Software Engineering and Architecture
Lecture 2: Software evolution

Olivier Liechti
olivier.liechti@heig-vd.ch

MSE | MASTER OF SCIENCE IN ENGINEERING

Today.

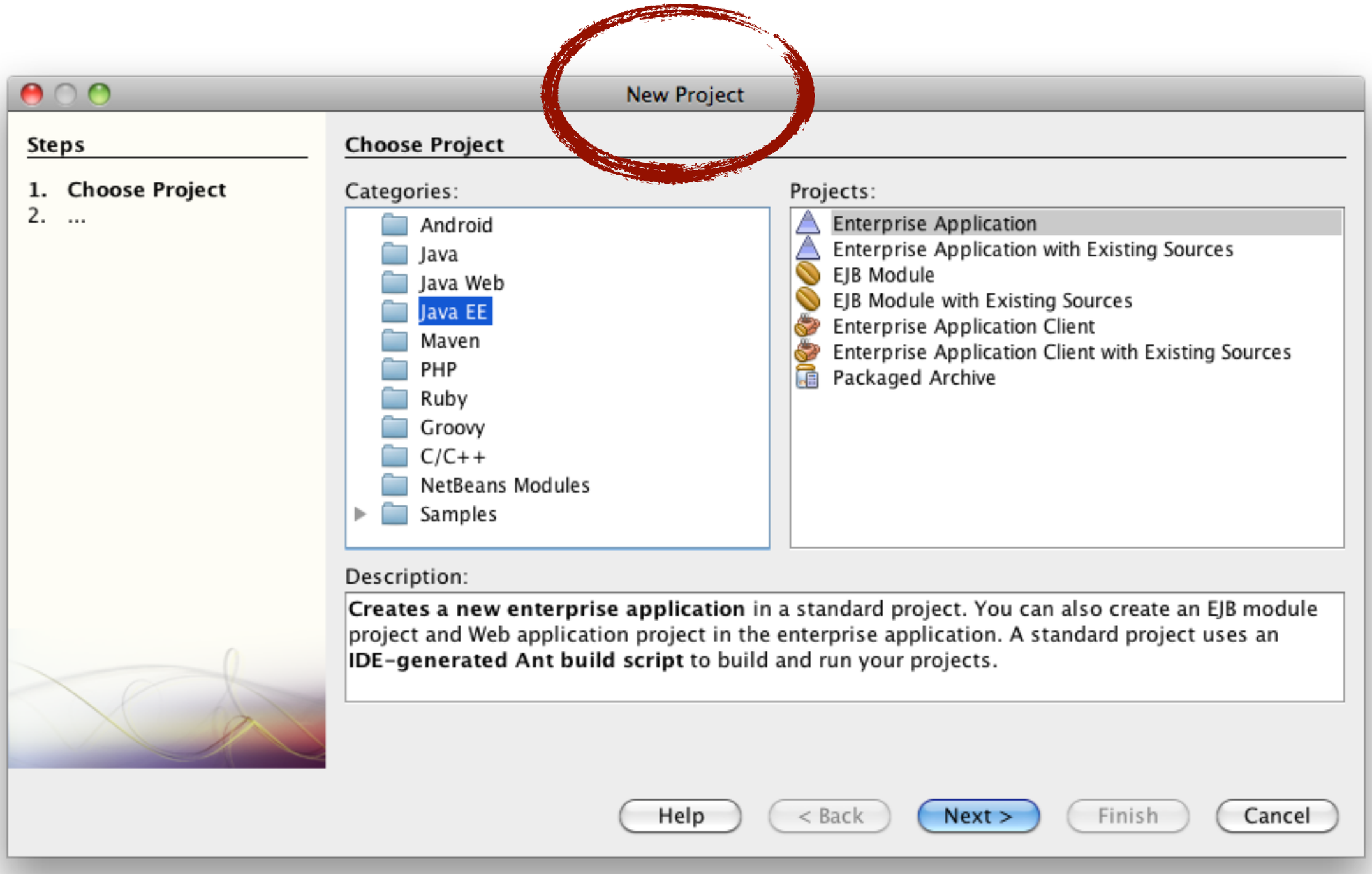
Wrap-up scrum & agile estimations
15h00 - 15h15

Software evolution + guidelines presentations
15h15 - 16h15

Develop a microservice with Spring Boot
16h15 - 17h20

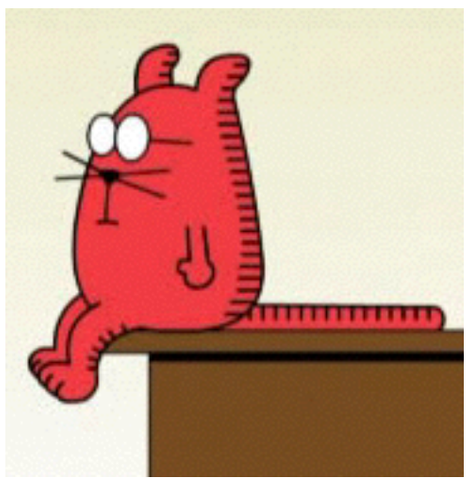
Week	Theory	OO Reengineering	Practice
#1	Agile, Scrum		Intro to Docker
#2	Software evolution	Introduction	Specify and implement a micro-service
#3	Continuous delivery (1)	*Setting Directions*	Intro to Jenkins & Travis
#4	Continuous delivery (2)	*First Contact*	Our first build pipeline
#5	Agile testing (1)	*Initial Understanding*	Intro to Cucumber
#6	Agile testing (2)	*Detailed Model Capture*	Add tests to pipeline
#7	Agile metrics	*Tests: Your Life Insurance!*	Add Sonar to pipeline
#8	Continuous improvement	*Migration Strategies I*	Add non-functional tests
#9	Wrap-up	*Migration Strategies II*	

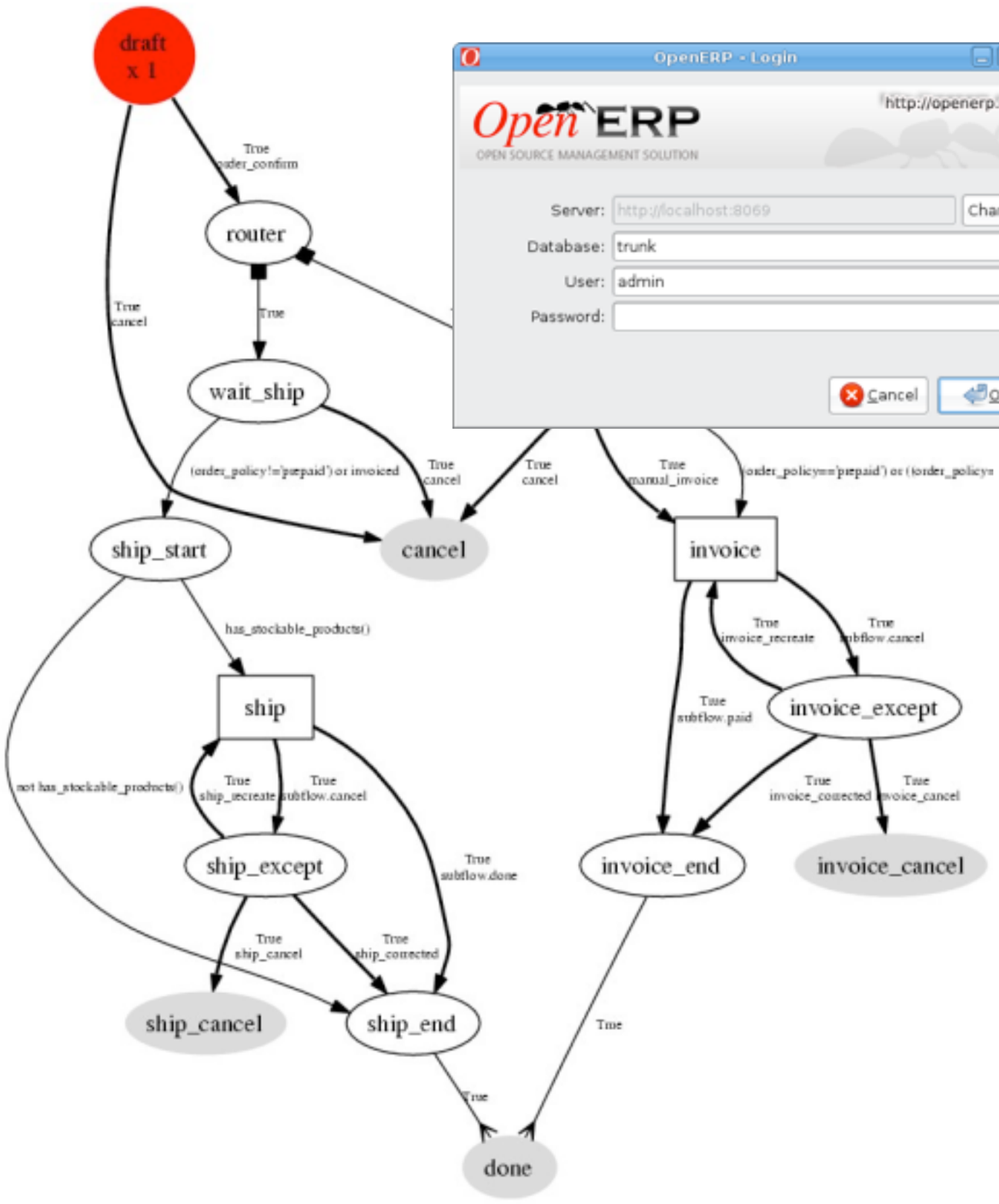
Software evolution


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!



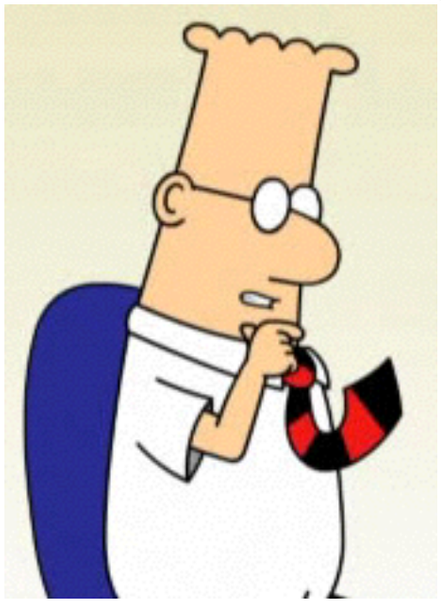


http://doc.openerp.com/developer/2_4_module_development/4_1_module_structure.html

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

Research

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 [illegible] - 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 [illegible] **software** [illegible] — 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,pdwl)@doc.ic.ac.uk URL:http:Nwww-dse.doc.ic.ac.uk/-mml/feast1/ ... M Turski Institute of Informatics Warsaw University Warsaw ...
Cited by [illegible] - 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** [illegible] — 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?

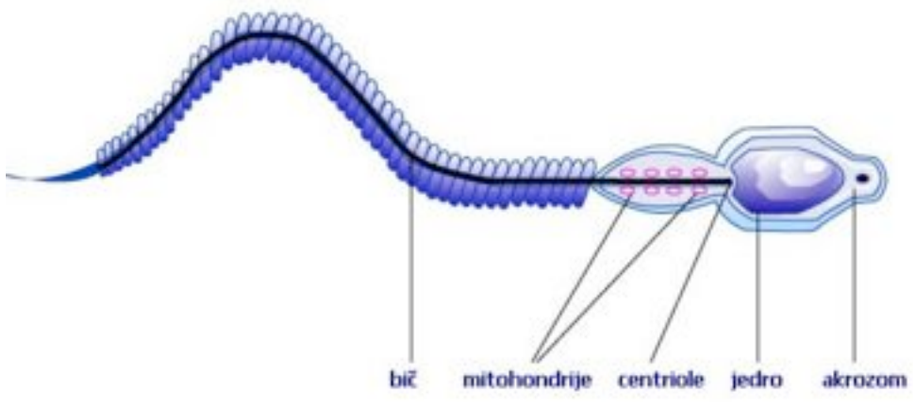
Research on Software Evolution

- **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

- **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

- **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

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.

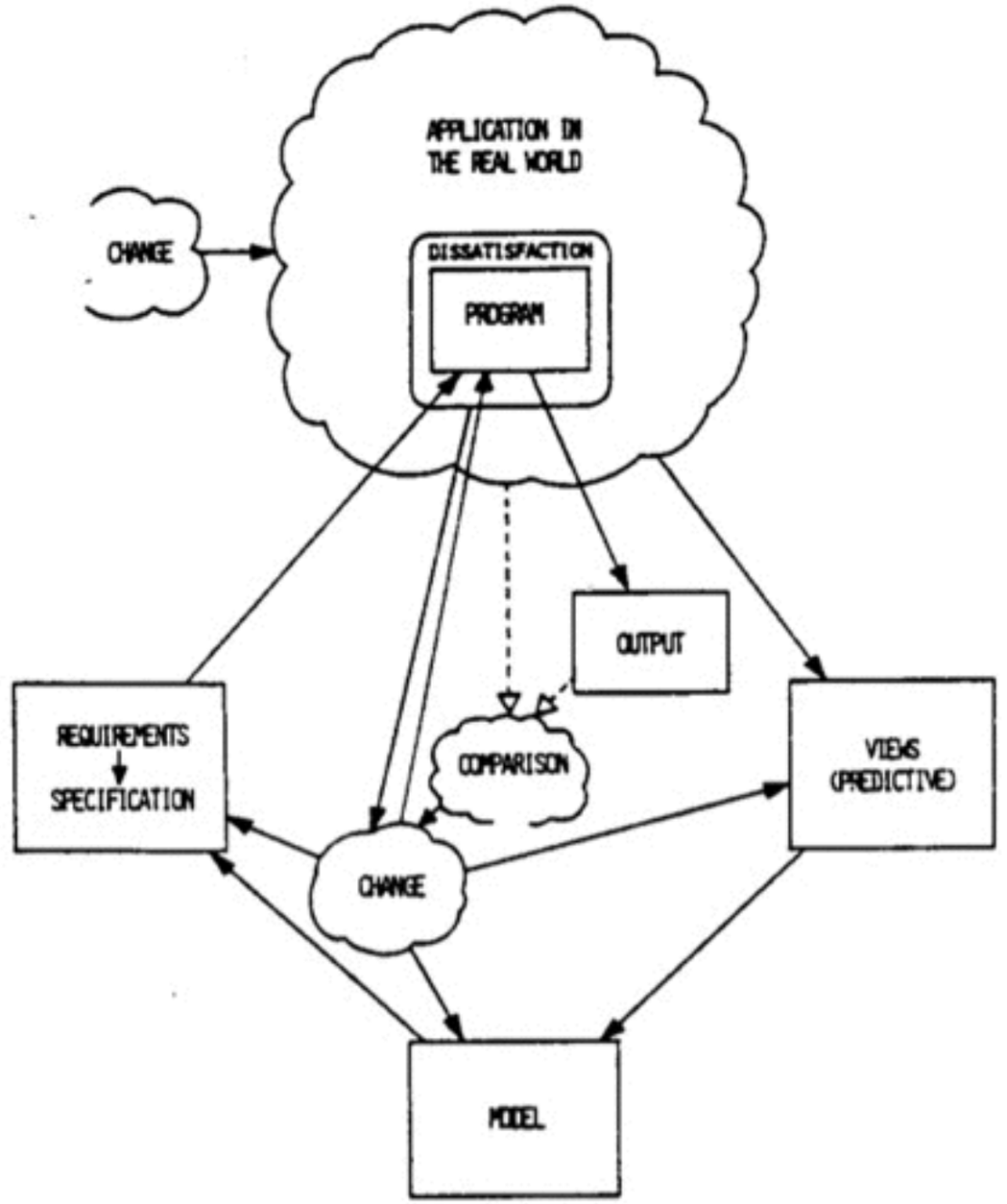
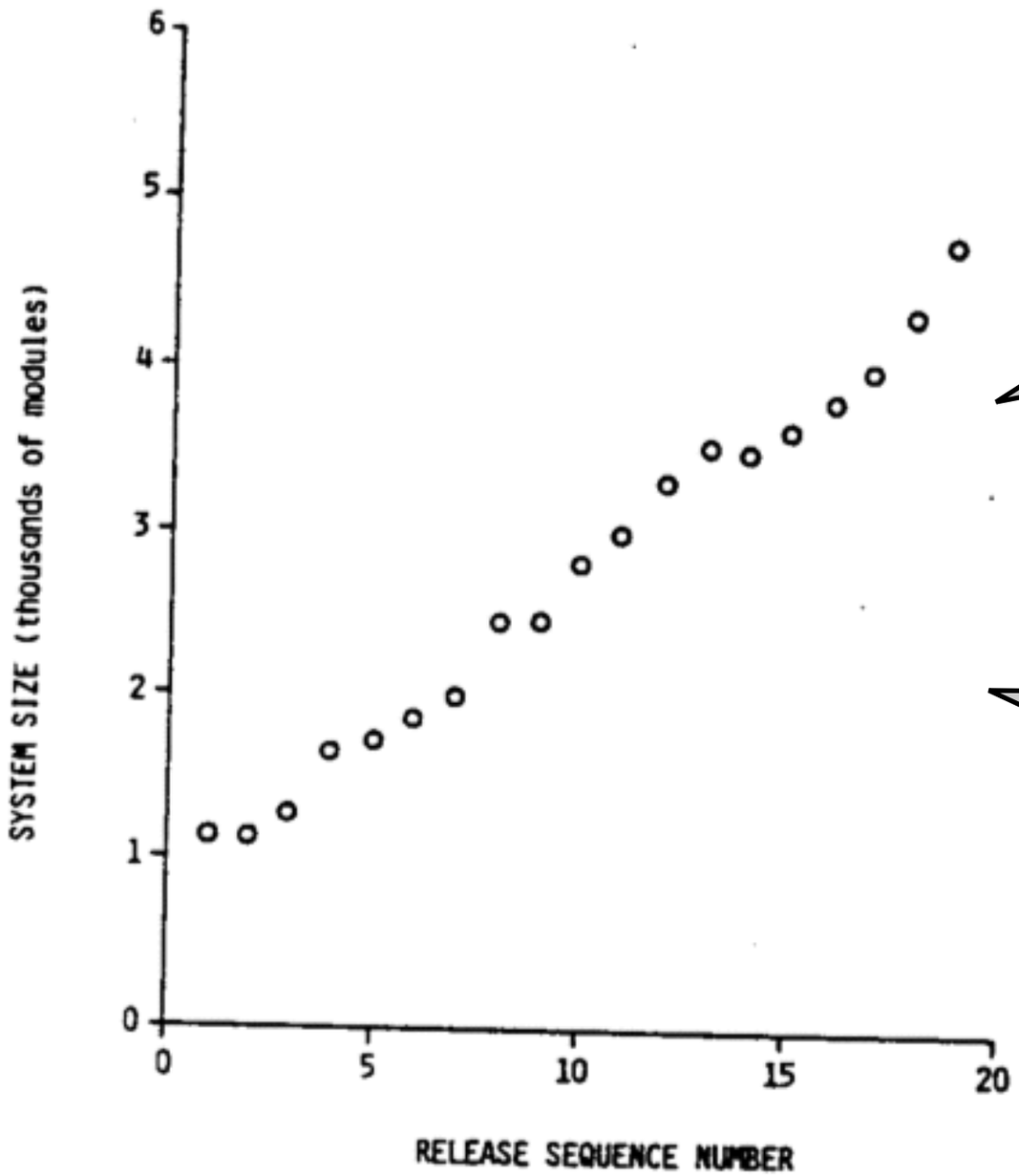


Fig. 4. *E*-programs.

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.**

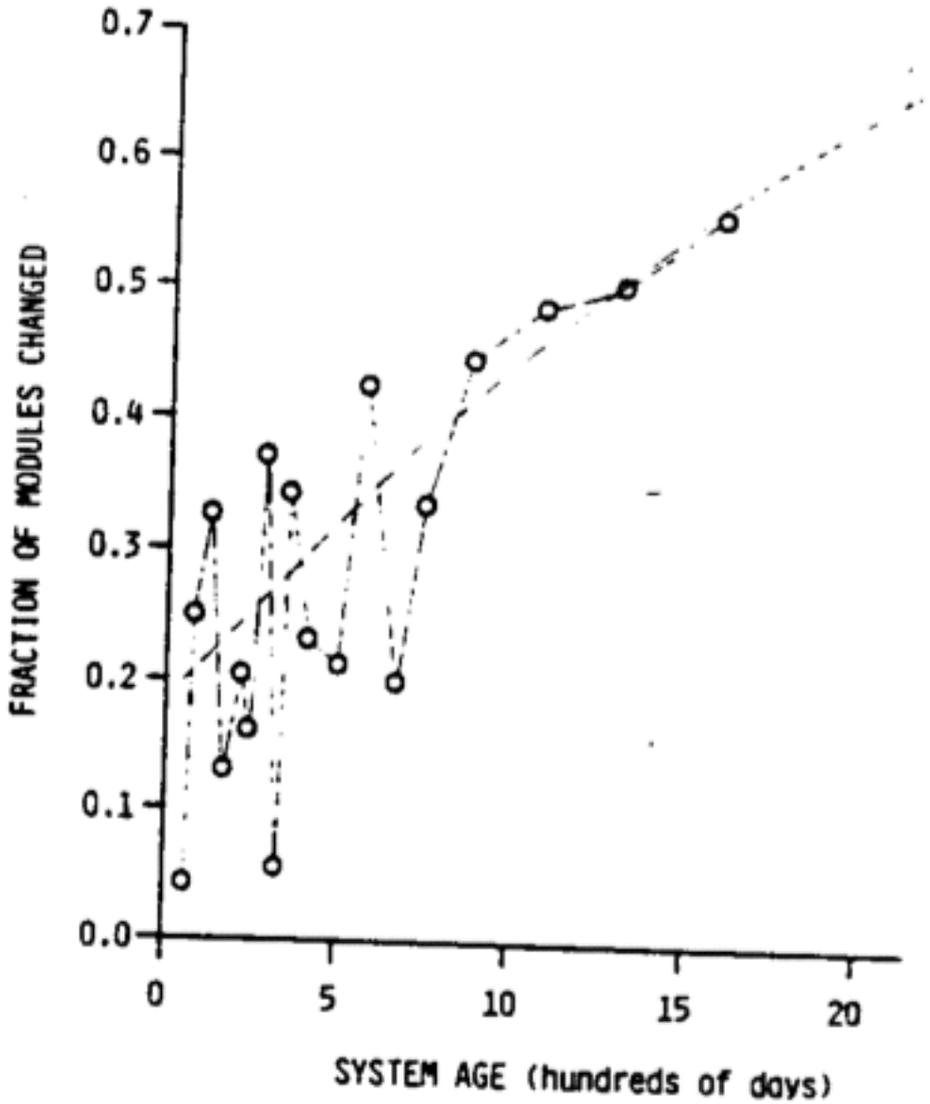
Quantitative Studies



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

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

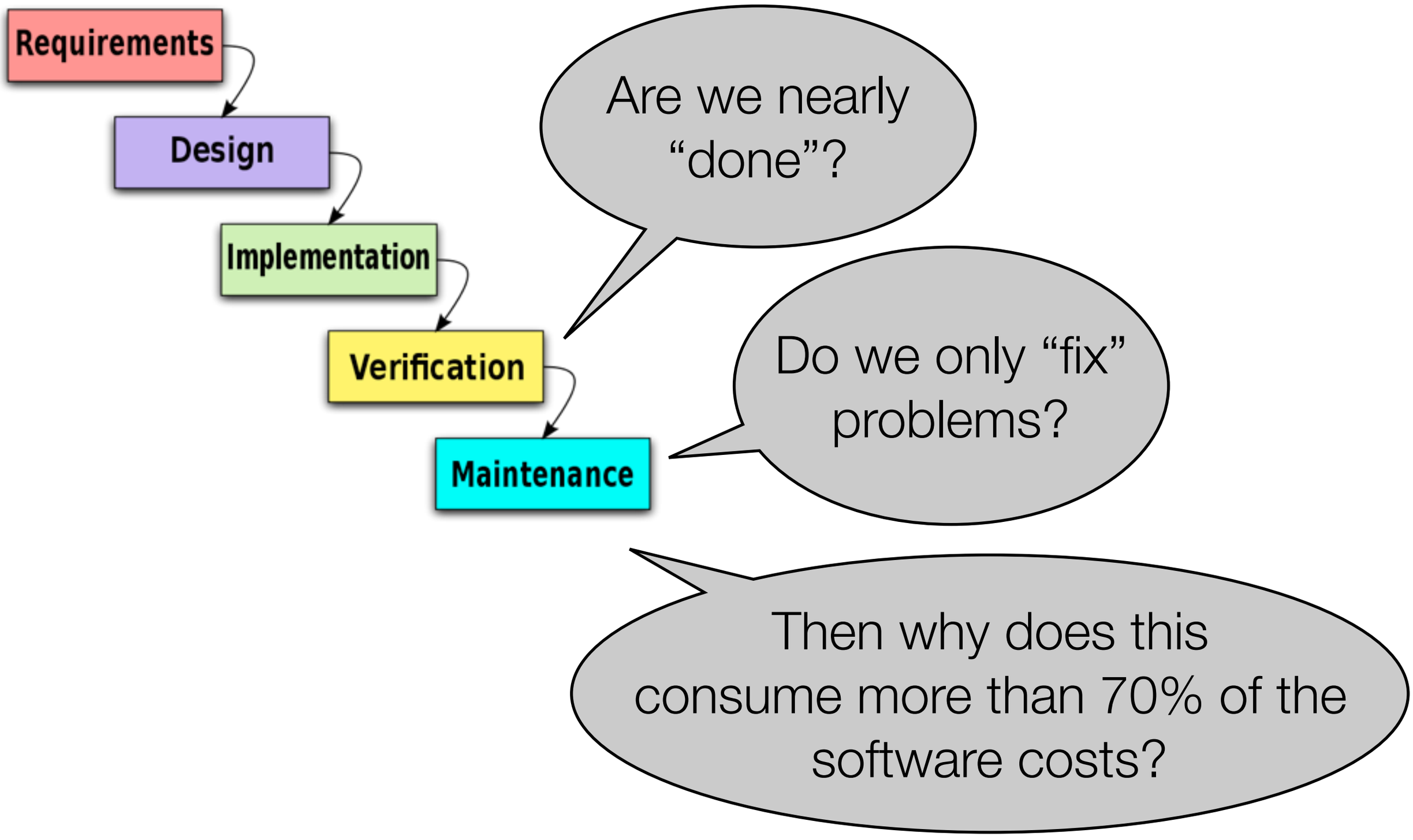
Software Maintenance

- 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**.

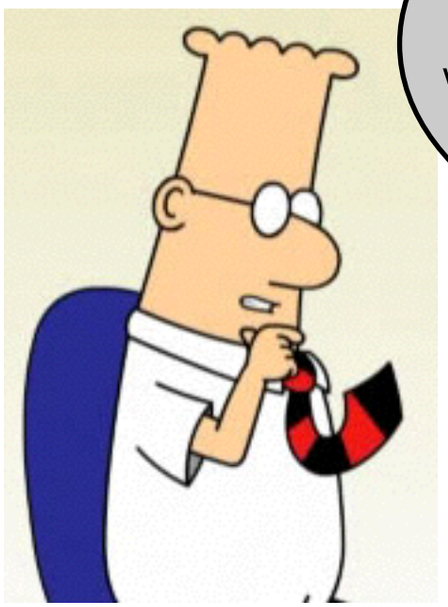
http://www.flickr.com/photos/laenulfean/474217855/sizes/m/#cc_license

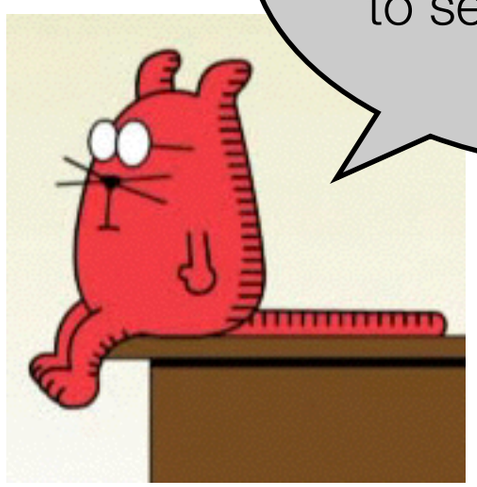
Is software maintenance only about fixing defects?

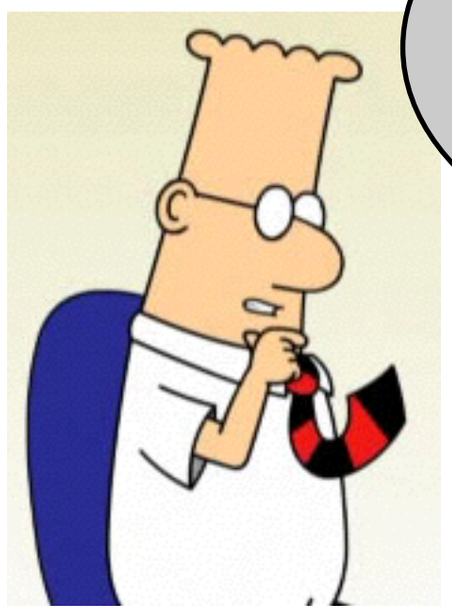
What does it mean to “maintain” software?



Software Maintenance







Software Maintenance

Corrective

Preventive

Adaptive

Perfective

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.

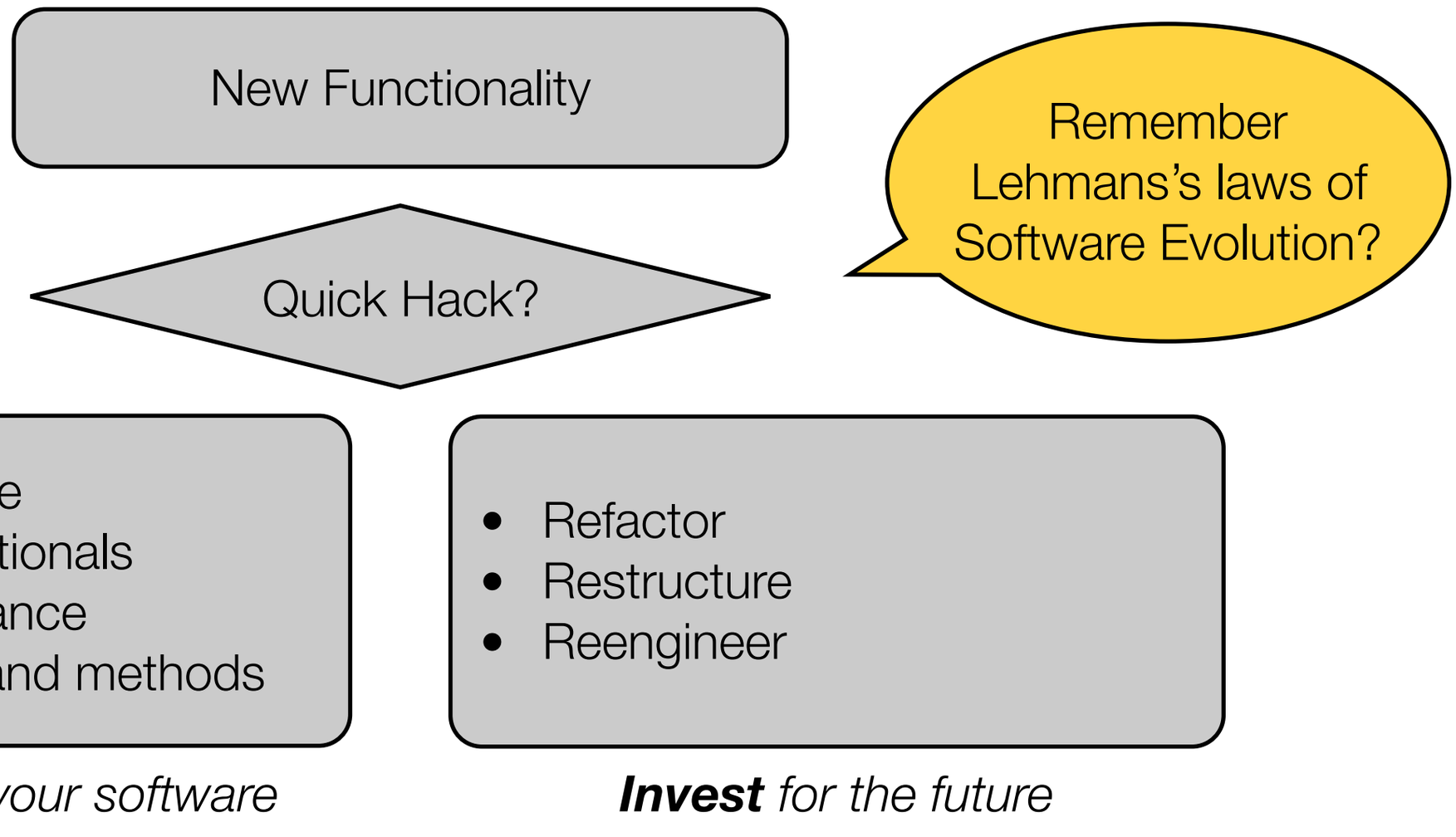
Communications of the ACM
June 1978 Volume 21 Number 6

Legacy Systems Technical Debt & Re-engineering

MSE | MASTER OF SCIENCE IN ENGINEERING

Dealing with Legacy Systems

New or changing requirements will gradually degrade original design…
…unless extra development effort is spent to adapt the structure.



Technical Debt

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.*

- http://www.youtube.com/watch?v=pqeJFYwnkjE
- http://martinfowler.com/bliki/TechnicalDebt.html

Technical Debt

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.*

- http://www.youtube.com/watch?v=pqeJFYwnkjE
- http://martinfowler.com/bliki/TechnicalDebt.html

Technical 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.

- http://www.youtube.com/watch?v=pqeJFYwnkjE
- http://martinfowler.com/bliki/TechnicalDebt.html

Technical Debt

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.*

- http://www.youtube.com/watch?v=pqeJFYwnkjE
- http://martinfowler.com/bliki/TechnicalDebt.html

Object-Oriented Reengineering

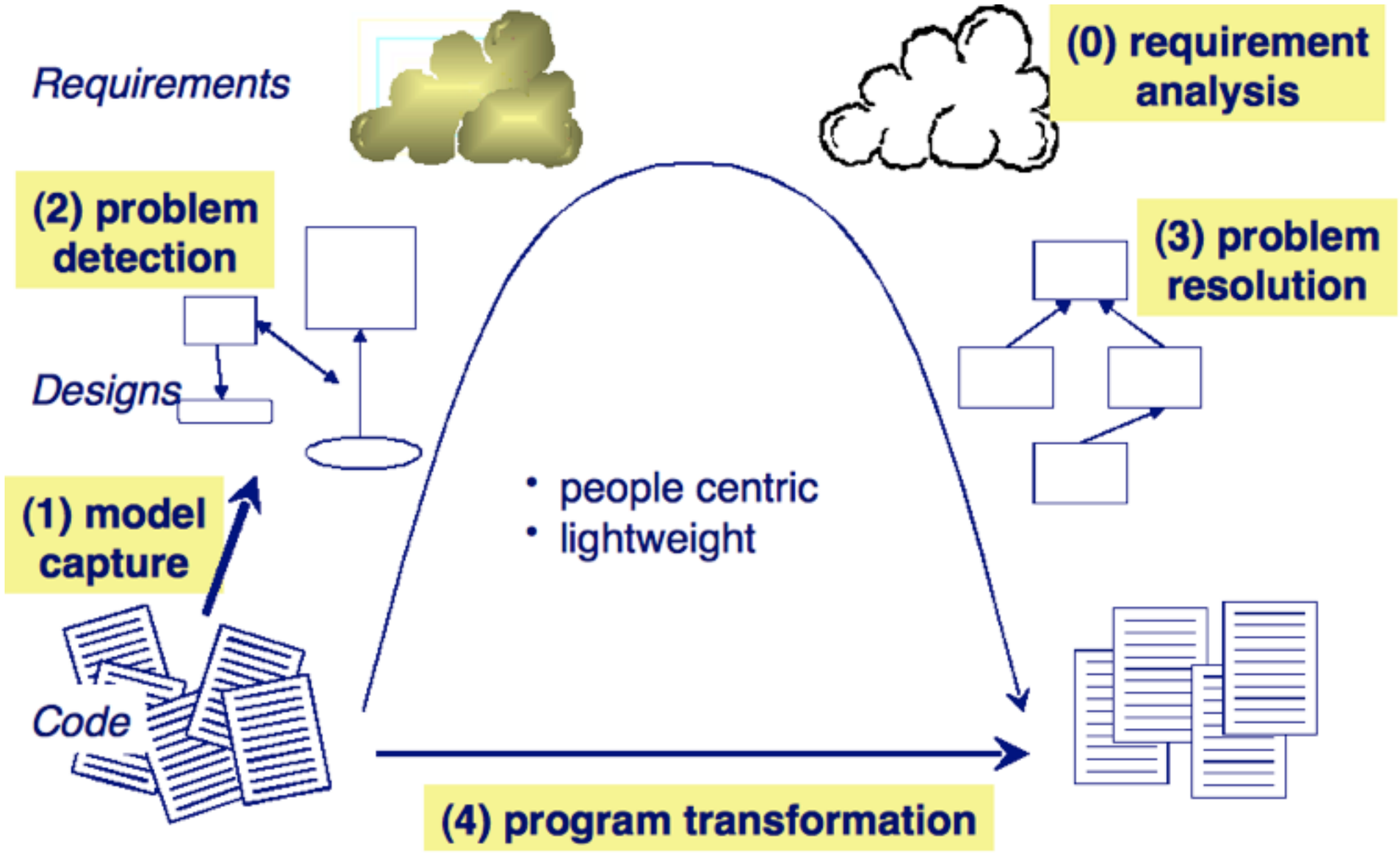
- Open source version of this book:
- http://scg.unibe.ch/download/oorp/

Terminology

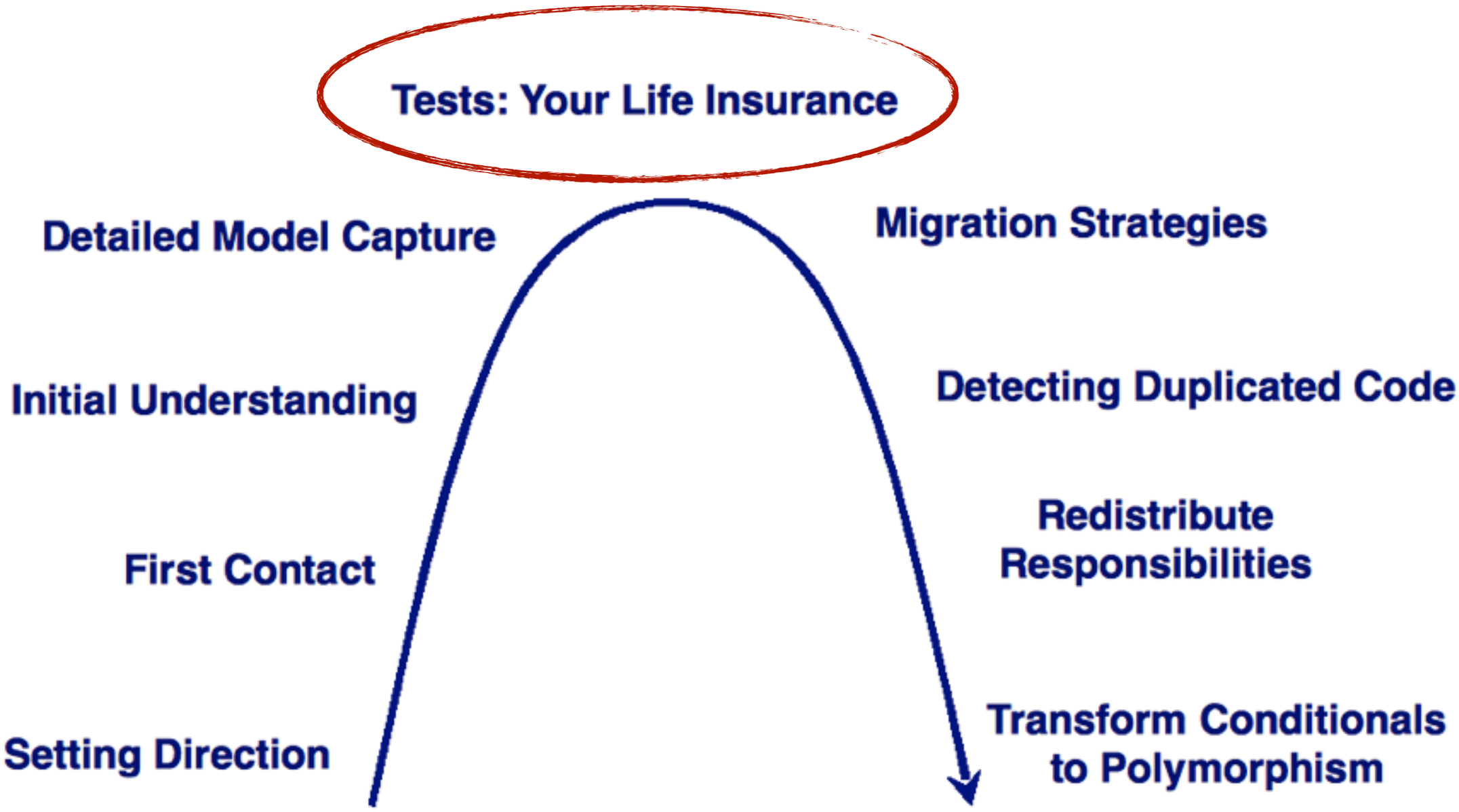
- **Forward Engineering**
  - traditional process of moving **from high-level abstractions** and logical, implementation-independent designs **to the physical implementation** of a system.
- **Reverse Engineering**
  - process of analyzing a subject system to identify the system’s components and their 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

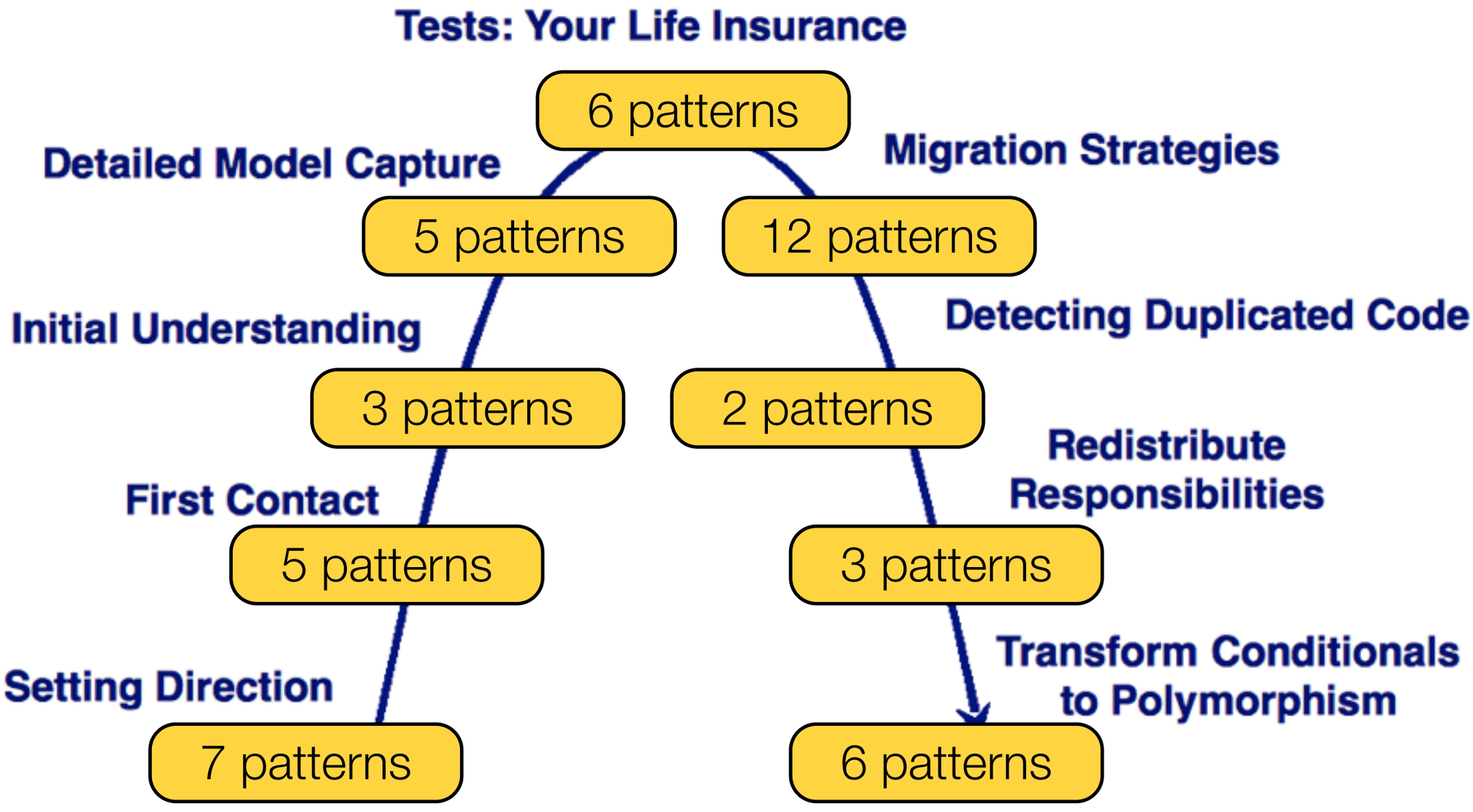


A Map of Reengineering Patterns



Based on https://www.iam.unibe.ch/scg/svn_repos/Lectures/OORPT/

A Map of Reengineering Patterns



Based on https://www.iam.unibe.ch/scg/svn_repos/Lectures/OORPT/

MASTER OF SCIENCE
IN ENGINEERING

Guidelines presentations

Schedule

https://docs.google.com/spreadsheets/d/14ZY6xSNyXjTvdPUKJ3Ked2H0MwxqKeqGrAXIlmovMGg/edit?usp=sharing

Common slide deck

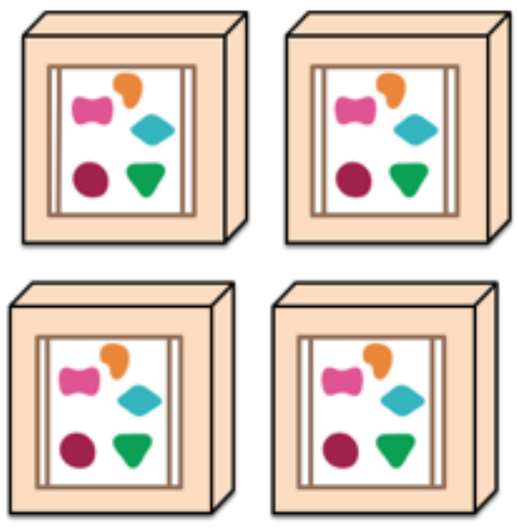
https://docs.google.com/presentation/d/1AMcviruGNDQtEF_fsg9tQ0W-zzHT_zbLBCKSPLsQmTo/edit?usp=sharing

Micro-service architecture

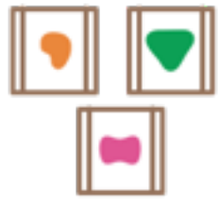
A new trend for building cloud applications

A monolithic application puts all its functionality into a single process...

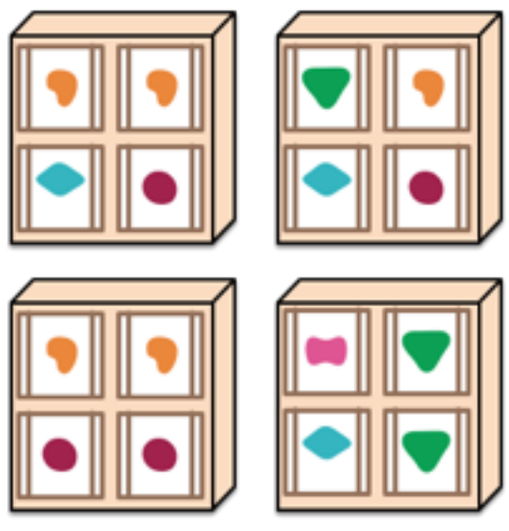
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...



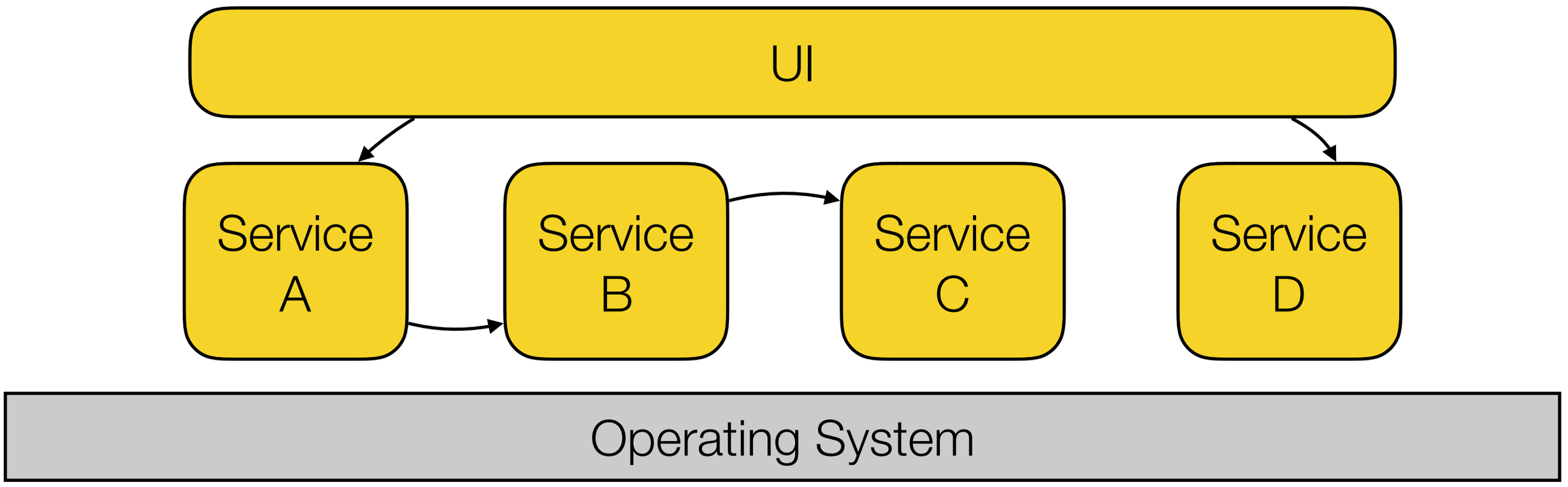
... and scales by distributing these services across servers, replicating as needed.



http://martinfowler.com/articles/microservices.html

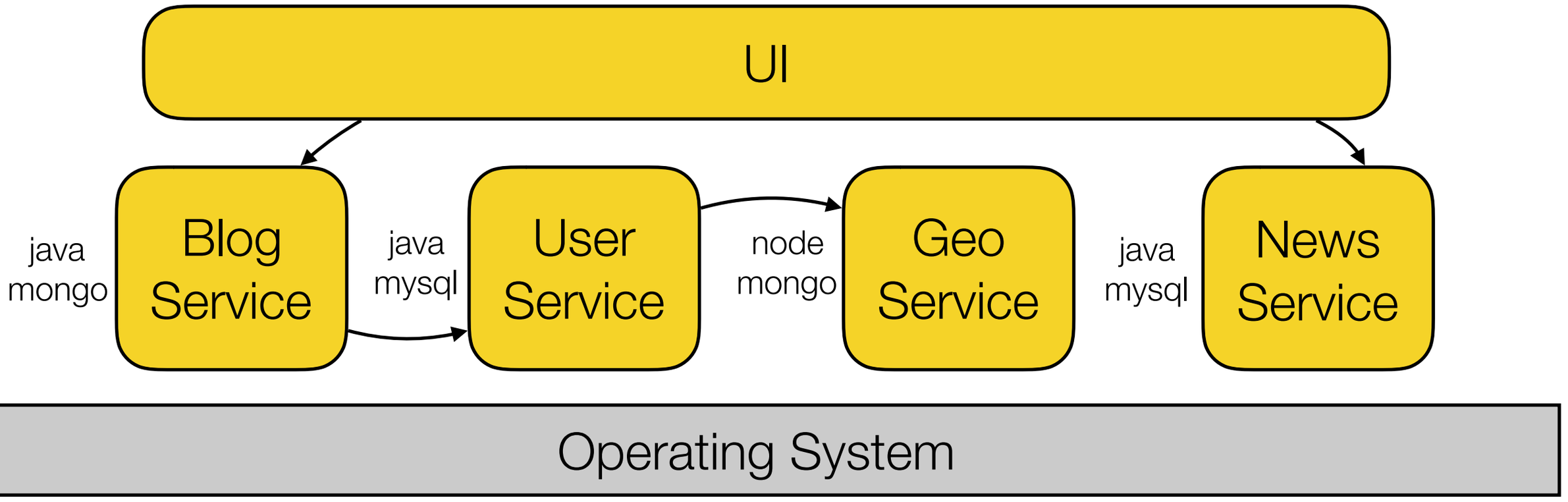
What is a micro service?

"In short, the **microservice architectural style** is an approach to developing a single application as a suite of small services, each **running in its own process** and communicating with **lightweight** mechanisms, often an HTTP resource **API**."

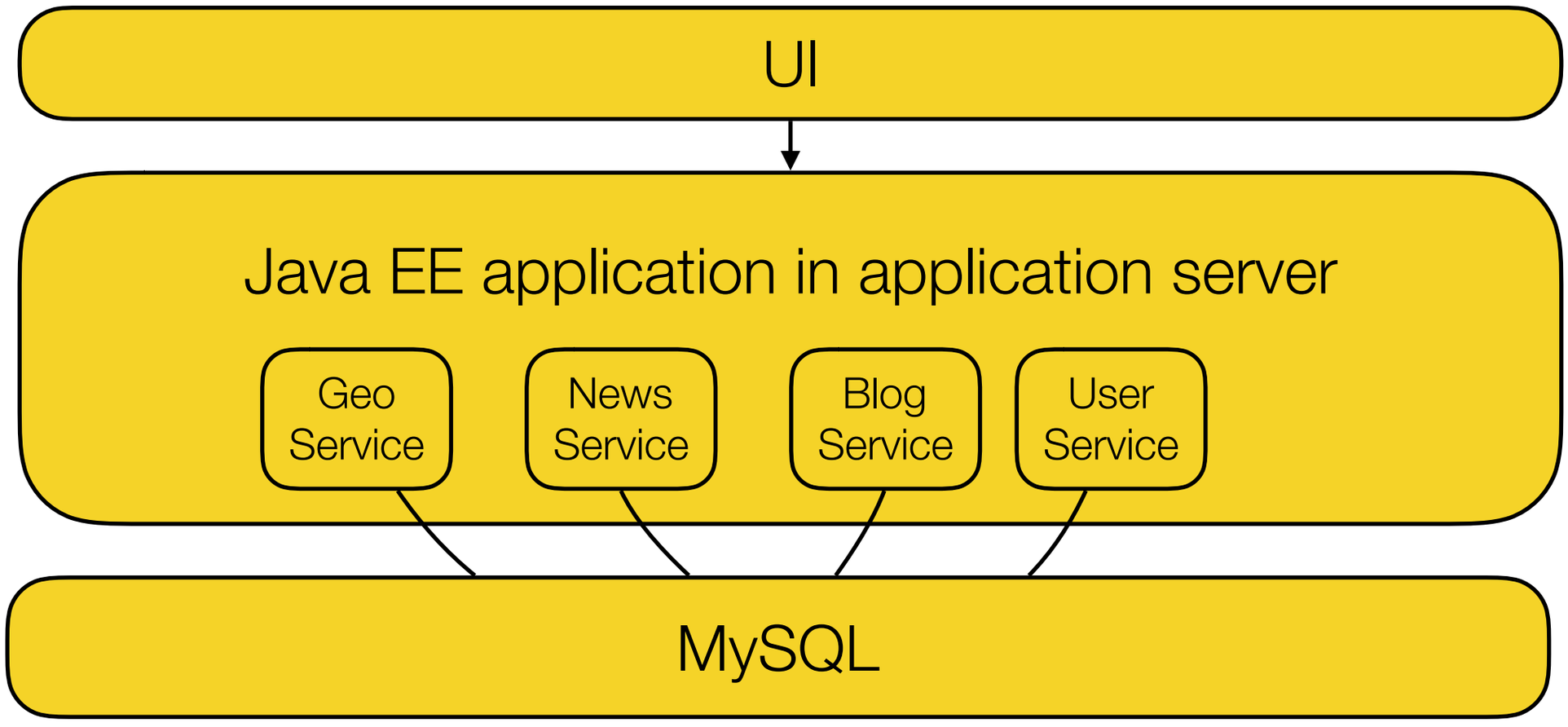


What is a micro service?

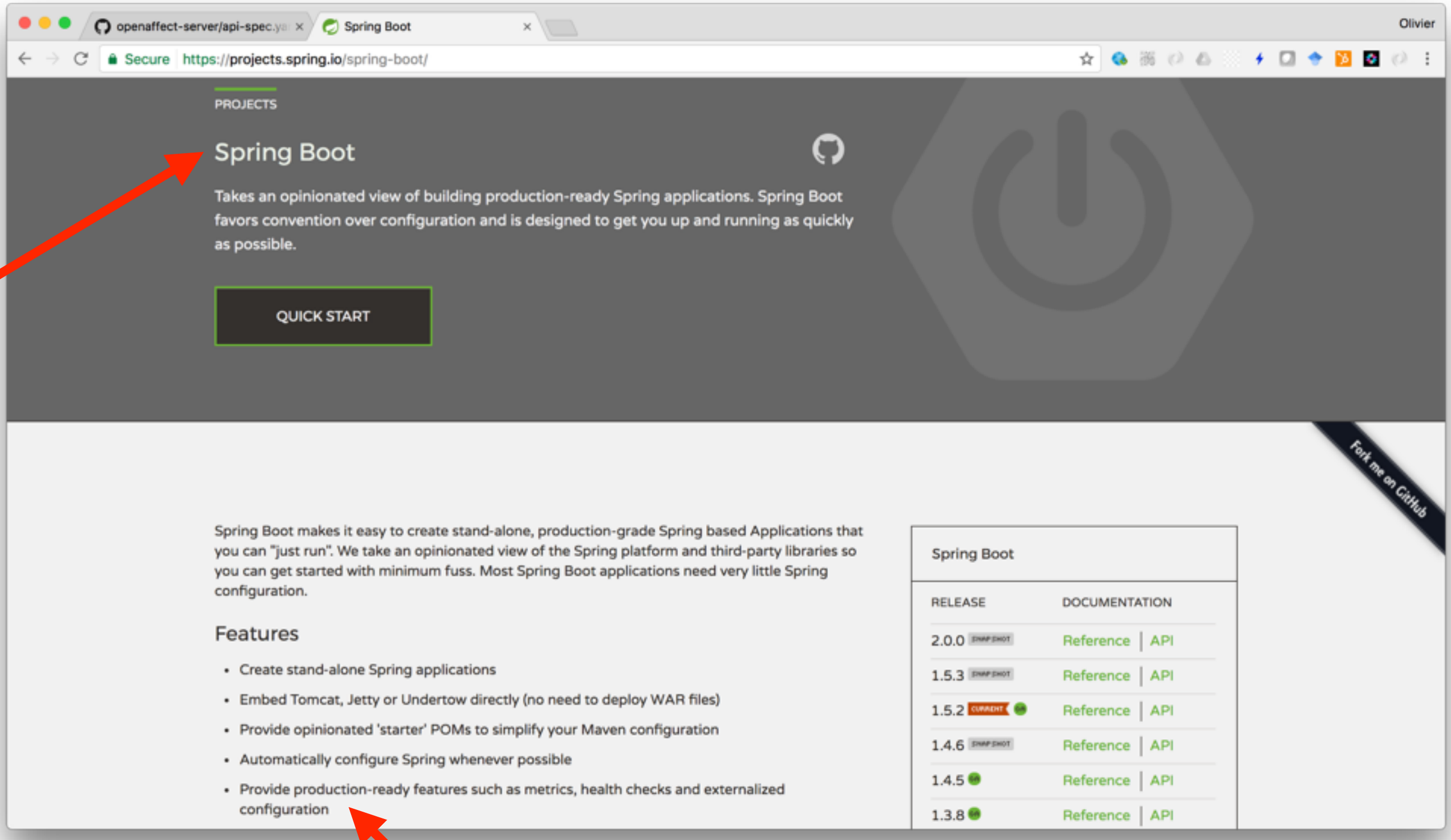
"These services are built around **business capabilities** and **independently deployable** by fully **automated deployment machinery**. There is a bare minimum of centralized management of these services, which **may** be written in different programming languages and use different data storage technologies."



Monolith vs micro-services



Having one large codebase and one deployment unit raises a number of issues. Microservices promise to address these issues (but beware of the hype and apply wisely!)


openaffect-server/api-spec.ya
Spring Boot
Olivier
Secure https://projects.spring.io/spring-boot/
PROJECTS
Spring Boot
Takes an opinionated view of building production-ready Spring applications. Spring Boot favors convention over configuration and is designed to get you up and running as quickly as possible.
QUICK START
Fork me on GitHub
Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run". We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need very little Spring configuration.
Features
Create stand-alone Spring applications
Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
Provide opinionated 'starter' POMs to simplify your Maven configuration
Automatically configure Spring whenever possible
Provide production-ready features such as metrics, health checks and externalized configuration
Spring Boot
RELEASE
DOCUMENTATION
2.0.0 SNAPSHOT Reference | API
1.5.3 SNAPSHOT Reference | API
1.5.2 CURRENT GA Reference | API
1.4.6 SNAPSHOT Reference | API
1.4.5 GA Reference | API
1.3.8 GA Reference | API

What is Spring Boot?

- The **Spring Framework** is one of the most widely used frameworks built on top of Java EE technologies.
- It was created more than a decade ago, to provide a "**lightweight**" alternative to full-fledged application servers (based on the principle that you do not have to use EJBs to write robust enterprise apps).
- **Beyond the framework**, Spring is associated to a large and very active **open source community**. It is also associated to a commercial company (**SpringSource**, now part of **Pivotal**).
- Spring is now a **large collection of projects**, that can be combined to create a complete platform. Spring Boot is one of these projects.

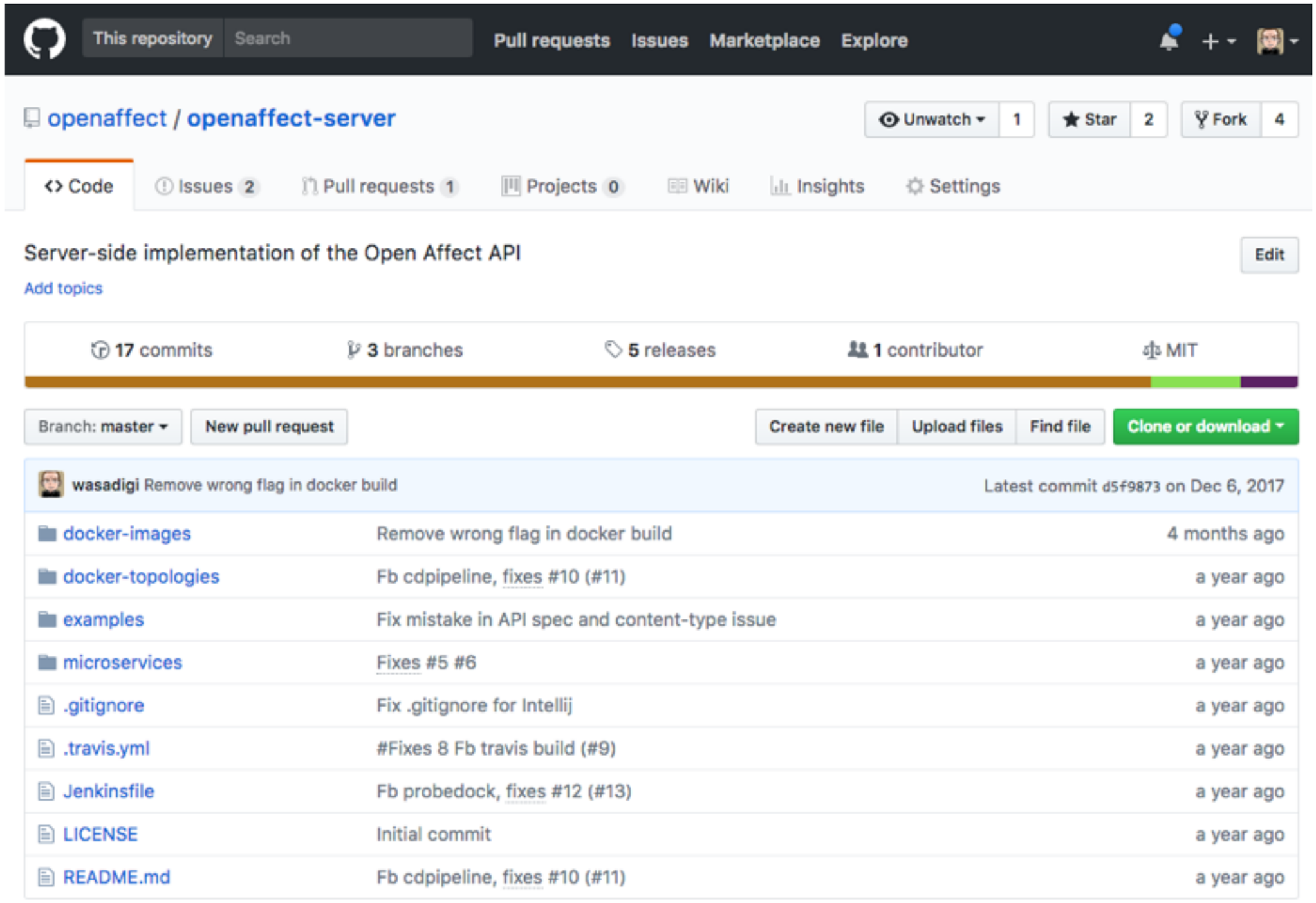
What is Spring Boot?

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run". We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need very little Spring configuration.

Features

- Create stand-alone Spring applications
- Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
- Provide opinionated 'starter' POMs to simplify your Maven configuration
- Automatically configure Spring whenever possible
- Provide production-ready features such as metrics, health checks and externalized configuration
- Absolutely no code generation and no requirement for XML configuration

Example project for the pipeline



http://goodrequirements.com/2013/pair-programming/

Discovering Spring Boot (1)

- **Step 1: build a minimal REST API**
  - Guide title: “Building a RESTful Web Service”
  - https://spring.io/guides/gs/rest-service/
  - You will have the choice between Gradle and Maven. Pick maven (which we will use when building the entire pipeline)
- **Step 1b: dockerize the result of step 1**
  - Create a Dockerfile that extends the java8 image
  - Add the executable .jar file generated by maven
  - Test your setup by building an image and running a container

Discovering Spring Boot (2)

- **Step 1: build a REST API with a (NoSQL) database in the back-end**
  - Guide title: “Accessing MongoDB Data with REST”
  - https://spring.io/guides/gs/accessing-mongodb-data-rest/
  - Do NOT install MongoDB on your host. Instead, use the Docker image available on Docker Hub.
- **Step 1: fully dockerize the result of step 1**
  - Create a Dockerfile that extends the java8 image
  - Add the executable .jar file generated by maven
  - Create a docker-compose.yml based on the example available in the OpenAffect repository (here).