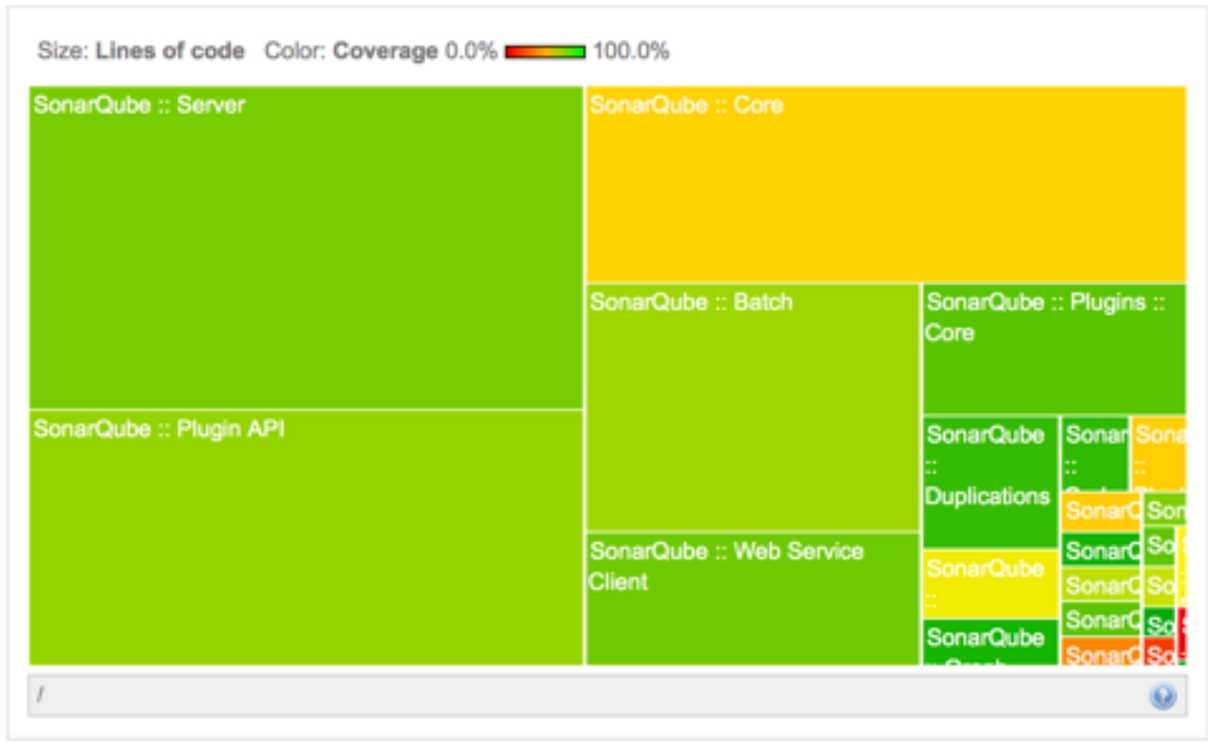
SonarQube on open source projects

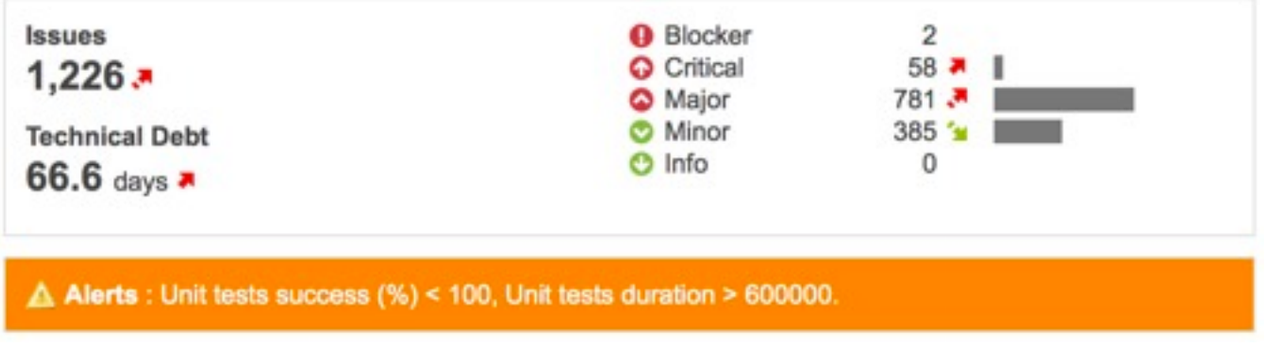


http://nemo.sonarqube.org/

SonarQube on SonarQube







SonarQube on SonarQube

Severity

Blocker	2
Critical	58
Major	781
Minor	385
Info	0

Rule

Exception handlers should provide some context and preserve the original exception	49
Methods named "equals" should override Object.equals(Object)	6
System.exit(...) and Runtime.getRuntime().exit(...) should not be called	1
"equals(Object obj)" should be overridden along with the "compareTo(T obj)" method	1
String literals should not be duplicated	1

SonarQube :: Server	22
SonarQube :: Plugin API	11
SonarQube :: Core	10
SonarQube :: Testing Harness	2
SonarQube :: Web Service Client	2
SonarQube :: Batch	1

src/main/java/org/sonar/server/issue	10
src/main/java/org/sonar/api/utils	4
src/main/java/org/sonar/core/plugins	2
src/main/java/org/sonar/server/charts/deprecated	2
src/main/java/org/sonar/server/debt	2
src/main/java/org/sonar/core/persistence	2

InternalRubyIssueService.java	10
DateUtils.java	2
PluginClassloaders.java	2
LocalizedMessages.java	2
WebServiceEngine.java	2
DebtModelXMLExporter.java	2

SonarQube / SonarQube :: Core

src/main/java/org/sonar/core/plugins/PluginClassloaders.java

Coverage | Duplications | **Issues** | Metrics | Source | Raw

10 issues | Blocker: **0** | Critical: **2** | Major: **3** | Minor: **5** | Info: **0** | Technical Debt: **0.3 days**

Full source | Time changes... | Exception handlers should provide some context and preserve the original exception (2)

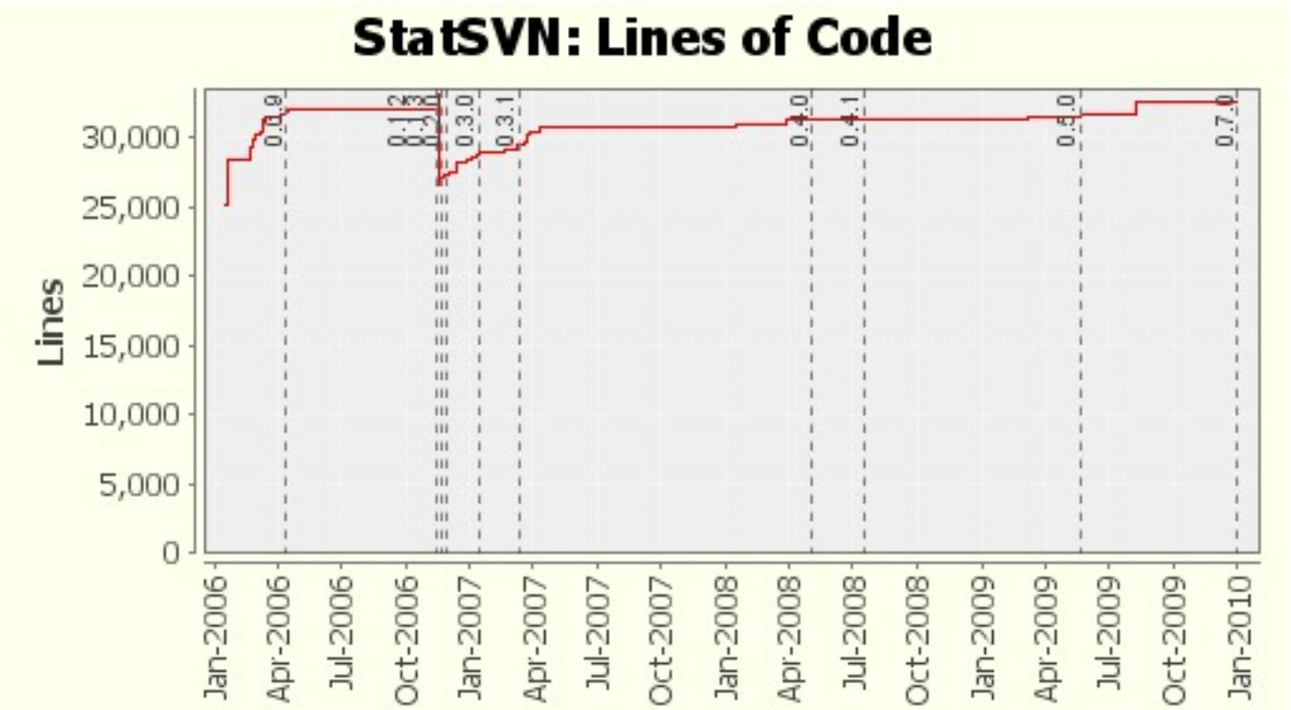
```
2011-06-10 simon.brandhof@gmail.com  211      throw new IllegalStateException("Plugin classloaders are not initialized");
                                     212    }
2010-10-15 mandrikov@gmail.com       213    try {
2010-11-24 mandrikov@gmail.com       214      return world.getRealm(key);
2010-10-15 mandrikov@gmail.com       215    } catch (NoSuchRealmException e) {
```

Either log or rethrow this exception along with some contextual information.

Open | Debt: 10 minutes | Author: mandrikov@gmail.com

```
                                     216      return null;
                                     217    }
                                     218  }
                                     219
```

Tool: StatSVN



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

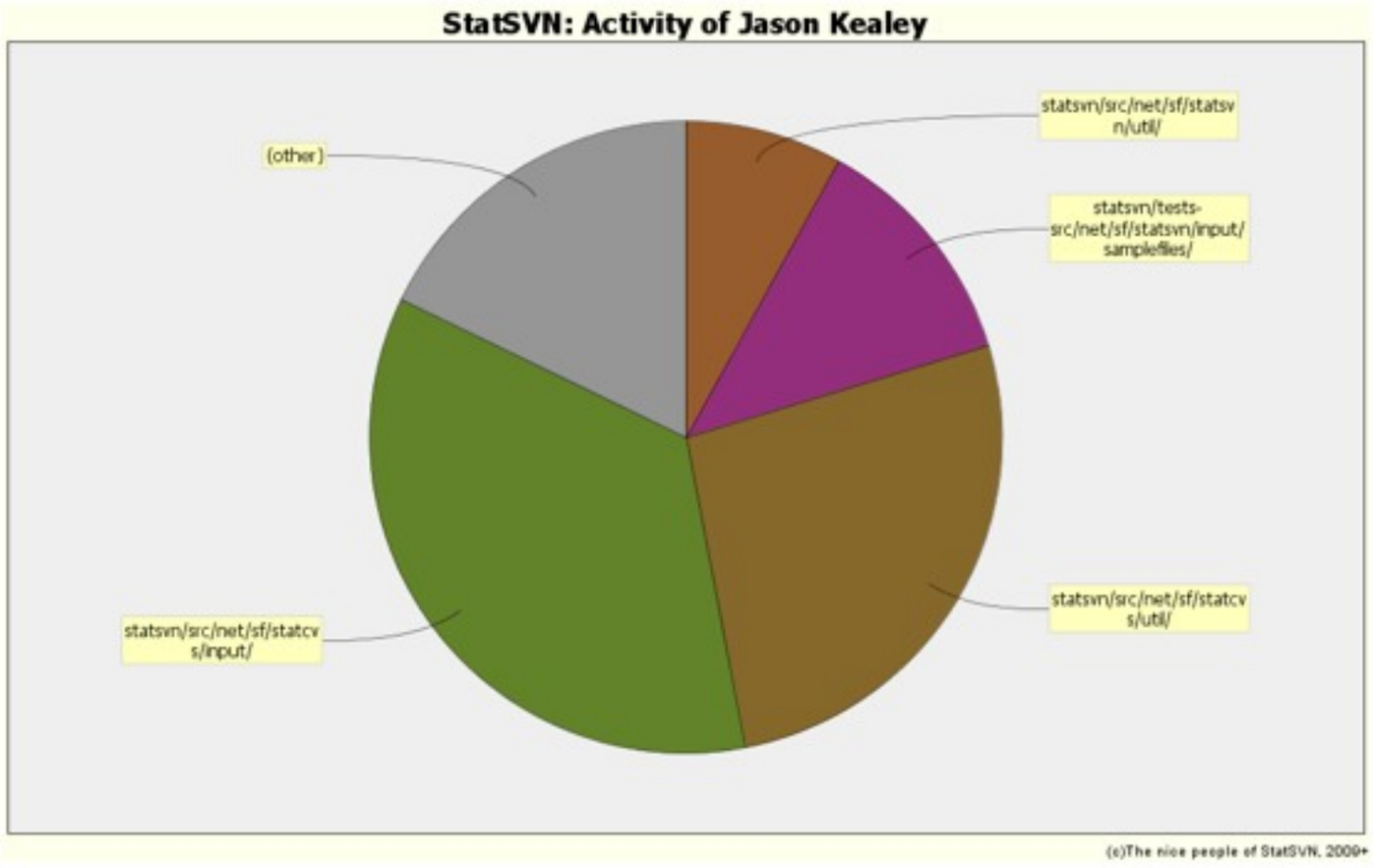
The nice people of StatSVN, 2009+

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

http://www.statsvn.org/

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	jpdaigle	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/

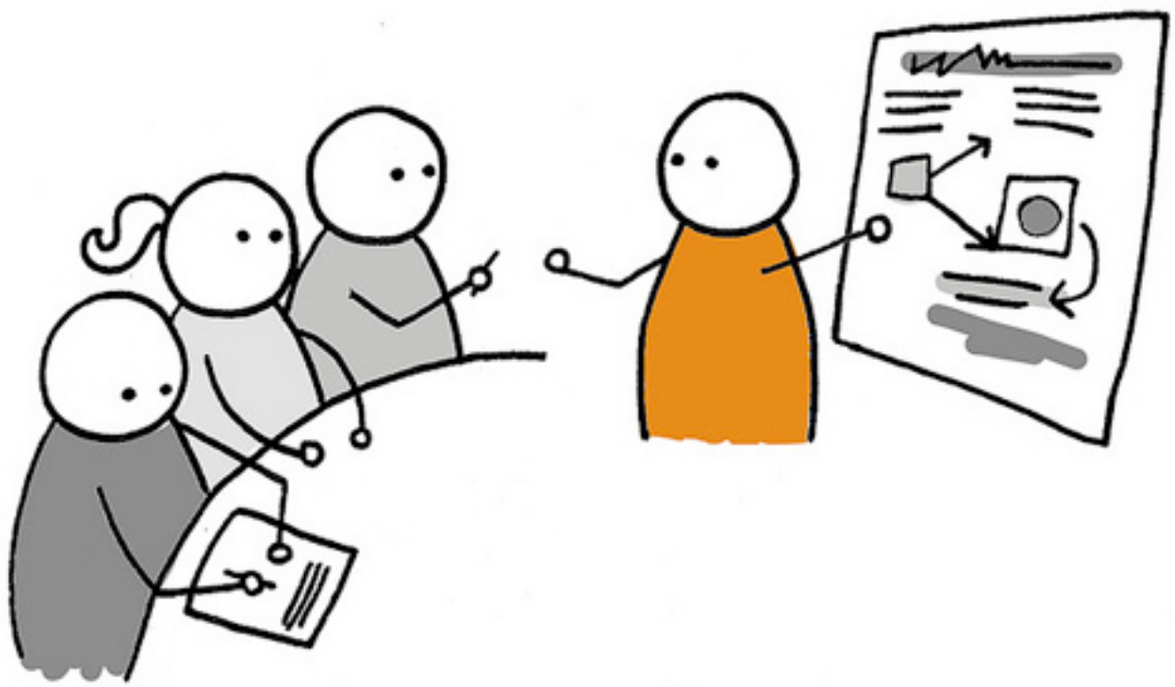
Developer of the Month

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://2014.msrconf.org/
  - 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

Group Work



Group Work

- **Split in 4 groups**
  - Ideally, each group should have at least one student who has experience working in a software development team.
- **Discussion (20’)**
  - Which agile practice do you think is **the most important one** and **why** (note: it does not matter whether you have used it or not)?
  - Describe one agile practice that you have tried to apply, but with **limited or no success**. Explain **why**, what you learned in the process and whether you will try again.
  - Describe one agile practice that **you would not recommend**. Explain why.
- **Presentation (4 x 5’)**

Some Agile (XP) Practices

- Stories
- The Planning Game
- Testing
- Refactoring
- Test-First Programming
- Pair programming
- Coding standards
- Collective ownership
- Sit Together
- On-site customer
- Metaphor
- Informative Workspace
- Small releases
- Weekly Cycle
- Quarterly Cycle
- Continuous Integration
- Simple design
- Incremental Design
- 40 hours week
- Slack

Lab

Lab

- **For people who have never used apache maven:**
  - Quick introduction to apache maven: objectives, concepts, references
  - **Activity**: install maven and be able to build, test, run a project with maven
  - **Objective** for next week: be able to demo the “AppFuse QuickStart” application (http://appfuse.org/display/APF/AppFuse+QuickStart)
- **For people who are familiar with maven:**
  - Task: installation of SonarQube
  - Next week: be able to demo SonarQube running on the “AppFuse QuickStart” application

Hints

- You can use different databases with the demo app and can use an in-memory app (H2):
  - See: http://raibledesigns.com/rd/entry/database_profiles_in_appfuse_2
  - Use the profile of your choice, such as: `mvn jetty:run -Ph2`
- You will certainly need to increase the memory:
  - `export MAVEN_OPTS="-Xmx1024M -XX:PermSize=512m"`
- Installation guide for SonarQube:
  - http://docs.codehaus.org/display/SONAR/Installing
  - http://docs.codehaus.org/display/SONAR/Installing+and+Configuring+Maven

MSE | MASTER OF SCIENCE IN ENGINEERING

Introduction to Maven

Agenda

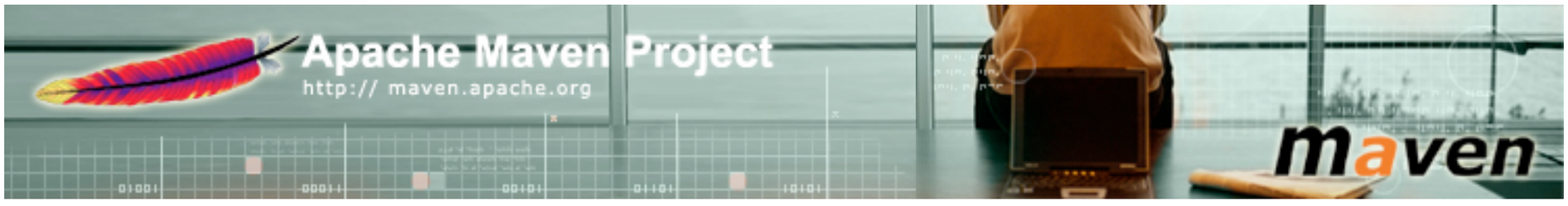
- Maven
- Core concepts
- Lifecycles, phases, plugins and goals (mojos)
- Project relationships
- Maven and Java EE

What is the difference between a **library** and a **framework**?


HOLLYWOOD

What is the difference between a **ant** and a **maven**?

Maven, a Yiddish word meaning **accumulator of knowledge**, was originally started as an attempt to **simplify the build processes** in the Jakarta Turbine project. There were several projects each with their own Ant build files that were all slightly different and JARs were checked into CVS. We wanted **a standard way to build the projects**, a clear **definition of what the project consisted of**, an easy way to **publish project information** and a way to **share JARs** across several projects.



References

http://www.sonatype.com/products/maven/documentation/book-defguide

Getting started...

- Install maven 2
  - http://maven.apache.org/download.html
- Create and build a project
- Look at these directories and files
  - ~/.m2/repository
  - ~/.m2/settings.xml
  - $INSTALLATION/conf/settings.xml

```
mvn archetype:generate \
   -DarchetypeGroupId=org.apache.maven.archetypes \
   -DarchetypeArtifactId=maven-archetype-quickstart \
   -DgroupId=ch.heigvd.osf.cool \
   -DartifactId=coolProject
cd coolProject
mvn install
```

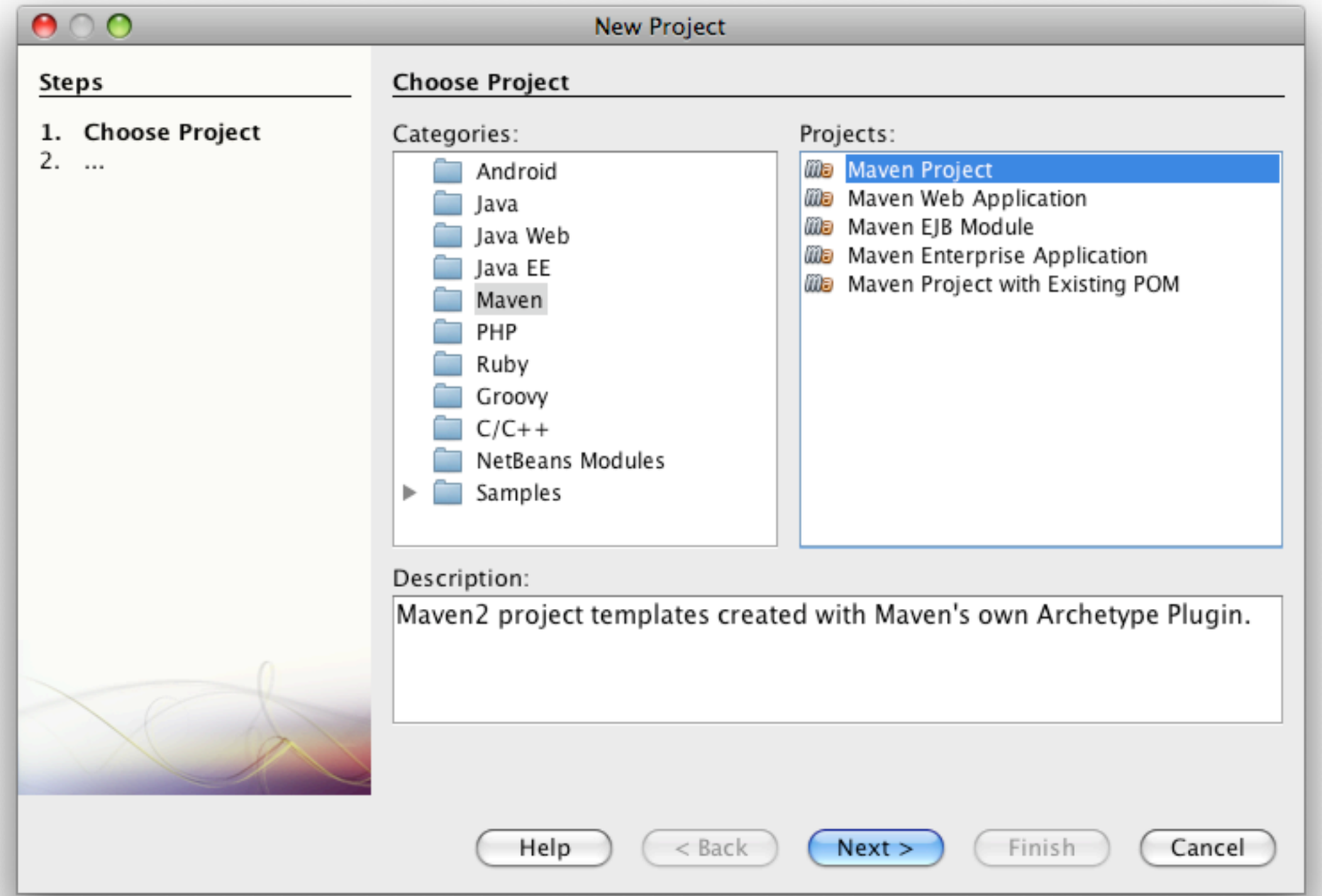
Project Object Model (POM)

- The POM is an XML document that describes the project:
  - What kind of artifacts are we building (jar, war, ear, etc.)?
  - What libraries do we depend on to build the artifact?
  - What are the special actions that need to be done during the build?
  - Where can plug-ins and dependencies be found?
  - Are there relationships with other maven-driven projects?
- In other words, with maven:
  - You declare “properties” about your project.
  - You let maven build the artifact, based on conventions and best practices.

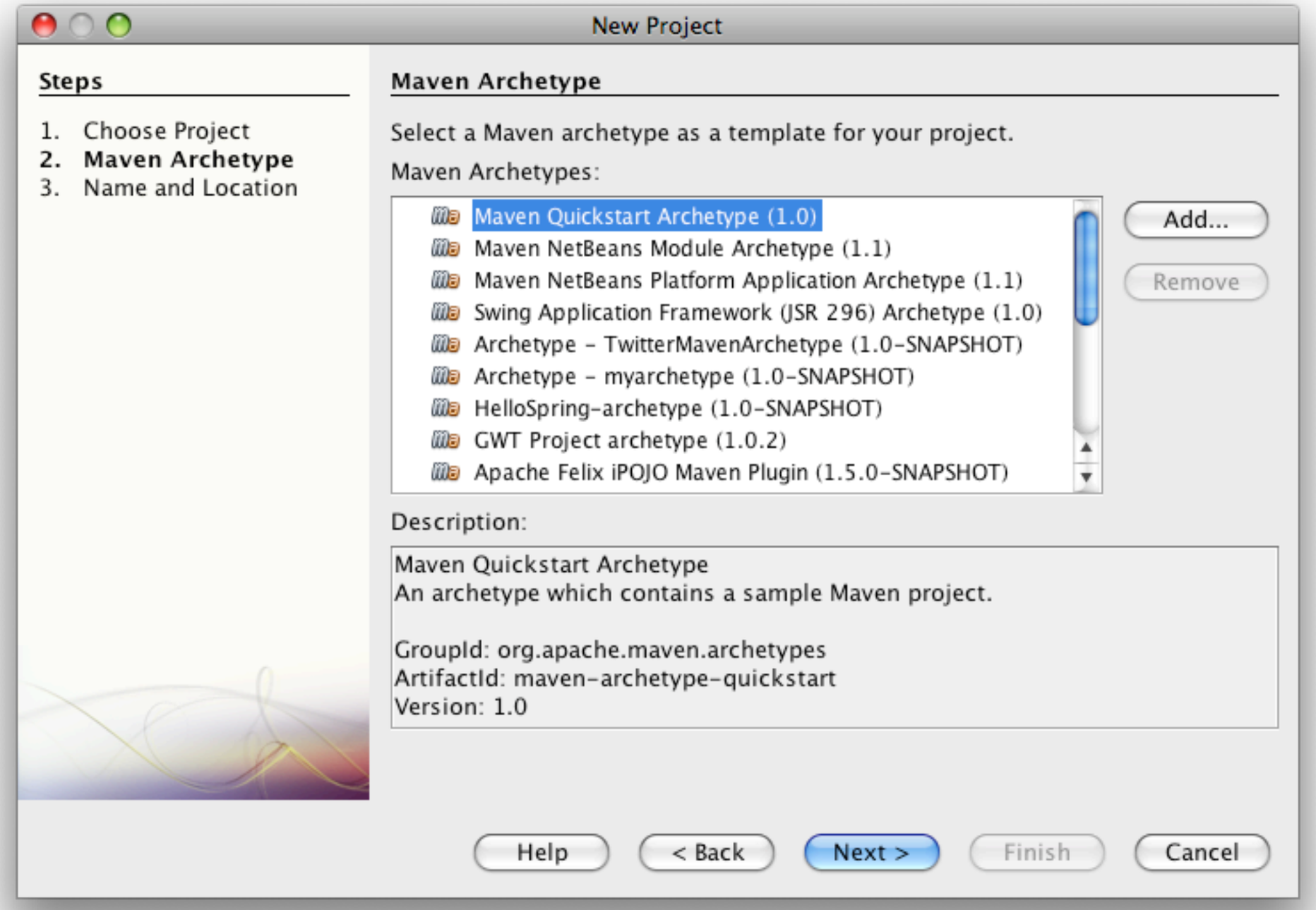
Maven Coordinates

- Every maven project produces one **artifact**, whether it is a jar file, a war file or anything else.
- When a project has a **dependency** on an artifact produced by another project, it needs a way to **reference this artifact**.
- Maven **coordinates** address this need: every project is identified by three values:
  - A group id (used to group related artifacts)
  - An artifact id
  - A version number
- Maven coordinates are the things you will find in every **POM**.
- Maven coordinates are also used to capture **inheritance relationships** between projects (more on this later).

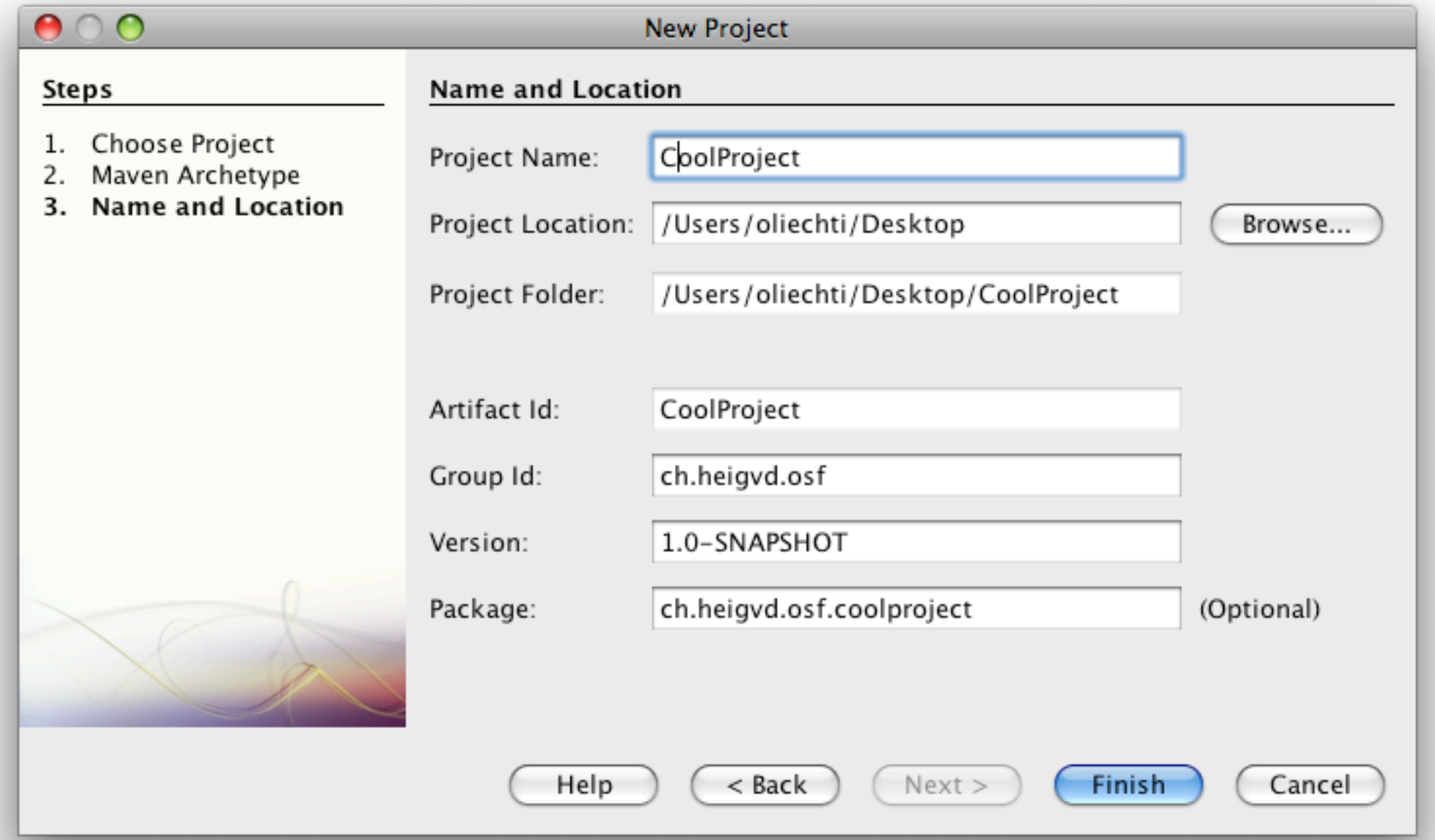
Creating a maven project with NetBeans



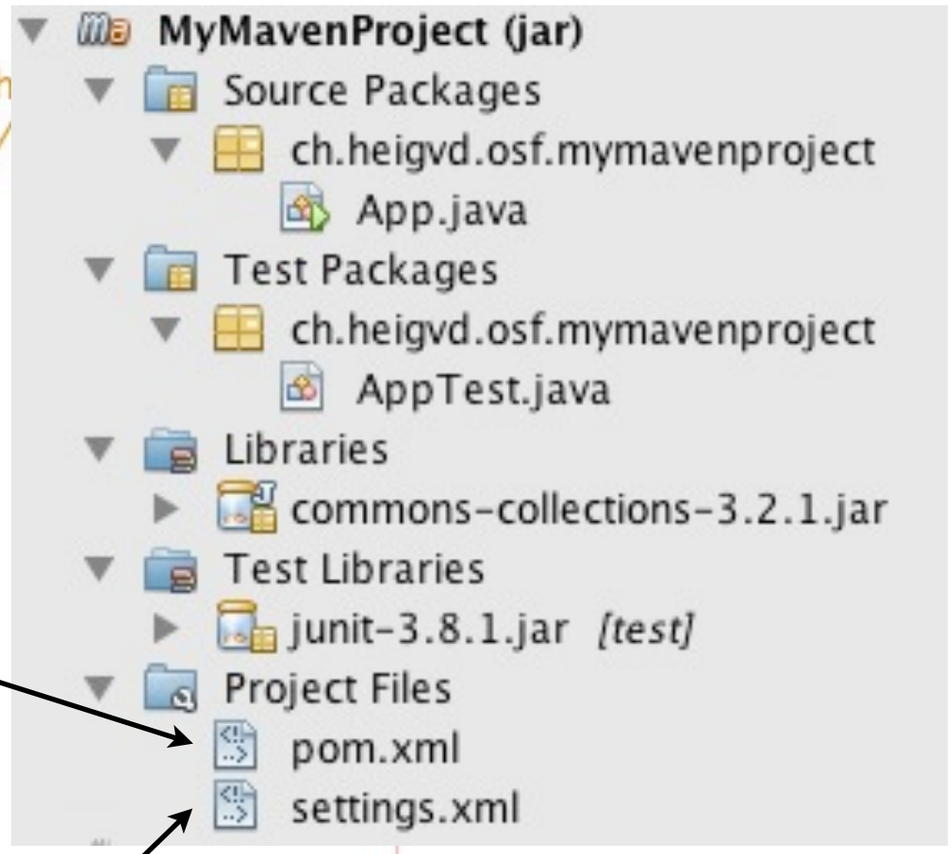
Creating a maven project with NetBeans



Creating a maven project with NetBeans



```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="h
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
  <modelVersion>4.0.0</modelVersion>
  <groupId>ch.heigvd.osf</groupId>
  <artifactId>MyMavenProject</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>MyMavenProject</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>commons-collections</groupId>
      <artifactId>commons-collections</artifactId>
      <version>3.2.1</version>
    </dependency>
  </dependencies>
</project>
```



Shortcut on your maven preferences

Dependencies

- Scopes
  - **compile**: you need to compile and it will be packaged with the artifact.
  - **provided**: you need it to compile, but it will not be packaged - it will be provided by the runtime environment.
  - **runtime**: you need it to execute the system, but not to compile it.
  - **test**: you only need it during for running tests.
  - **system**: similar to provided, but path has to be provided.

```
<project>
  ...
  <dependencies>
    <dependency>
      <groupId>org.codehaus.xfire</groupId>
      <artifactId>xfire-java5</artifactId>
      <version>1.2.5</version>
    </dependency>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <version>2.4</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>
  ...
</project>
```

Local and Remote Repositories

- **Local repository**
  - On **every machine**, there is a local repository. It is located in ~/.m2/ repository.
  - This repository acts as a **cache** (when you fetch libraries during a build, they are kept there for later usage).
  - When you build your artifacts, they are "**installed**" here.
- **Remote repositories**
  - Remote repositories are used to **share artifacts** between several developers & organizations.
  - A remote repository can be **public** (e.g. http://repo1.maven.org/maven2/, maven repositories of open source projects, etc.)
  - A remote repository can be **private** (i.e. a company manages a repository so that internal developers can share artifacts).

Want to manage your repository?

- http://archiva.apache.org/
- http://nexus.sonatype.org/

http://www.flickr.com/photos/garryknight/2818736770/sizes/m/#cc_license

Plugins

“The core of Maven is pretty dumb, it doesn't know how to do much beyond parsing a few XML documents and keeping track of a lifecycle and a few plugins.

Maven has been designed to delegate most responsibility to a set of Maven Plugins which can affect the Maven Lifecycle and offer access to goals.”

http://www.sonatype.com/books/mvnref-book/reference/installation-sect-universal-reuse.html

Plugins

- A **plugin** implements a number of actions, called "**goals**" (aka "**mojos**")
  - The `clean` plugin has one goal: `clean:clean`
  - The `jar` plugin has too goals: `jar:jar` and `jar:test-jar`
- Plugin goals often accept **parameters** (some optional, some required)
- Plugins are contributed by the **community** - you can write your own plugins.
- You can **invoke** a goal directly:

```
mvn clean:clean
```

- You can let maven invoke the right goals at the right time (**hollywood** principle)

```
mvn install
```

Maven Plugins

Plugin
Core plugins
clean
compiler
deploy
failsafe
install
resources
site
surefire
verifier

Packaging types / tools
ear
ejb
jar
rar
war
shade

Tools
ant
antrun
archetype
assembly
dependency
enforcer
gpg
help
invoker
jarsigner
one
patch
pdf
plugin
release
reactor
remote-resources
repository
scm
source
stage
toolchains

Reporting plugins
changelog
changes
checkstyle
clover
doap
docck
javadoc
jxr
pmd
project-info-reports
surefire-report

Lifecycle

- A **lifecycle** is a sequence of phases (e.g. compile, test, package, etc.)
- A **phase** is where some actions can be attached (e.g. invoke a compiler during the compile phase)
- Maven provides **standard lifecycles**:
  - clean lifecycle
  - default lifecycle
  - site lifecycle
- Lifecycles can be customized depending on the **type of artifact** being built (building .jar does not involve the same steps as building a .war).
- **Convention over configuration**: if you don't specify otherwise, you let maven proceed "as usual" and do not care about the setup of lifecycles and plugins.

Table 4.3. Default Goals for POM Packaging

Lifecycle Phase	Goal
package	site:attach-descriptor
install	install:install
deploy	deploy:deploy

Table 4.2. Default Goals for JAR Packaging

Lifecycle Phase	Goal
process-resources	resources:resources
compile	compiler:compile
process-test-resources	resources:testResources
test-compile	compiler:testCompile
test	surefire:test
package	jar:jar
install	install:install
deploy	deploy:deploy

Table 4.6. Default Goals for WAR Packaging

Lifecycle Phase	Goal
process-resources	resources:resources
compile	compiler:compile
process-test-resources	resources:testResources
test-compile	compiler:testCompile
test	surefire:test
package	war:war
install	install:install
deploy	deploy:deploy

http://www.sonatype.com/books/mvnref-book/reference/lifecycle-sect-war.html#

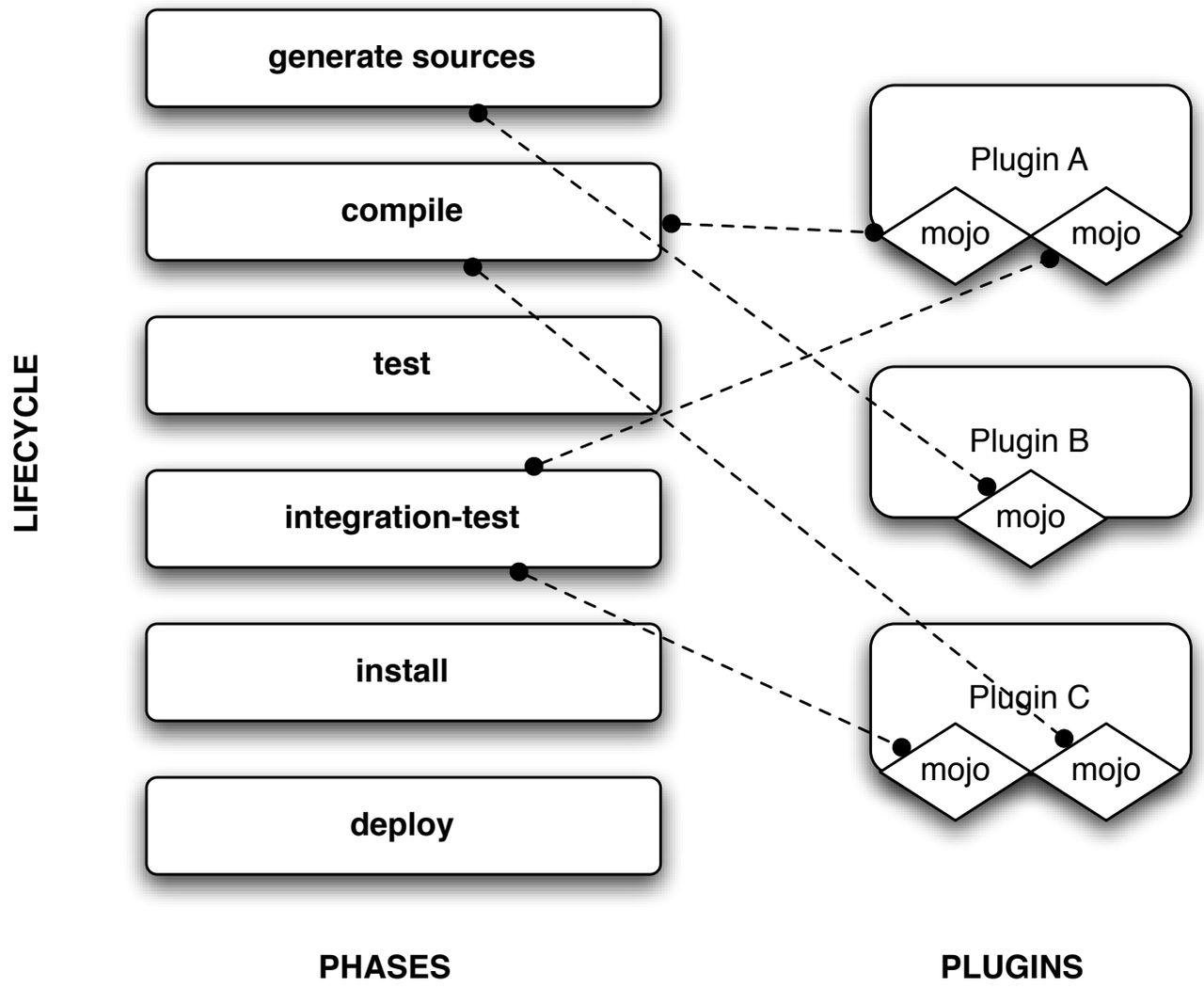
```
mvn install
```

This runs the lifecycle until the install phase; this results in the installation of the artifact in the local maven repository.

All mojos attached to the phases are executed.

```
mvn deploy
```

This runs the lifecycle until the deploy phase; this results in the deployment of the artifact in a remote maven repository.



Relationships between Projects (1)

- Multi-modules projects
  - A project can be organized in sub-projects.
  - When you built the project artifact, you also want to build the artifacts of the sub-projects.
  - If there are dependencies between some of the sub-projects, you need to build the artifacts in the right order.
- Maven takes care of that, through the “Reactor”.

```
<modules>
  <module>moduleA</module>
  <module>moduleC</module>
  <module>moduleB</module>
</modules>
```

Relationships between Projects (2)

- Inheritance relationships between project
  - A maven project can extend another maven project.
  - This is a way to inherit various properties, such as the versions of the libraries used all sub-projects (“everybody should use log4j version 1.3.2”).
  - The relationship is captured at the beginning of the POM, using maven coordinates.
- Lost in your inheritance tree?
  - Run this goal: mvn help:effective-pom

```
<parent>
  <groupId>your.group.id</groupId>
  <artifactId>toto</artifactId>
  <version>1.0-SNAPSHOT</version>
</parent>
```

Inheritance & dependency management

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sonatype.mavenbook</groupId>
  <artifactId>a-parent</artifactId>
  <version>1.0.0</version>
  ...
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.2</version>
      </dependency>
      ...
    <dependencies>
</dependencyManagement>
```

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.sonatype.mavenbook</groupId>
    <artifactId>a-parent</artifactId>
    <version>1.0.0</version>
  </parent>
  <artifactId>project-a</artifactId>
  ...
  <dependencies>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
    </dependency>
  </dependencies>
</project>
```

Profiles

- For the same project, you often want to be able to do **different types of builds**:
  - A developer builds the artifact on his local machine during development.
  - Continuous integration triggers daily builds on a QA server.
  - You want to build the artifact before putting into production.
- For every **different type of build**, you may want to:
  - Use different “components” in the environment (different glassfish domains, different DBs, etc.)
  - Skip some of the phases in the build process.
  - Attach some special plugins to some of the phases
- Profiles allow you to do just that. Within a **profile definition**, you declare the expected behavior. You then **activate** one or more profiles (either via a command line flag, or indirectly through maven properties).

Profiles

```
<project>
  <profiles>
    <profile>
      <build>
        <defaultGoal>...</defaultGoal>
        <finalName>...</finalName>
        <resources>...</resources>
        <testResources>...</testResources>
        <plugins>...</plugins>
      </build>
      <reporting>...</reporting>
      <modules>...</modules>
      <dependencies>...</dependencies>
      <dependencyManagement>...</dependencyManagement>
      <distributionManagement>...</distributionManagement>
      <repositories>...</repositories>
      <pluginRepositories>...</pluginRepositories>
      <properties>...</properties>
    </profile>
  </profiles>
</project>
```

Resource Filtering

- Very often, you want some parts of your resource files, possibly of your source code, to depend on the execution environment:
  - If I deploy on the QA server, I want to talk to this DB server; if I deploy on the production server, I want to talk to this other DB server.
  - If I deploy on the test server, I want to invoke this web service endpoint; if I deploy on the production server, I want to use this other endpoint.
- Resource filtering is a mechanism, where you can ask maven to **expand properties** in your code base during the build process.

Resource Filtering

```
<profiles>
  <profile>
    <id>production</id>
    <properties>
      <jdbc.driverClassName>oracle.jdbc.driver.OracleDriver</jdbc.driverClassName>
      <jdbc.url>jdbc:oracle:thin:@proddb01:1521:PROD</jdbc.url>
      <jdbc.username>prod_user</jdbc.username>
      <jdbc.password>s00p3rs3cr3t</jdbc.password>
    </properties>
  </profile>
</profiles>
```

POM.xml - you give values to properties

```
    <bean id="dataSource" destroy-method="close"
            class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName" value="${jdbc.driverClassName}"/>
        <property name="url" value="${jdbc.url}"/>
        <property name="username" value="${jdbc.username}"/>
        <property name="password" value="${jdbc.password}"/>
    </bean>
</beans>
```

Source - you reference maven properties with ${xxx}

Archetypes

- A maven archetype is a **skeleton** for a certain type of project.
- You have already used archetypes when creating your first maven project.
- Archetype definitions can be **shared** via repositories.
- **Creating you own archetype is a very efficient way to create a SDK.**
- **AppFuse** is an open source project that uses archetypes as a way to pre-integrate several open source frameworks (with a layer on top of them).
- The archetype facility is available through the **archetype plugin**:

  http://maven.apache.org/archetype/maven-archetype-plugin/

http://appfuse.org