Software Engineering Theory and Practice

School of Computing	
Title	Software Engineering Theory and Practice
Module Coordinator	Steven Ossont
Email	steven.ossont@port.ac.uk
Code	M30819
Moodle	https://moodle.port.ac.uk/course/view.php?id=11429

U30819: Software Engineering Theory and Practice

Evolution and Maintenance

Based on Ian Somerville, Software Engineering, 10th edition, Chapter 9

https://moodle.port.ac.uk/course/view.php?id=11429

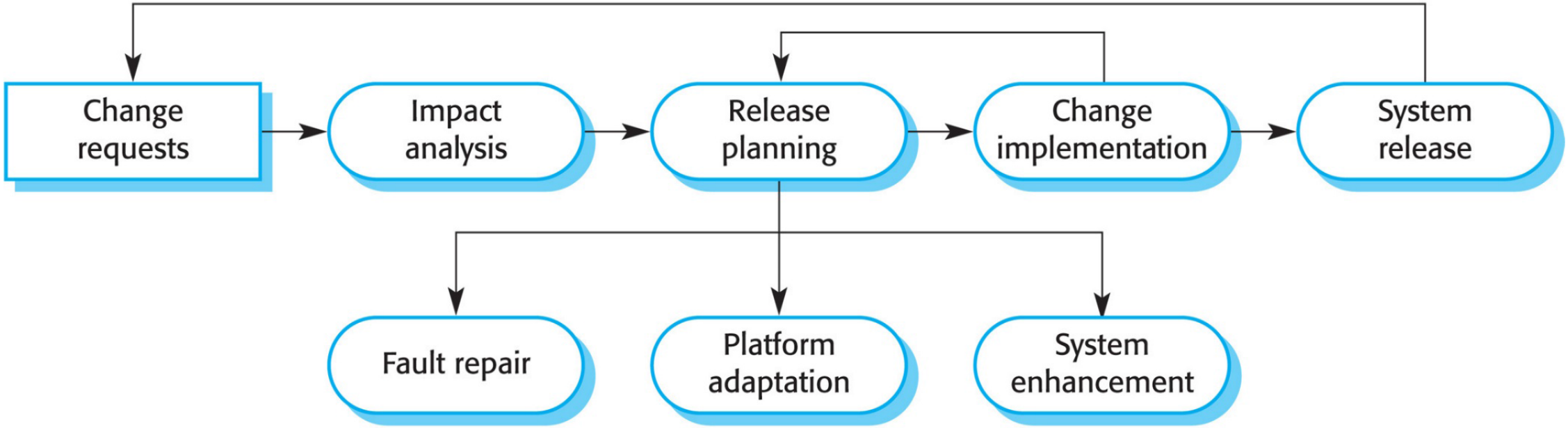
Software Evolution

- Software development does not stop when a system is delivered
- 85 - 90% of organizational software costs are evolution costs (Erlikh, 2000)

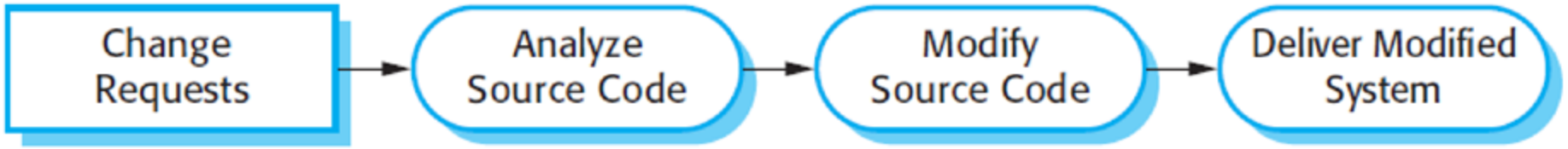
Triggers for software evolution

- Changing business requirements
- Reports of software defects
- Changes to other systems in a software system’s environment

Software Evolution Process



Emergency Change Requests



- A serious fault occurs that restricts the normal operation of the system
- Changes to the systems operating environment that have unexpected disruptive effects
- Unanticipated changes to the business running the system

Why Analyze Software Evolution?

- Nevertheless, the industrial track record raises the question, why, despite so many advances, [...]
  - Satisfactory functionality, performance, and quality are only achieved over a lengthy evolutionary process
  - Software maintenance never ceases until a system is scrapped
  - Software is still generally regarded as the weakest link in the development of computer-based systems

> Lehman et al., 1997
> http://www.inf.ed.ac.uk/teaching/courses/rtse/Lectures/lehmanslaws.pdf

Lehman's Laws of Software Evolution

I	Continuing Change	An E-type system must be continually adapted else it becomes progressively less satisfactory in use
II	Increasing Complexity	As an E-type system is evolved its complexity increases unless work is done to maintain or reduce it
III	Self regulation	Global E-type system evolution processes are self-regulating
IV	Conservation of Organisational Stability	Average activity rate in an E-type process tends to remain constant over system lifetime or segments of that lifetime
V	Conservation of Familiarity	In general, the average incremental growth (growth rate trend) of E-type systems tends to decline
VI	Continuing Growth	The functional capability of E-type systems must be continually enhanced to maintain user satisfaction over system lifetime
VII	Declining Quality	Unless rigorously adapted to take into account changes in the operational environment, the quality of an E-type system will appear to be declining as it is evolved
VIII	Feedback System	E-type evolution processes are multi-level, multi-loop, multi-agent feedback systems

An E-type system is a real world system

Why Analyze Software Evolution?

- Goal: investigate the evolution of a software system to identify potential shortcomings in its architecture or logical structure
- Structural shortcomings can then be subjected to reengineering or restructuring

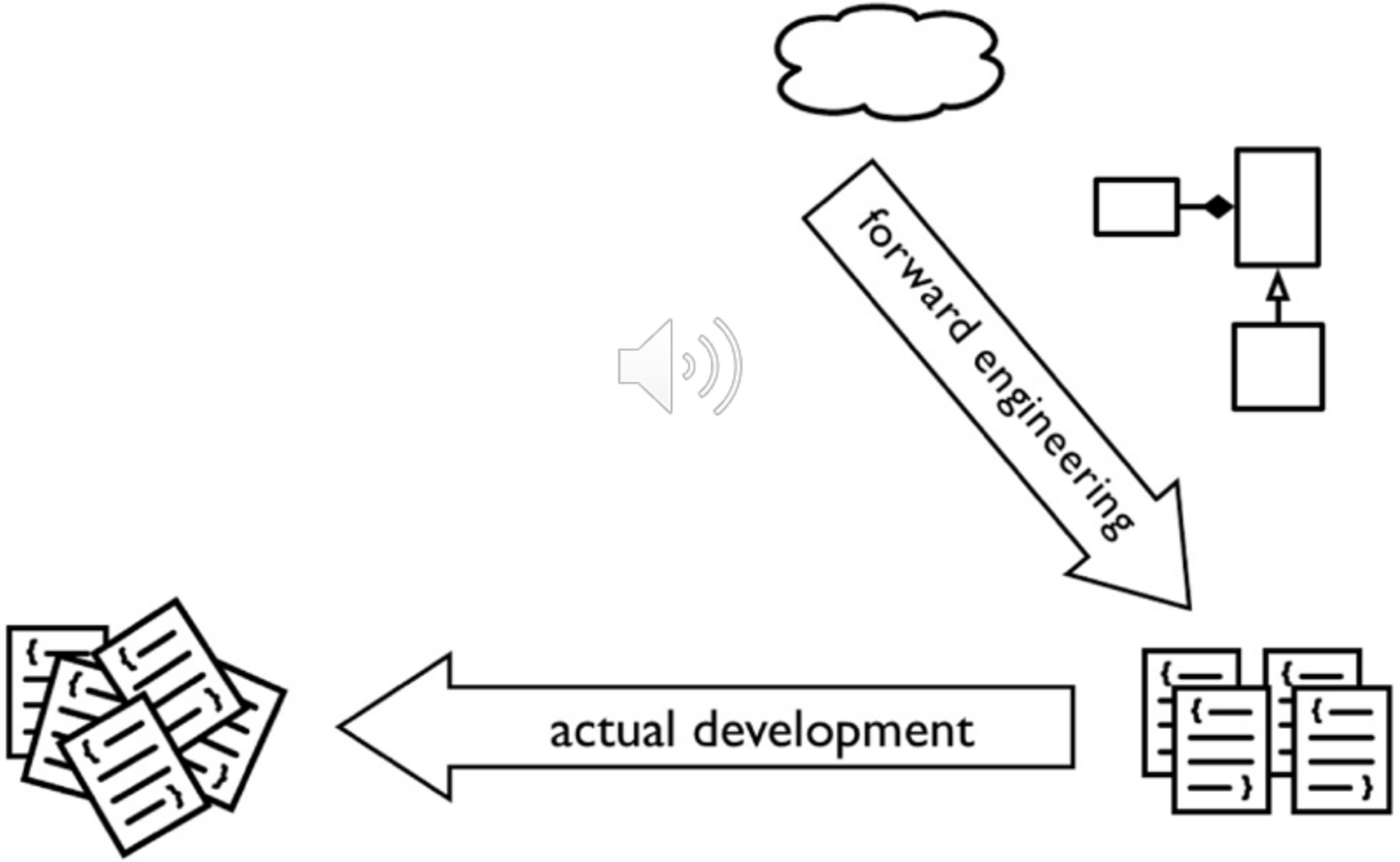
https://www.slideshare.net/michele.lanza/soft-evo

Software Reverse Engineering

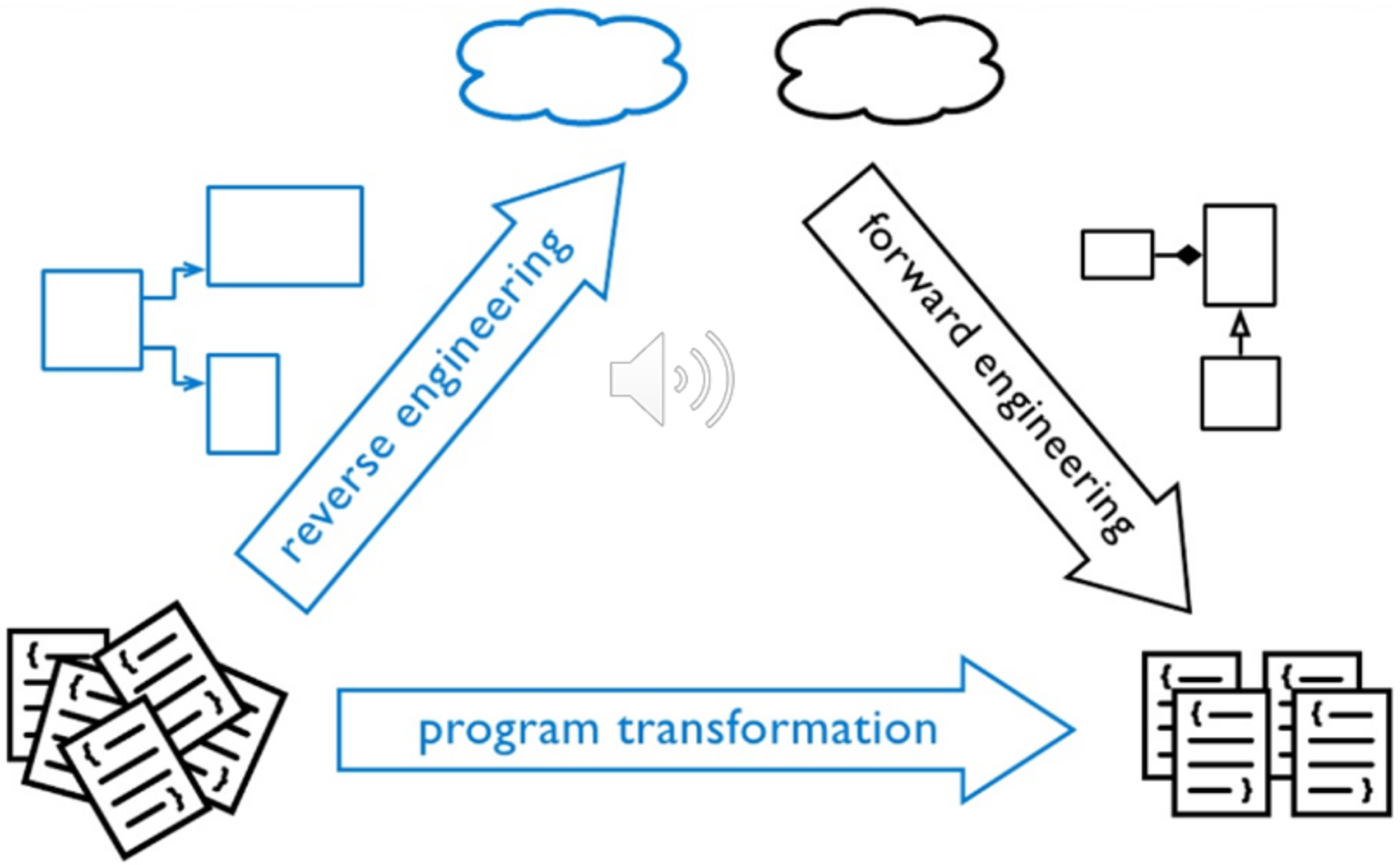
- The process of analysing a subject system to:
  - Identify the system’s components and their interrelationships
  - Create representations of the system in another form or at a higher level of abstraction” (Chikofsky&Cross, 1990)
- Goal: understand other people’s code – newcomers in the team, code reviewing

https://www.slideshare.net/michele.lanza/soft-evo

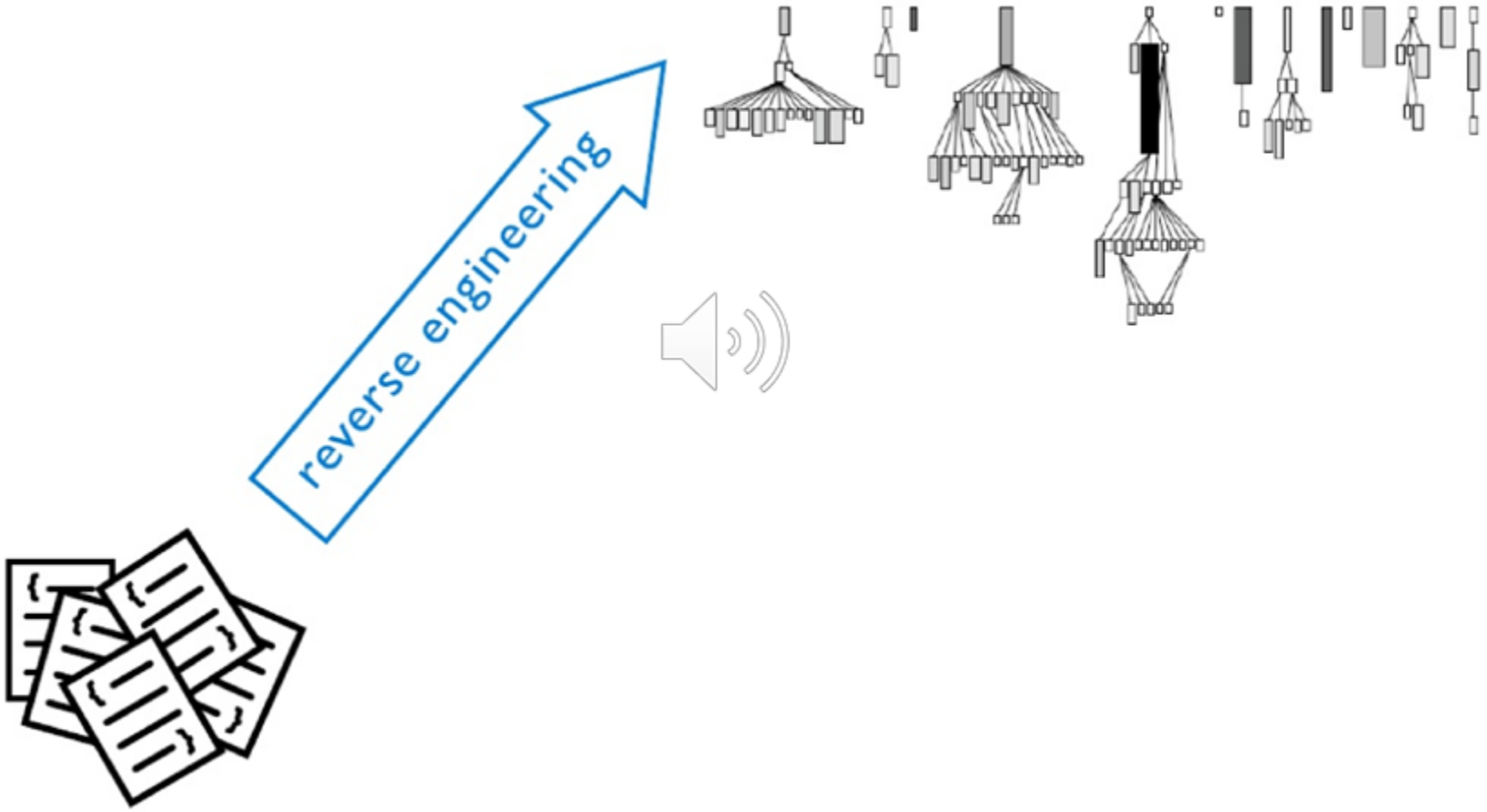
Software Reverse Engineering



Software Reverse Engineering



Software Reverse Engineering



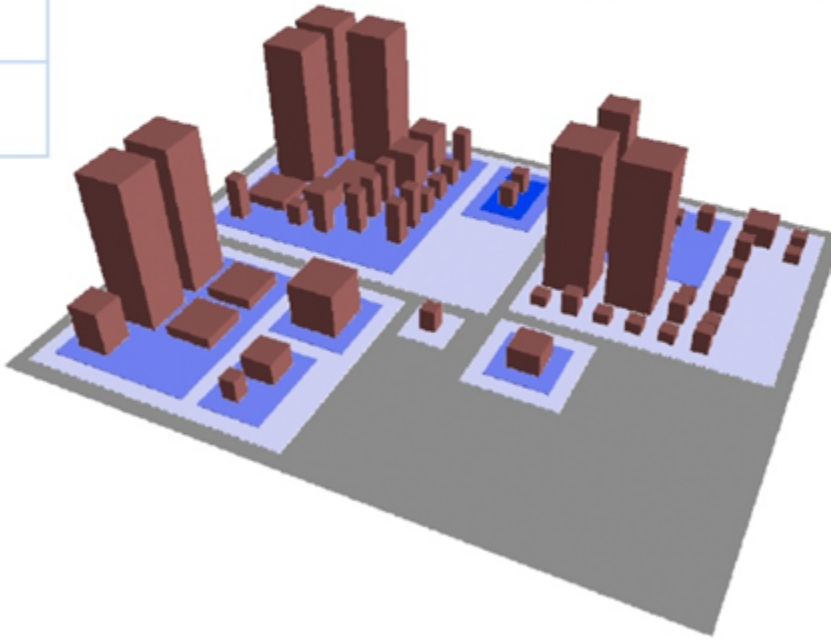
CodeCity

Tool for software analysis - *software systems are visualized as interactive, navigable 3D cities*

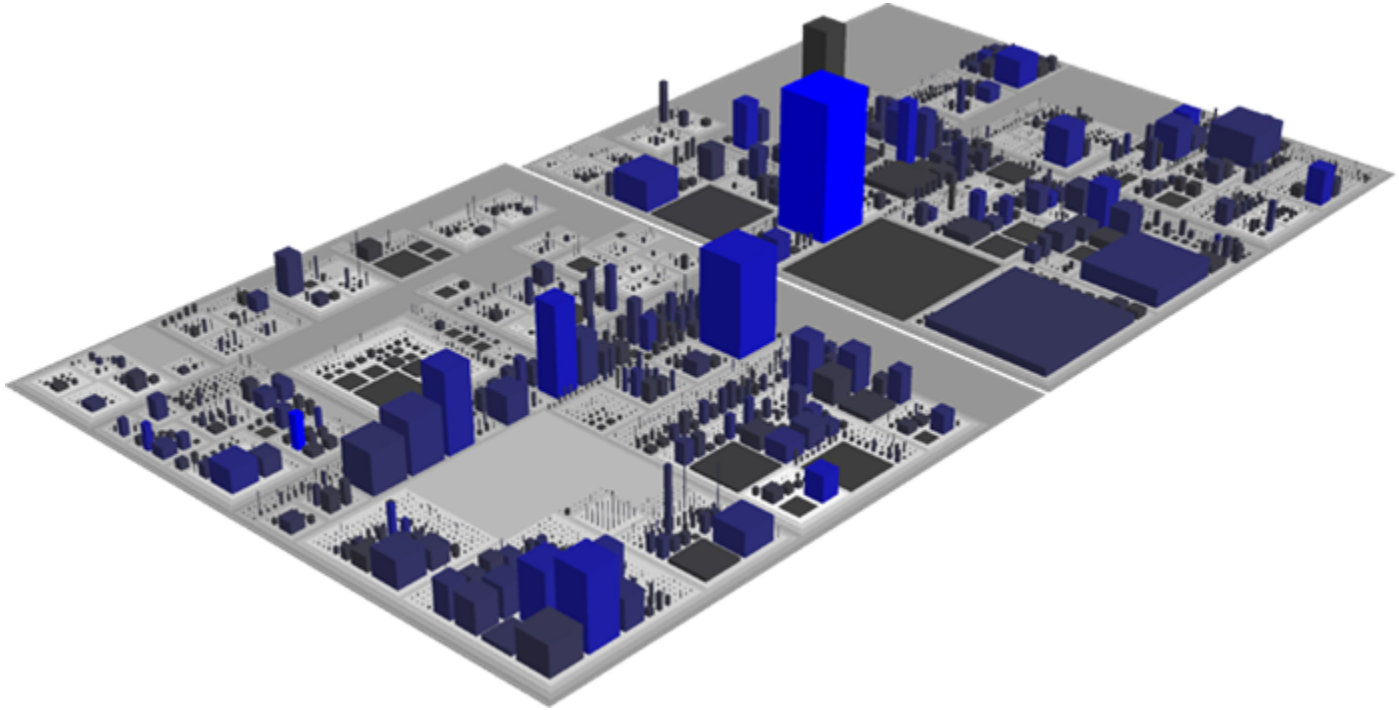
software	representation
classes	buildings
packages	districts
system	city

class metric	building property
number of methods	height
number of attributes	width
number of attributes	length

package metric	district property
nesting level	color saturation

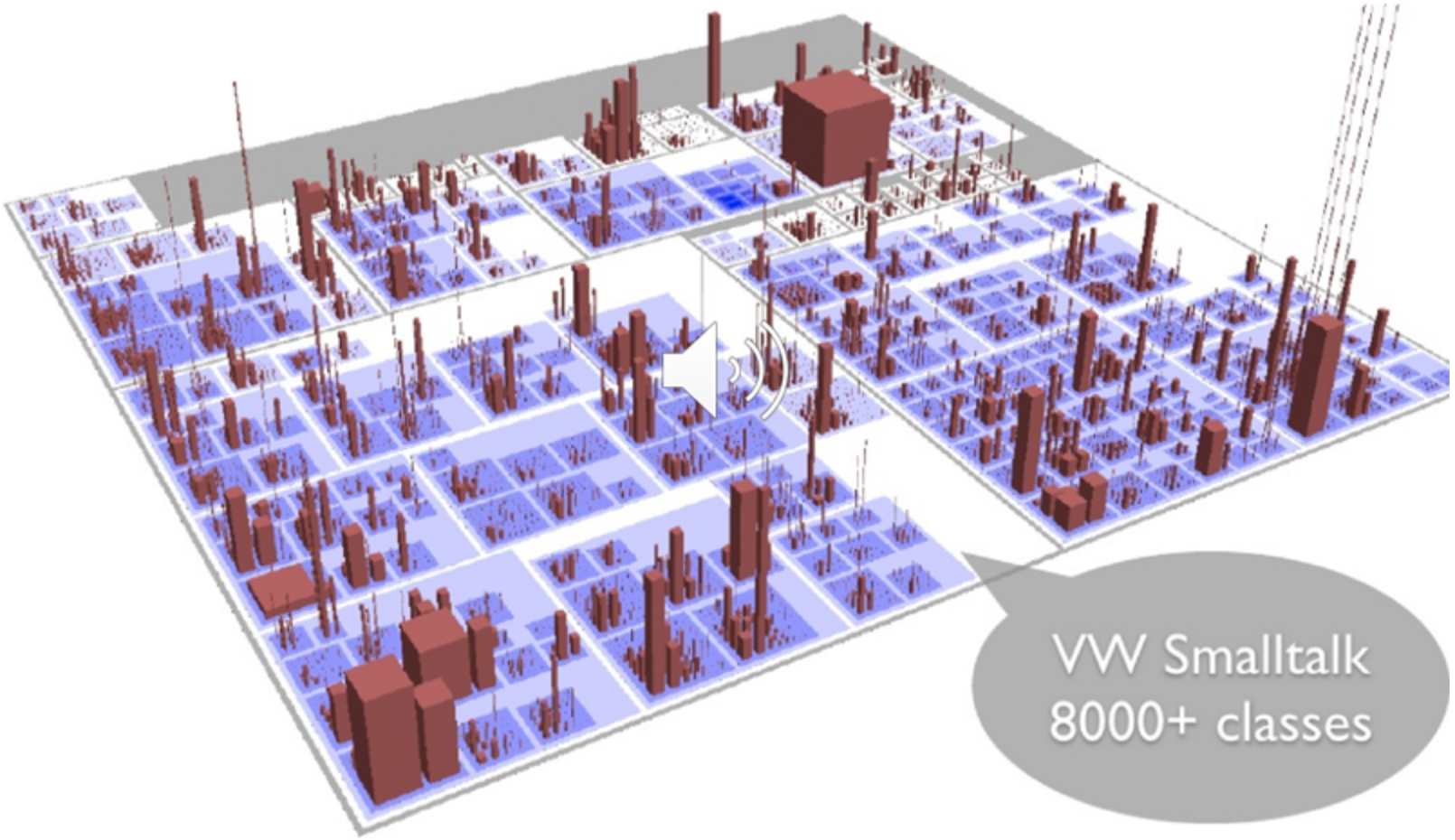


JDK Representation in CodeCity

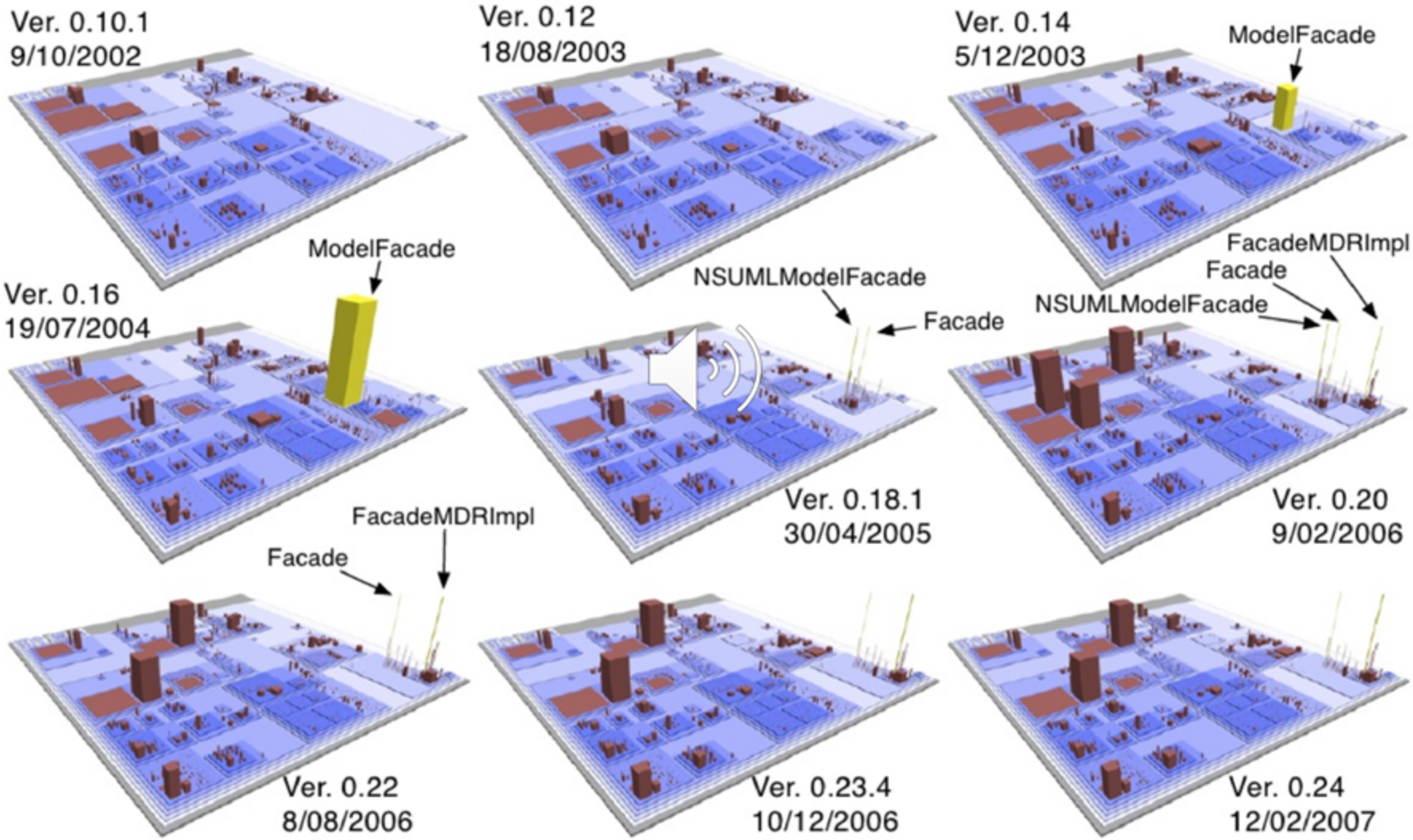


> https://wettel.github.io/codecity.html

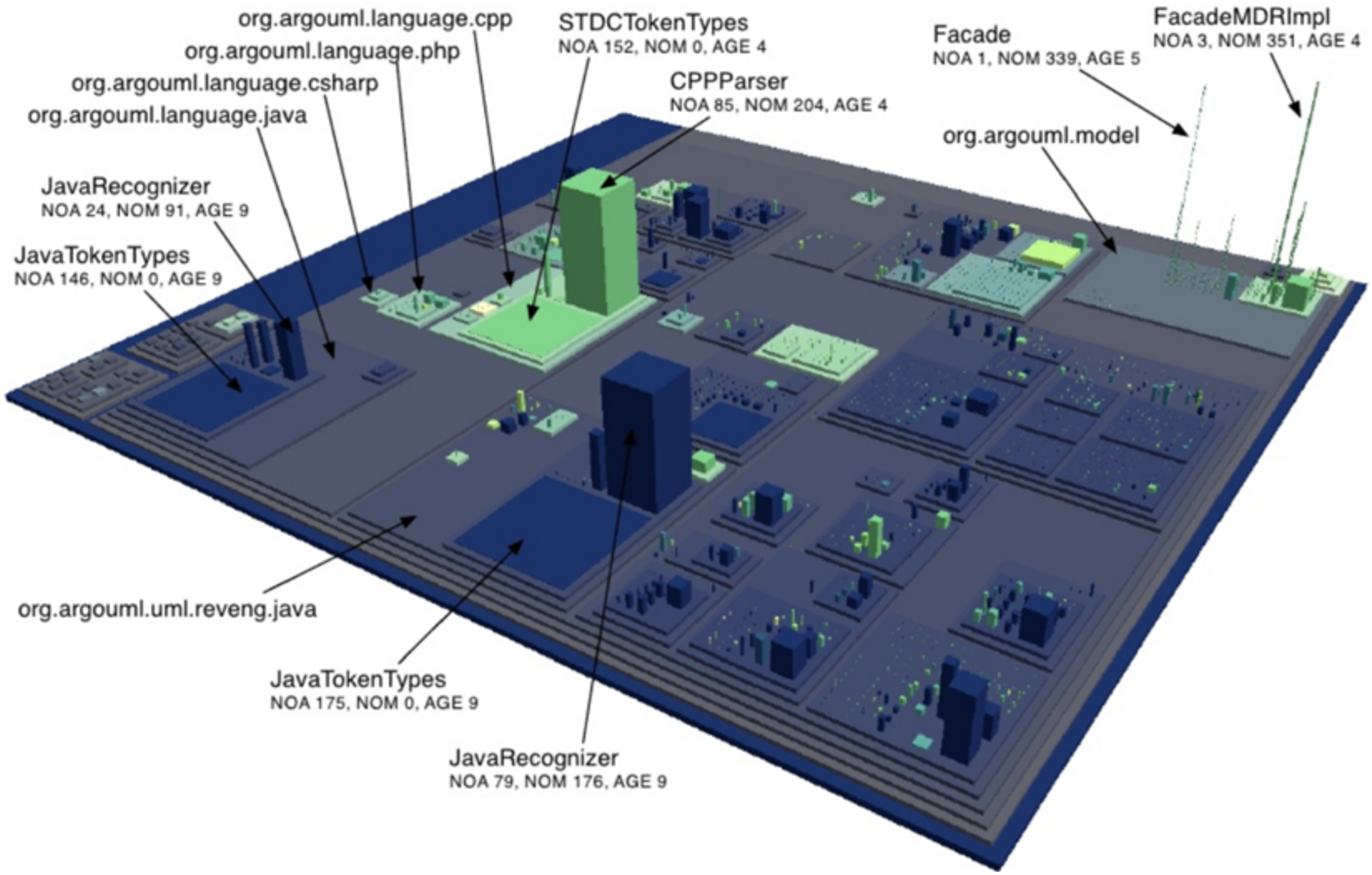
Smalltalk Representation in CodeCity



ArgoUML over time



ArgoUML Age Map



Software Maintenance

- General process of changing a system after it has been delivered
  - Correct coding errors
  - Correct design errors
  - Enhancements to correct specification errors or accommodate new requirements

Types of Software Maintenance

- Corrective
  - Correcting errors in code, design, or requirements
- Adaptive
  - Some aspects of the system’s environment, such as the hardware or the OS change
- Perfective
  - Modifications to improve performance or maintainability
- Preventive
  - Regularly performed on a piece of equipment to lessen the likelihood of it failing

Types of Software Maintenance

- Corrective
  - Policies for reporting and fixing errors
  - Requires strategy – negotiation with client
- Perfective
  - Treated as development – iterative development approaches
- Adaptive and Preventive
  - Monitoring, scheduling

Software Maintenance Efforts

- Corrective (21%)
  - 12.4% emergency debugging
  - 9.3% routine debugging

Software Maintenance Efforts

- Adaptive (25%)
  - 17.3 % data environment adaptation
  - 6.2% changes to hardware or OS

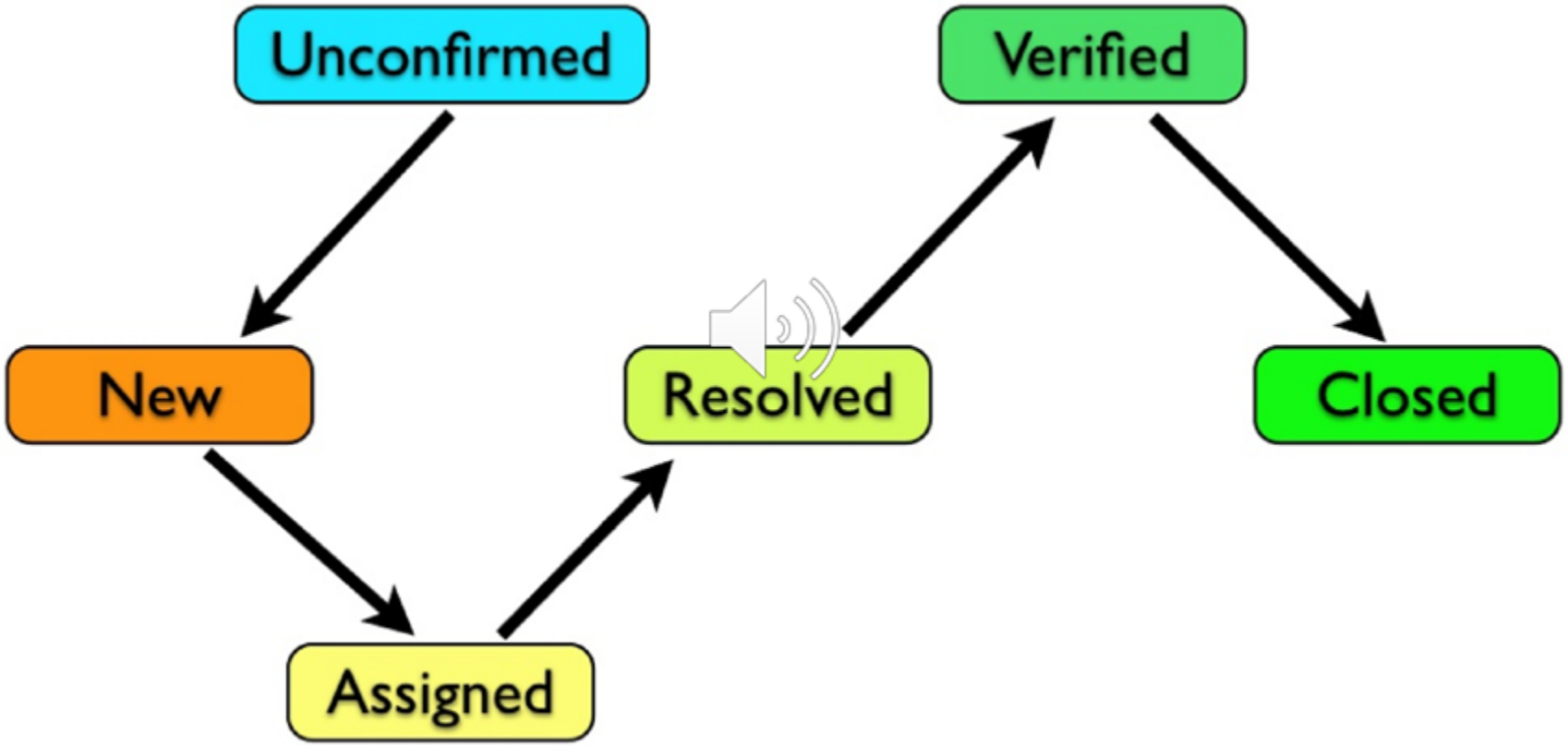
Software Maintenance Efforts

- Perfective (50%)
  - 41.8% enhancements for users
  - 5.5% improve documentation
  - 3.4% other

Software Maintenance Efforts

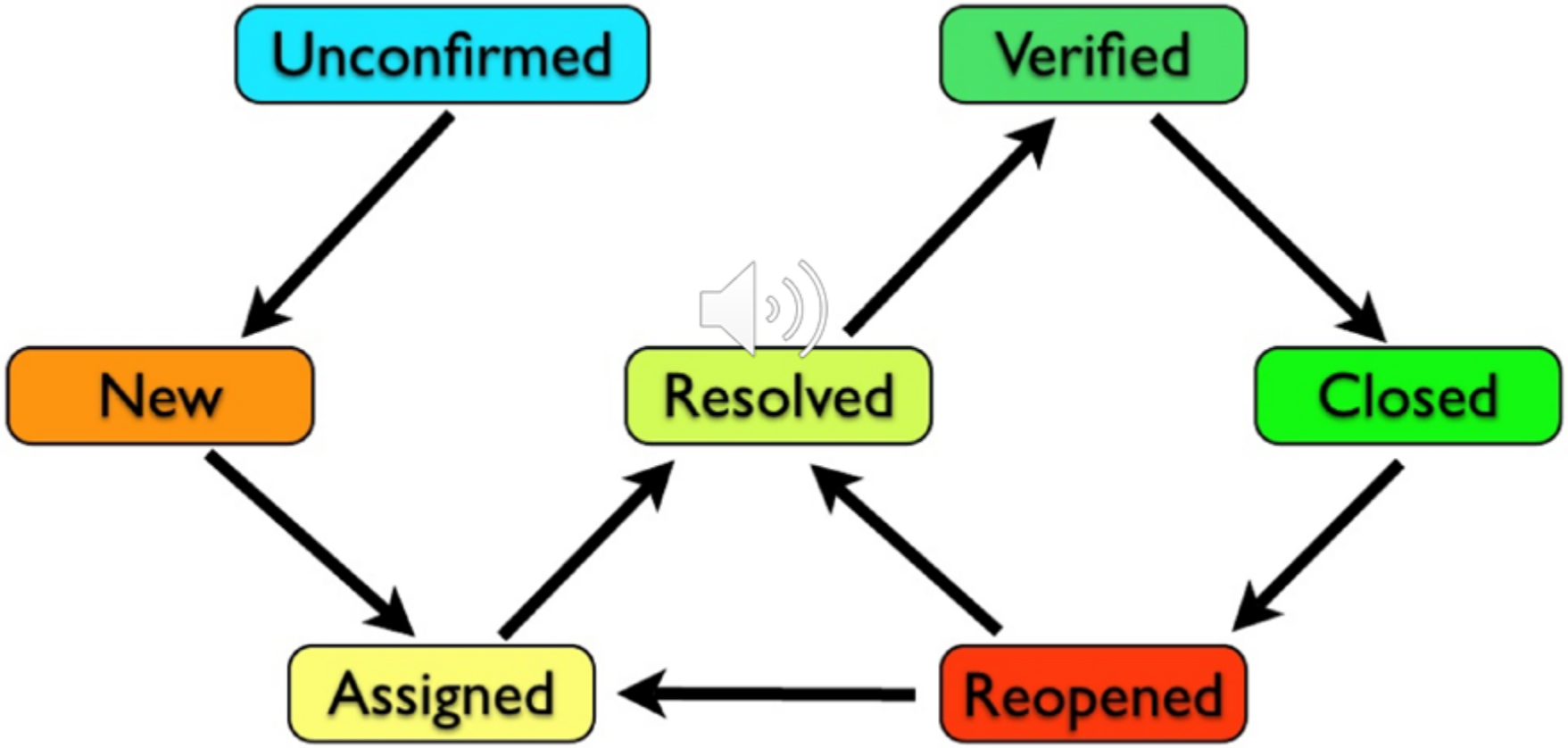
- Preventive (4%)
  - 4% improve code efficiency

A Bug's Life – Ideal Case Scenario

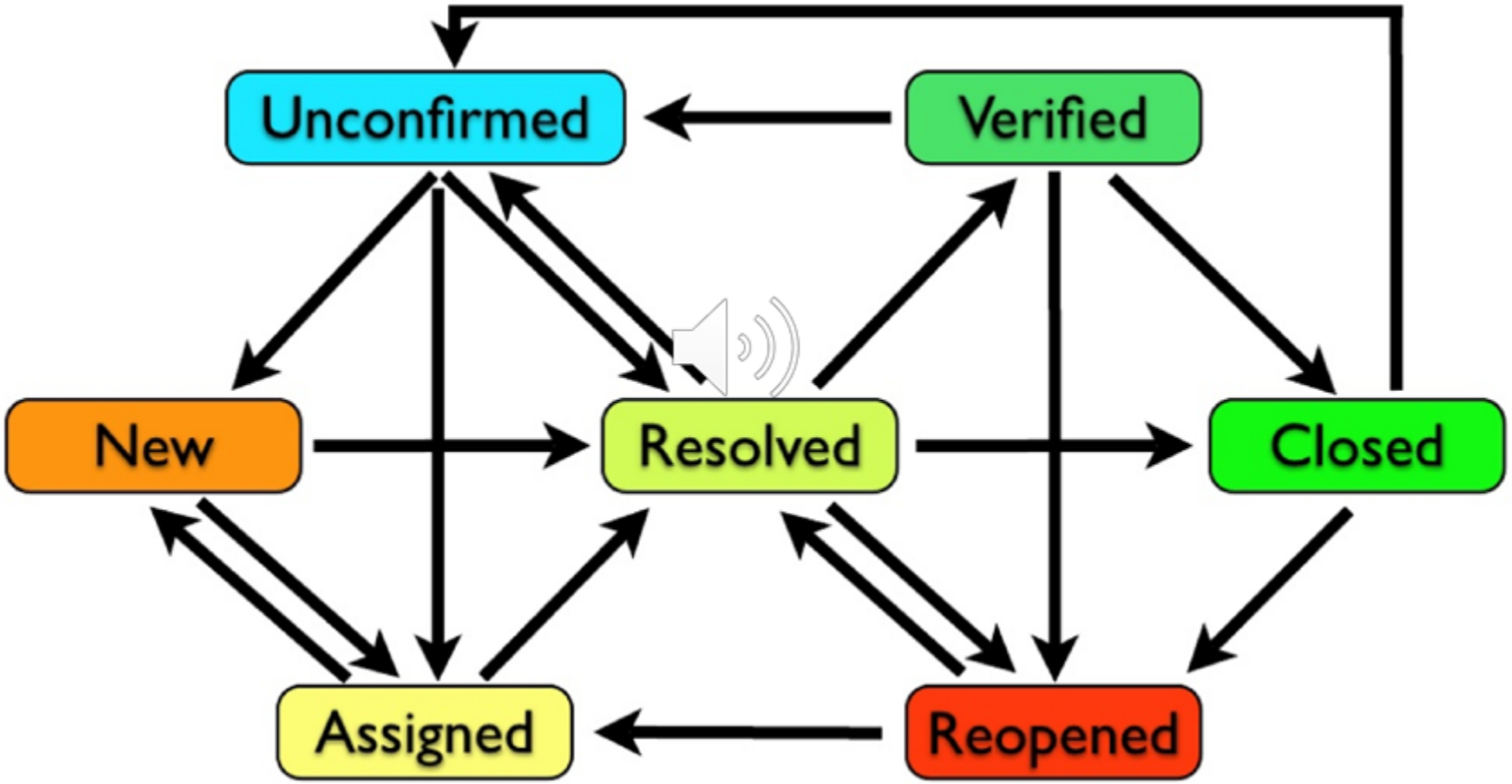


> https://www.slideshare.net/michele.lanza/soft-evo2

A Bug's Life – Less Ideal Case Scenario

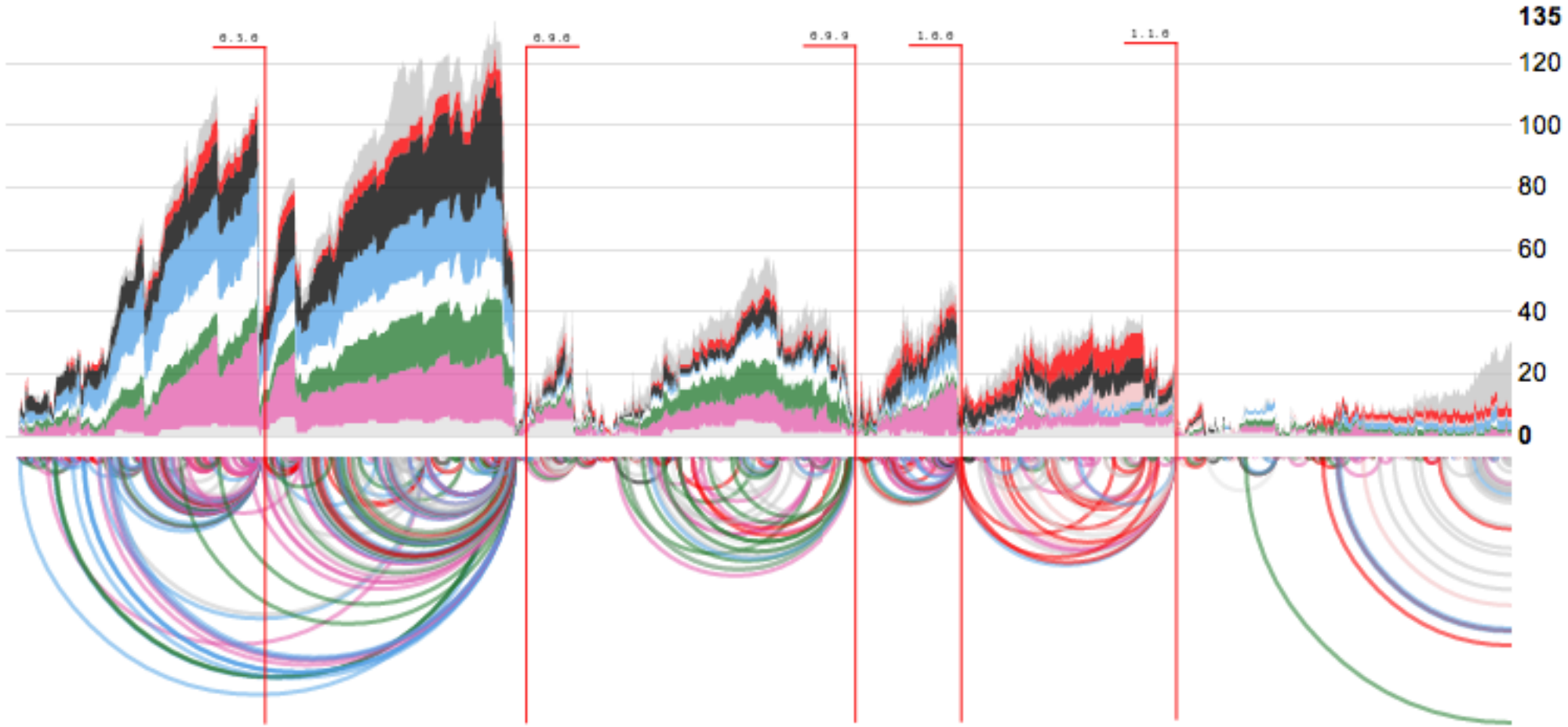


A Bug's Life – Real Case Scenario



Bug life

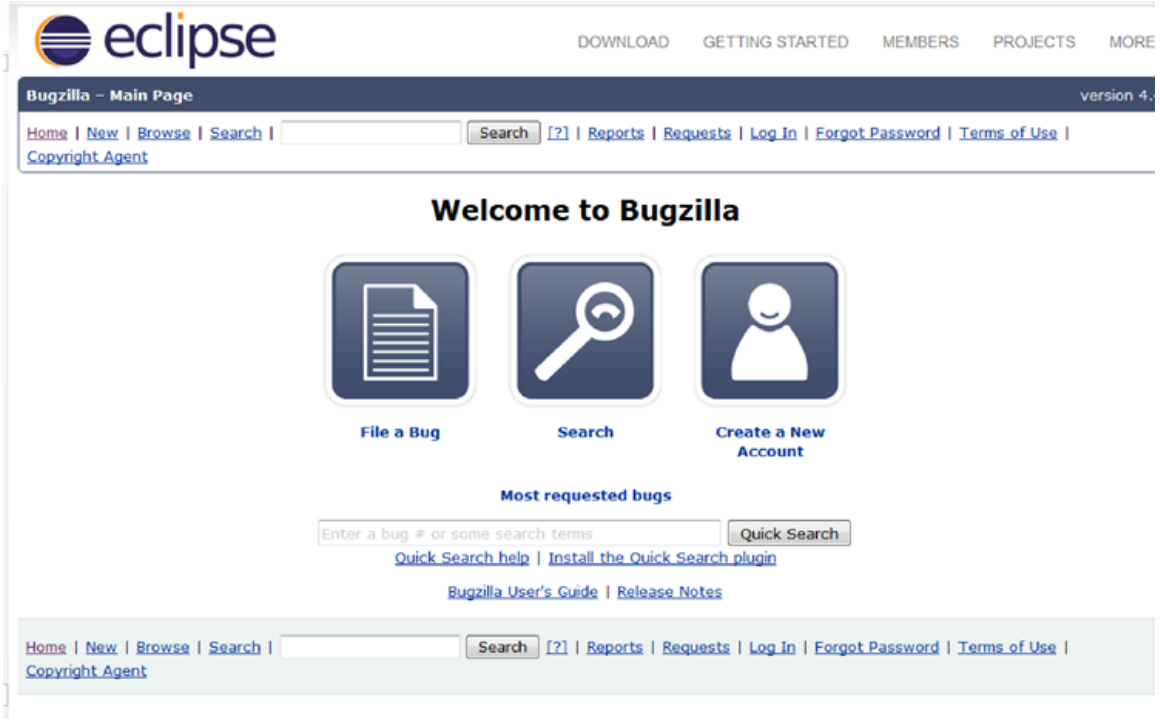
> https://9-volt.github.io/bug-life/


0.5.0
0.9.0
0.9.9
1.0.0
1.1.0
135
120
100
80
60
40
20
0

Bugzilla

https://www.bugzilla.org/

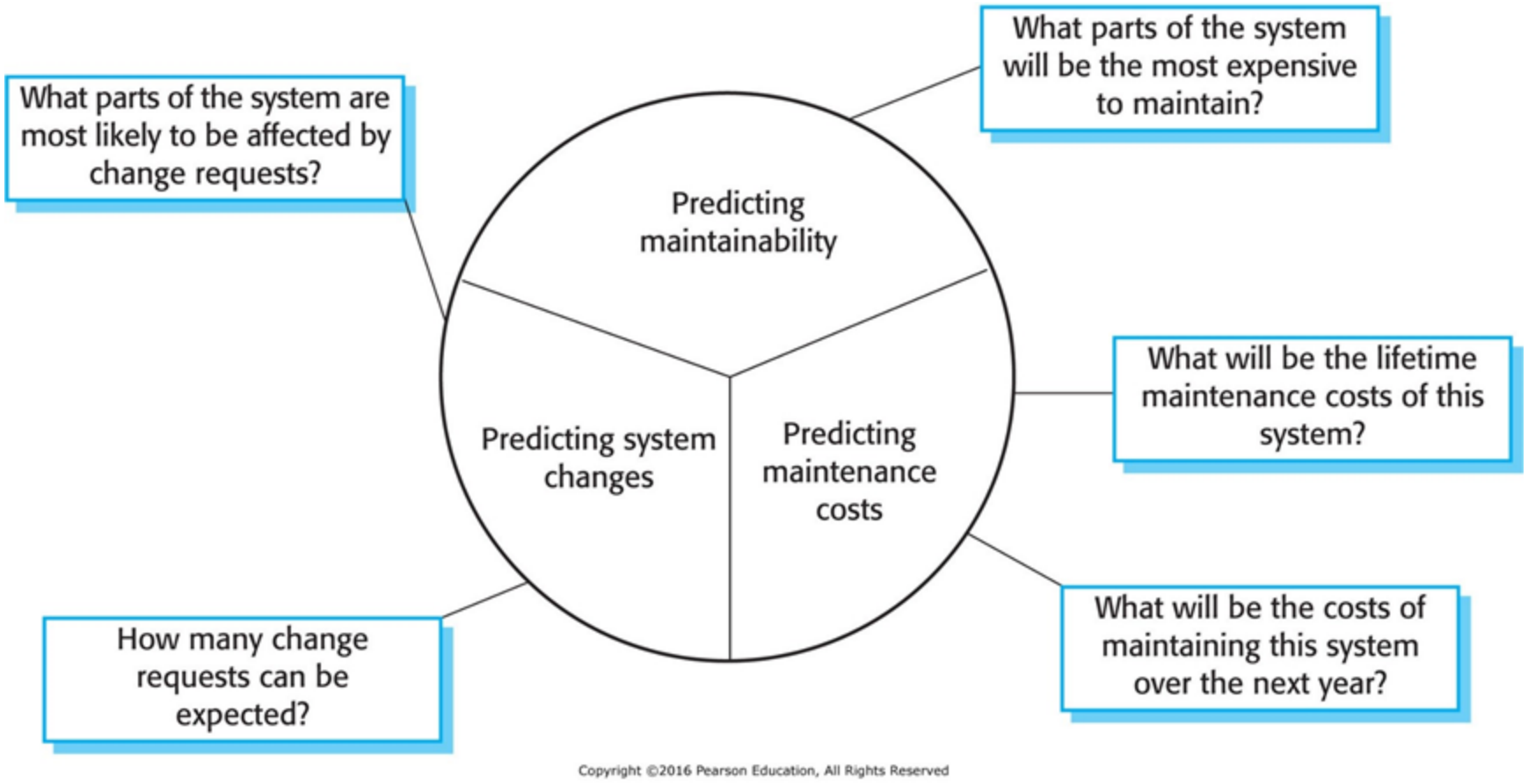
- e.g. https://bugs.eclipse.org/bugs/



Maintenance Problems

- Poor quality of documentation: code, design, requirements
- User demand for enhancement and extensions
- Competing demands for maintainer’s time
- Difficulty in meeting scheduled commitments
- Turnover in user organisations
- Systems not robust under change
- Key design concepts are not captured
- Attitude towards maintenance

Maintenance Prediction



Assessing Maintainability

- Number of requests for corrective maintenance
- Average time required for impact analysis
- Average time taken to implement a change request
- Number of outstanding change requests

Questions?