Exam Preparation

Software Maintenance & Evolution

Final Exam Info

- Written Exam
- Same time slot, same room as the lecture
- Covers the entire course
- Be as concise and short as possible, but no shorter with your answers
- Use non-red permanent ink pencils
- Write down only one single solution

In the following...

- Possible answers are presented
- There are not exhaustive
- There might be other answers or additional points to consider
- The title of a slide indicates the task number from the exam, date of the lecture relevant to answer the questions, and if available slide numbers
- No warranty!

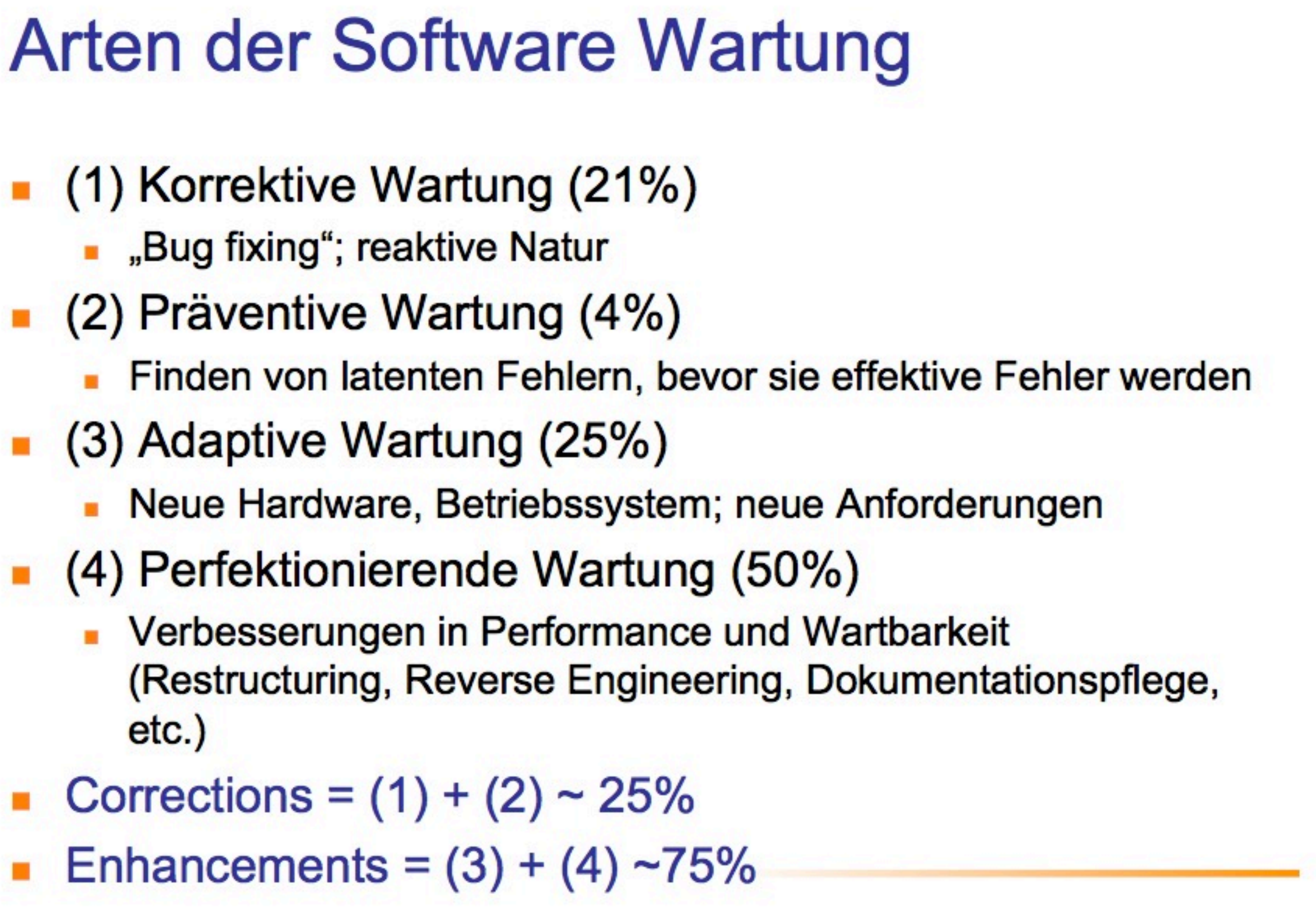
1a, 22.2 Slides 7, 28

- “the modification of a software product ***after delivery*** to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment”
- “standard for software life cycle processes depicts maintenance as one of the primary life cycle processes”
- “as the process of a software product undergoing modification to code and associated documentation due to a problem or the need for improvement. The objective is to modify the existing software product while preserving its integrity”

1a, 22.2 Slides 29

- Corrective Maintenance
- Preventive Maintenance
- Adaptive Maintenance
- Perfective Maintenance

1a, 22.2 Slide 29



Arten der Software Wartung

- (1) Korrektive Wartung (21%)
  - „Bug fixing“; reaktive Natur
- (2) Präventive Wartung (4%)
  - Finden von latenten Fehlern, bevor sie effektive Fehler werden
- (3) Adaptive Wartung (25%)
  - Neue Hardware, Betriebssystem; neue Anforderungen
- (4) Perfektionierende Wartung (50%)
  - Verbesserungen in Performance und Wartbarkeit (Restructuring, Reverse Engineering, Dokumentationspflege, etc.)
- Corrections = (1) + (2) ~ 25%
- Enhancements = (3) + (4) ~75%

2a, 1.3 & 8.3

- Process of improving the internal structure of the code
- During this process the external behavior, i.e., the functionality, of the system does not change
- Part of reverse engineering, e.g., refactor to understand
- Refactor to test
- Part of reengineering, e.g., remove duplicated code
- Can also occur during daily development work outside/without a larger reengineering project

1c, 22.02 Paper

- A series of Law's that describe principles of software evolution
- Discusses the driving forces behind the software evolution
- http://www.ifi.uzh.ch/seal/teaching/courses/SWEvo13/lehman-IEEE-80.pdf

1c, 22.02 Paper

- Law of increasing Entropy: *As an evolving program is continually changed, its complexity, reflecting deteriorating structure, increases unless work is done to maintain or reduce it.*
- Untangle dependency structure
- Extract Method
- Make function calls simpler
- In general: Refactoring as "cure" from a deteriorating structure

2 a

- Not explicitly mentioned on slides
- Postconditions: Adapt all method calls with the new method name (if not there will be compile and build errors)
- Precondition: Code needs to be compile-able to find all method calls

2b, 22.3

- Does not take into account the control flow
- Does not take into account the dependency structure
- Depends significantly on the programming language
- Are 10 methods with 20 LOC as complex as 1 big method with 200 LOC?
- Just a single, plain number: Hard for developers to deal with them

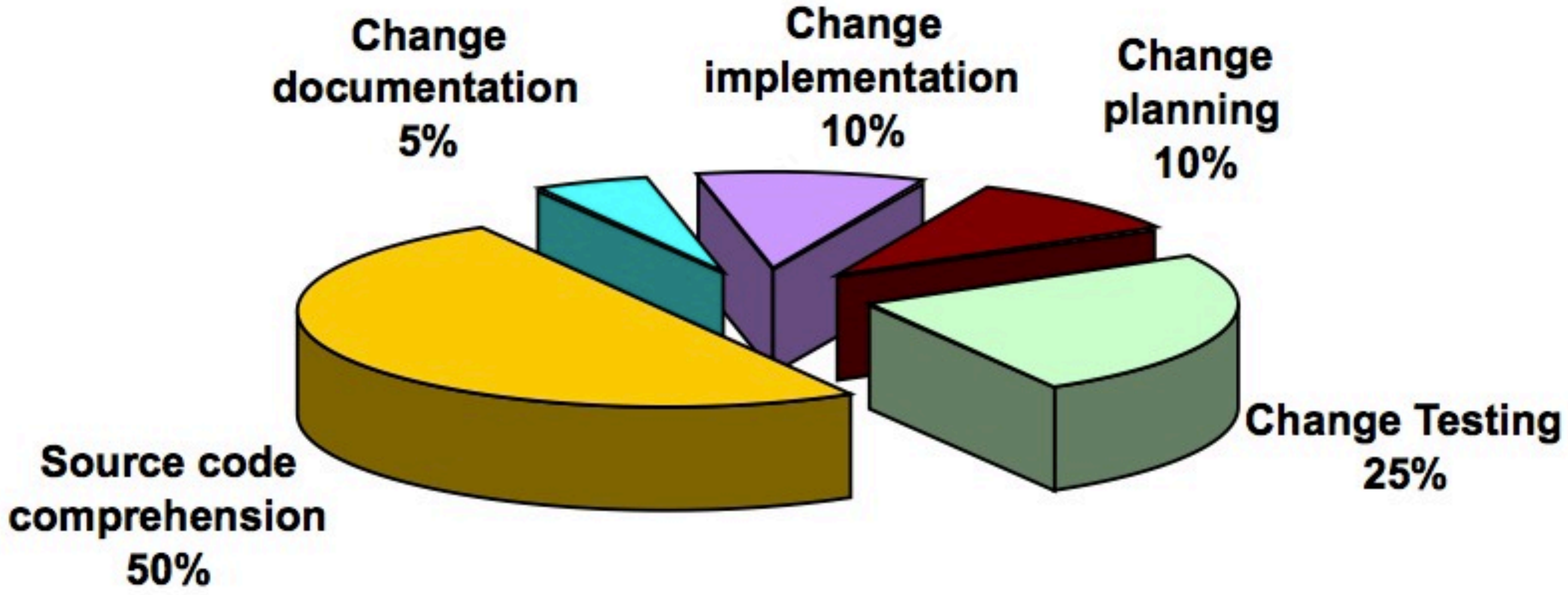
2b, 22.3

- Better: Metrics based on reference points or metrics that are independent of system size (or other individual characteristics of a system)
- FAN-IN & FAN-OUT
- Cyclomatic Complexity

2c, 1.3

- Changing one module leads to a rat-tail of changes in other modules
- More defects
- More difficult to understand
- Longer build times
- Changes take longer to implement
- More testing needed
- …

2d, 22.2 Slide 12 Part II



2e

- Not explicitly mentioned in lecture slides
- Source configuration management tool: Any tool (suite) that facilitates the management of code and artifacts: Version control systems, automated build and test system, continuos integration
- Defect tracker: Database for reporting new defects, discuss current defects, submit patches, ..
- There are products that integrate both functionalities

2f, 1.3

- Captures the context of a system
- Can give information about the rationale of the current implementation
- Can help to identify the important parts of a system

2g, 22.3

The left side: System Size & Complexity

Direct metrics: NOP, NOC, LOC, CYCLO

Derived Metrics: NOC/P, NOM/C, LOC/M, Cyclo/LOC

The right side: System Coupling

Direct metrics: CALLS, FANOUT

Derived Metrics: CALLS/M, FANOUT/CALL

What about reference points?

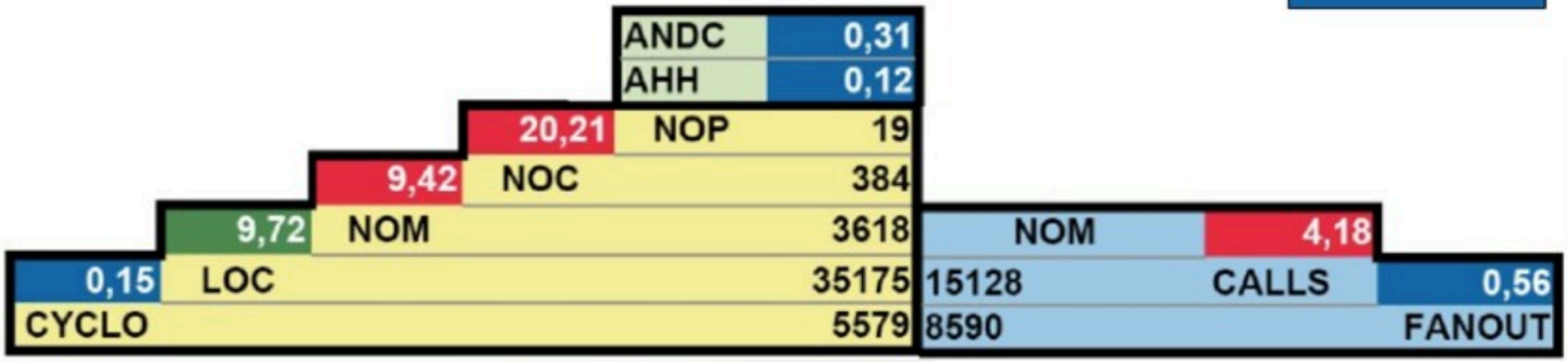
The top: System Inheritance

Direct metrics: ANDC, AHH

High

Average

Low

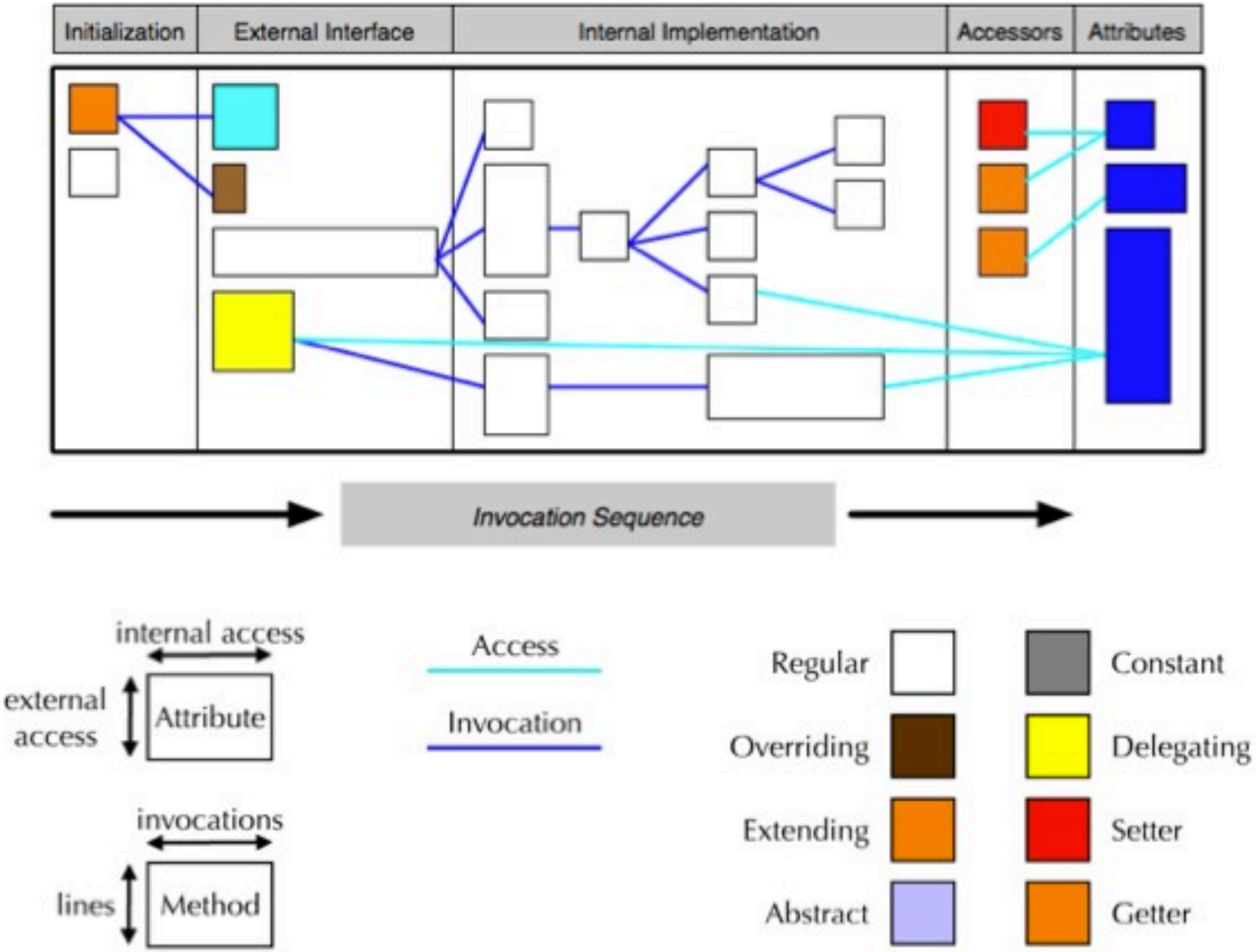


ANDC 0,31
AHH 0,12
20,21 NOP 19
9,42 NOC 384
9,72 NOM 3618 NOM 4,18
0,15 LOC 35175 15128 CALLS 0,56
CYCLO 5579 8590 FANOUT

2h 1.3 & 17.5

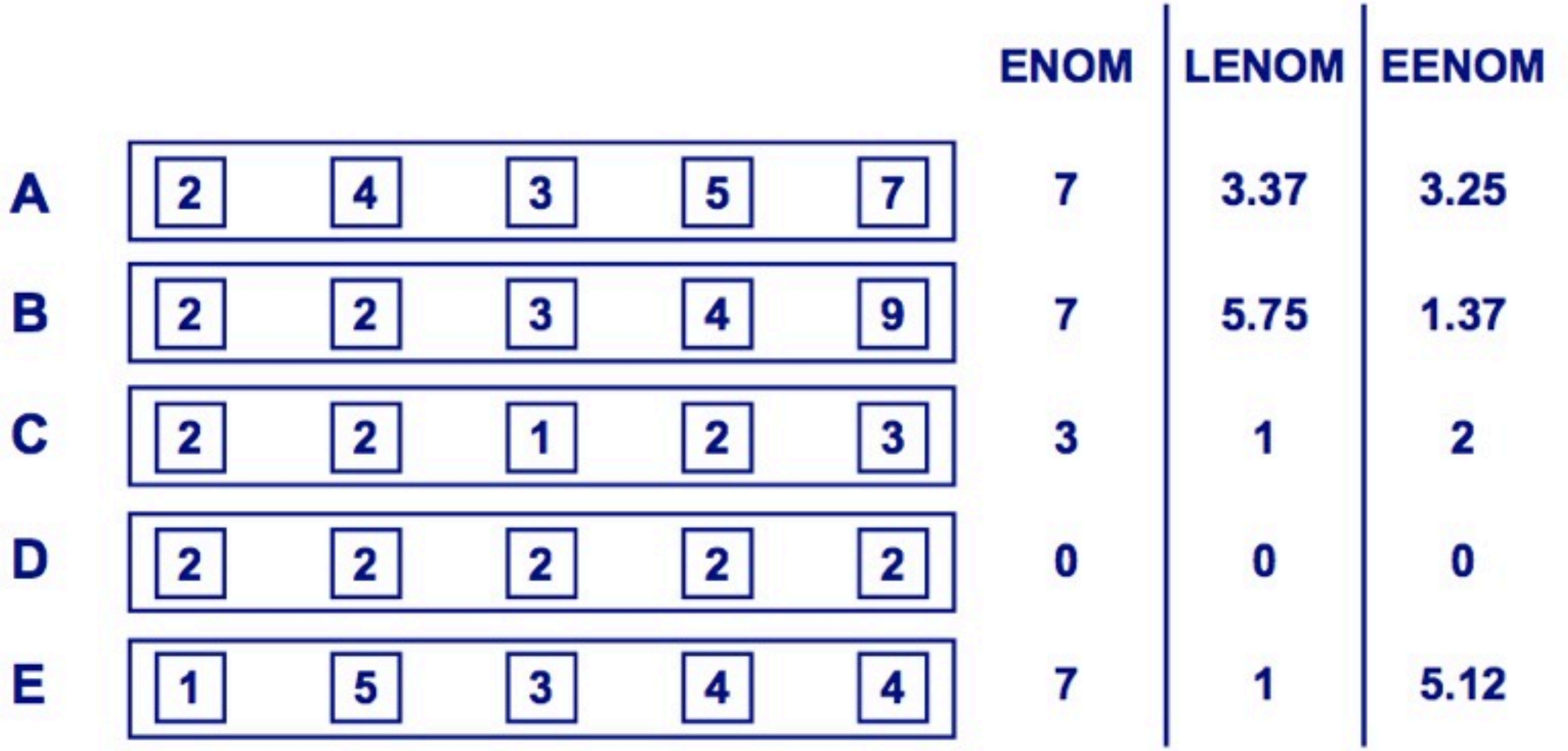
- Identify experts & code owners: Who are the active maintainers fixing defects
- Identify the critical parts of a system
- Get overall impression of the state of the system: How many critical bugs? How long does it take to fix a bug?
- Software business analyst
- Build defect prediction models to forecast the location of defects in the next release

3a, 22.3


Initialization
External Interface
Internal Implementation
Accessors
Attributes
Invocation Sequence
internal access
external access
Attribute
invocations
lines
Method
Access
Invocation
Regular
Overriding
Extending
Abstract
Constant
Delegating
Setter
Getter

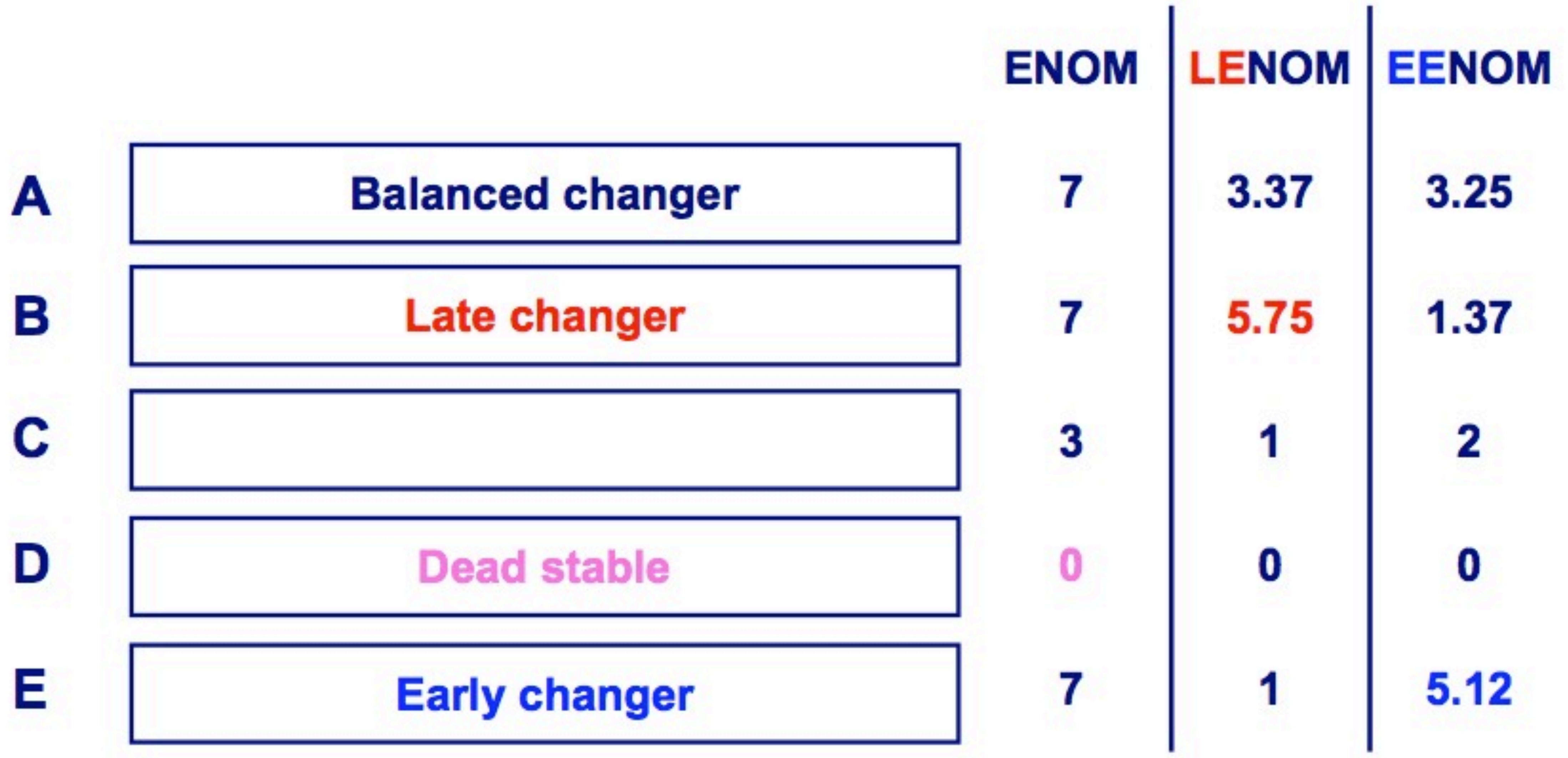
3e, 3.5 Slides 14-18

Evolution of the number of methods (NOM) of classes of a system



						ENOM	LENOM	EENOM
A	2	4	3	5	7	7	3.37	3.25
B	2	2	3	4	9	7	5.75	1.37
C	2	2	1	2	3	3	1	2
D	2	2	2	2	2	0	0	0
E	1	5	3	4	4	7	1	5.12

3e, 3.5 Slides 14-18



		ENOM	LENOM	EENOM
A	Balanced changer	7	3.37	3.25
B	Late changer	7	5.75	1.37
C		3	1	2
D	Dead stable	0	0	0
E	Early changer	7	1	5.12

4a, 10.5

- In particular see mentioned paper:"*Cloning Considered Harmful" Considered Harmful"*
- Pragmatic: If clones do not change, and hence, cause no additional effort (or defects) do not refactor
- Due to language or framework features: Boilerplate code
- Intentionally created clones: Hardware variation, e.g., Linux SCSI Driver -> for each platform code is cloned and modified
- Inconclusive results from empirical studies: Questioning at least the general bad reputation of code clones
- There are tools that support clone management