PV260 COURSE INTRODUCTION

ROADMAP TO SOFTWARE QUALITY

Barbora Bühnová
buhnova@fi.muni.cz

LAB OF SOFTWARE ARCHITECTURES
AND INFORMATION SYSTEMS

FACULTY OF INFORMATICS
MASARYK UNIVERSITY, BRNO

lasaris

Outline of the lecture

- Course introduction
  - Course motivation and goals
  - Course organization
  - Our team

- Roadmap to quality engineering methods
  - Define quality issues
  - Prevent quality issues
  - Detect quality issues
  - Repair quality issues
  - Keep track of quality issues

- Choose well, plan well

lasaris

Outline of the lecture

- Course introduction
  - Course motivation and goals
  - Course organization
  - Our team

- Roadmap to quality engineering methods
  - Define quality issues
  - Prevent quality issues
  - Detect quality issues
  - Repair quality issues
  - Keep track of quality issues

- Choose well, plan well

Course motivation and goals

"People forget **how fast** you did a job – but they remember **how well** you did it" – some guy named Howard Newton

- The **aim of the course** is to help the students to
  - understand activities contributing to building **high-quality software**;
  - develop **critical thinking** and be able to identify **code flaws** related to reliability, performance, scalability, maintainability and testability;
  - be able to **refactor existing code** to improve different quality attributes;
  - have practical experience with software **testing** and related **tools**.

Outline of lectures

Lect 1. Course organization. Roadmap to software quality engineering methods.

Lect 2. Basic Principles of Testing. Requirements and test cases. Test plans and risk analysis.

Lect 3. Automated testing and testability. **Continuous integration/delivery**.

Lect 4. Clean Code, Bad Code Smells and code refactoring.

Lect 5. Software measurement and metrics, and their role in quality improvement.

Lect 6. Focus on **quality attributes and conflicts** between them.

Lect 7. Challenges of quality management in **cloud applications**.

Lect 8. Performance engineering and performance testing.

Lect 9. The role of **software architecture** - practitioner view.

Lect 10. The role of **software architecture II** - practitioner view.

Lect 11. Static code analysis and code reviews.

Lect 12. Software architecture – **case study**.

Lect 13. Software quality management process.

31. 5. 10:00-14:00 [All] Colloquium event

Course organization

- Lectures
  - Shared by us and **experts** from companies
  - May **not be recorded**
  - Final **colloquium event** after the end of semester (**May 31, 2023**)

- Seminars
  - Practical assignments on computers
  - Teamwork, homework, projects
  - **Java** seminar groups – by **NetSuite, Aricoma** experts
  - **Java/multi** seminar group – by **Red Hat** experts
  - **C#** seminar groups – by **Y Soft** experts

Course organization

- Evaluation
  - **45** points for seminar **assignments**
    - All the assignments **need to be at least submitted**, otherwise, the student cannot attend the final colloquium event and write the test.
  - **10 seminar activity** points
  - **10 lecture activity** points
  - **35** points for **final colloquium assessment**, consisting of
    - obligatory **attendance** at the final colloquium event and
    - final **written test**
  - Minimum of **70 points for passing the course**

- Colloquium event
  - On **May 31, 2023, between 10:00-14:00**
  - **Discussion groups** led by industrial experts
  - **Student presentations** of outcomes
  - **Written test**

Outline of the lecture

- Course introduction
  - Course motivation and goals
  - Course organization
  - Our team

- Roadmap to quality engineering methods
  - Define quality issues
  - Prevent quality issues
  - Detect quality issues
  - Repair quality issues
  - Keep track of quality issues

- Choose well, plan well

Quality Engineering (QE) methods

Measurement and metrics
CMMI, ITIL

Functional testing
Performance testing

Coding best practices
Code conventions
Pair programming

Test driven development

Usability testing
Security testing

QA processes
Standards

Design inspections
Code reviews

Design patterns

Static code analysis

SOLID principles
Clean Code

Requirements engineering
Quality attributes

Issue tracking
Configuration management

Fault tolerance mechanisms
Performance tuning

V-model of testing

Technical debt management

Security tactics

Roadmap to QE methods

Prevent quality issues	**Detect** quality issues
Repair quality issues	**Keep track of** quality issues

Define
quality issues

lasaris
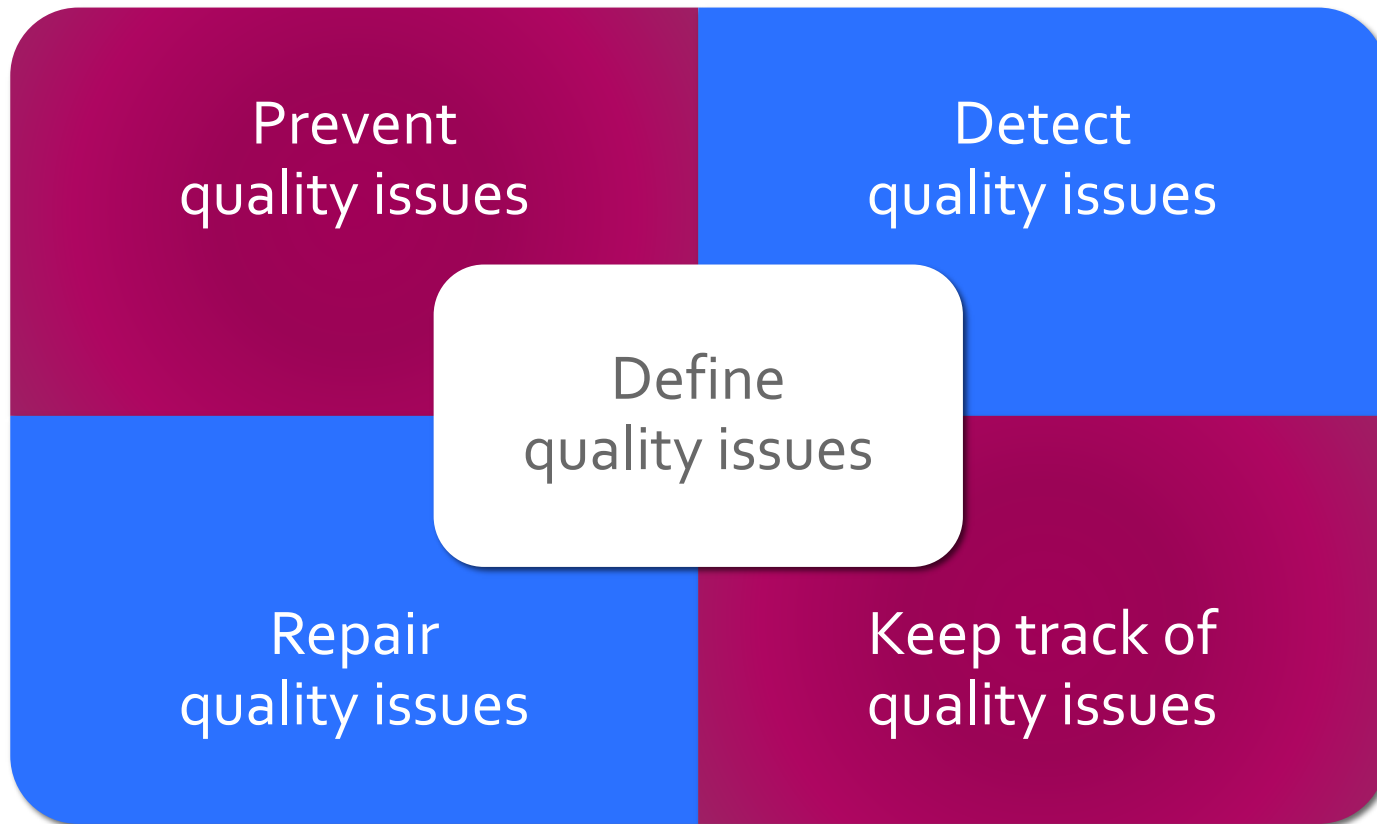
Define quality issues

- Software quality is commonly defined as the capability of a software product to conform to requirements [ISO/IEC 9001].

  customer needs
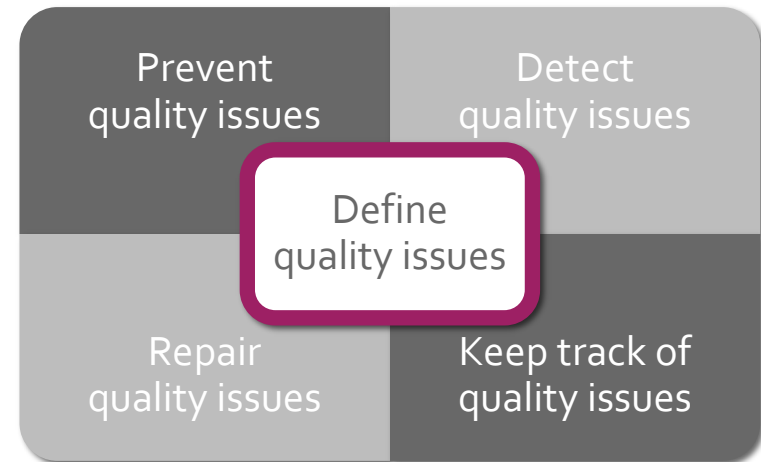
Prevent quality issues	Detect quality issues
Repair quality issues	Keep track of quality issues

Define quality issues

- Requirements engineering

- Software metrics
  - 'You cannot manage what you cannot measure'

- Quality attributes
  - Of a **product**, process and resources

lasaris

What "quality" means to you?

… and your customer?

… and your manager?

Stakeholders view | **Quality goals**

visible
it works

it looks good inside

invisible

it will work also next year

User Experience (customer)
- Usability
- Accuracy
- Reliability
- Performance
- Security

Feature

Code Quality (developer)
- Modularity
- Complexity
- Resilience
- Understandability
- Testability

Engineering

Long-term View (manager)
- Adaptability
- Portability
- Reusability
- Maintainability
- Scalability

Cost effectiveness

lasaris

The Software Quality Iceberg

Visible / Symptoms

testing

manual or automated?

reliability
cost
usability
security
accuracy
performance

EXTERNAL QUALITY

Invisible / Root

INTERNAL QUALITY

coding practices

program structure

complexity

testability

flexibility

reusability

maintainability

understandability

inspection

measurable or not?
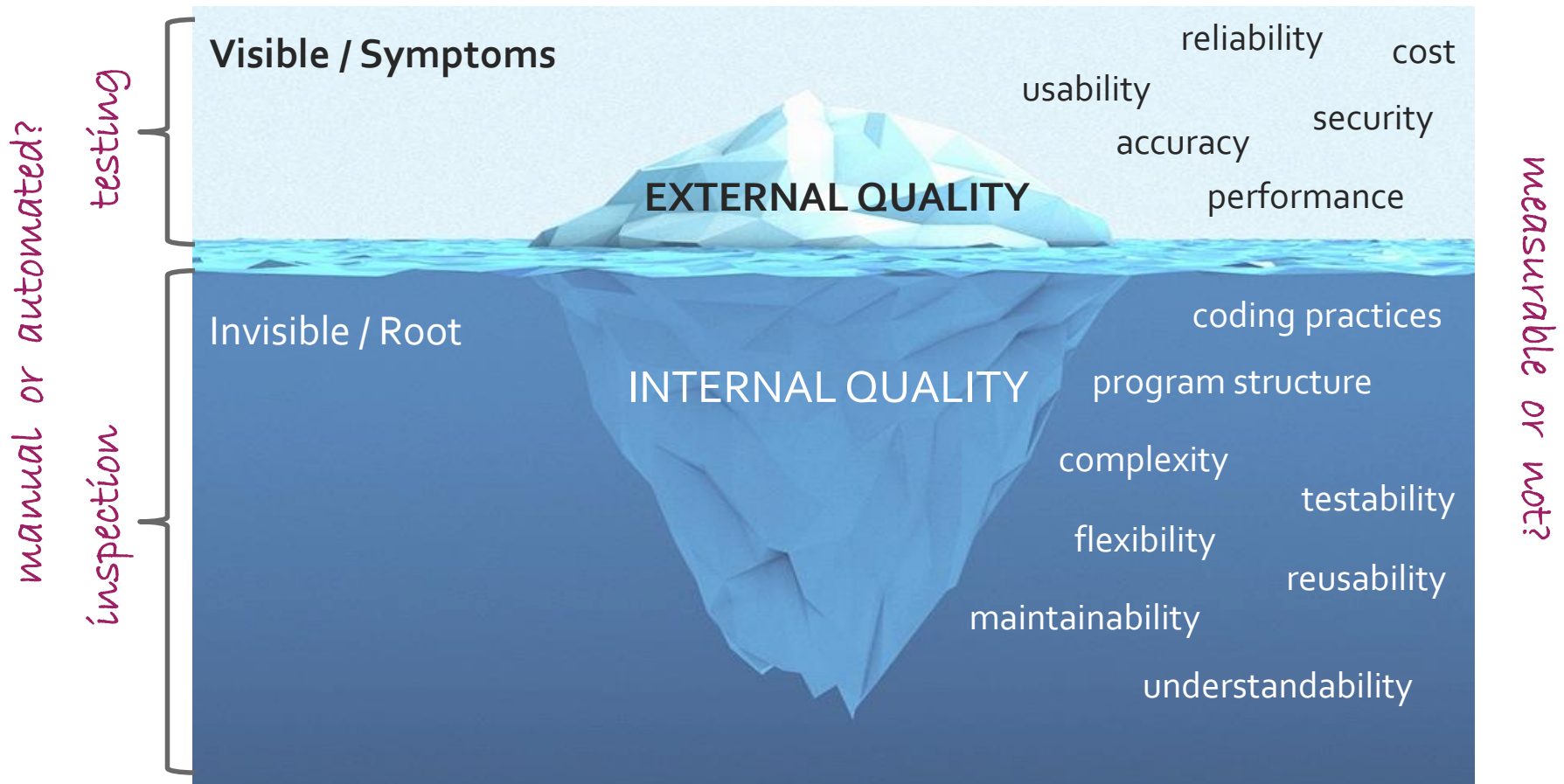
Inspiration from [5]

lasaris

The big five

- Along the course we will focus on:

  - **Maintainability** – **ease of change** (without increased technical debt)
  - **Performance** – **response time** and efficiency in resource utilization
  - **Reliability** – probability of **failure-free operation** over a period of time
  - **Testability** – degree to which the system **facilitates testing**
  - **Scalability** – system's ability to handle **growing work load**

- Quality attributes studied in related courses:

  - **Security** – system's ability to **protect itself** from attacks
  - **Usability** – ease of system **use and learnability**

lasaris

Prevent quality issues

code-level

- **Coding best practices**
  - Clean code, SOLID principles
  - Design patterns
  - Pair programming

- **Code conventions**
  - Language specif. recommendations

processes

- **Quality assurance processes**
  - V-model of testing, Test Driven Development

- **Standards for development process improvement**
  - CMMI and ITIL reference models
  - ISO 9000, ISO/IEC 25010

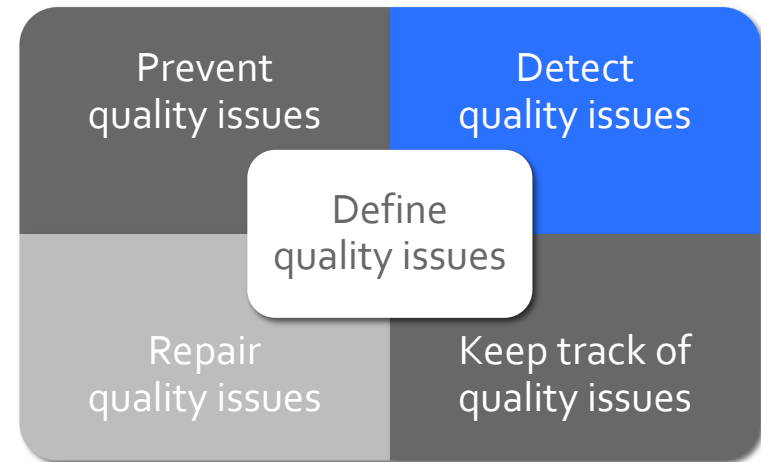| Prevent quality issues	Detect quality issues
Define quality issues	
Repair quality issues	Keep track of quality issues

lasaris
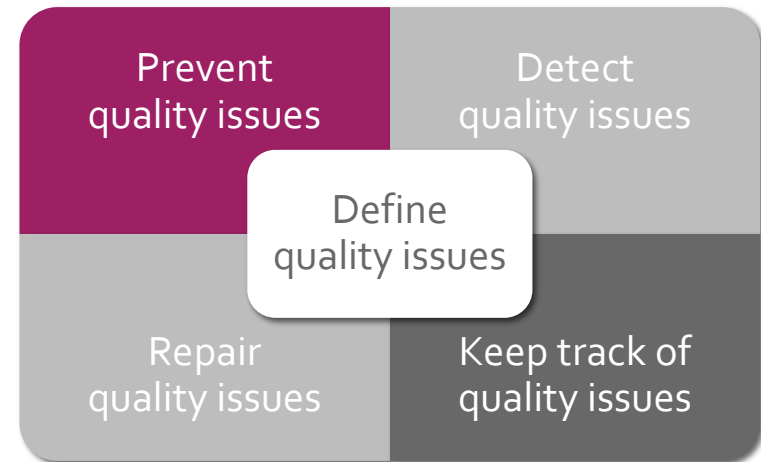
Detect quality issues

testing

- **Testing functional requirements**
  - Manual or automated

- **Testing non-functional req.**
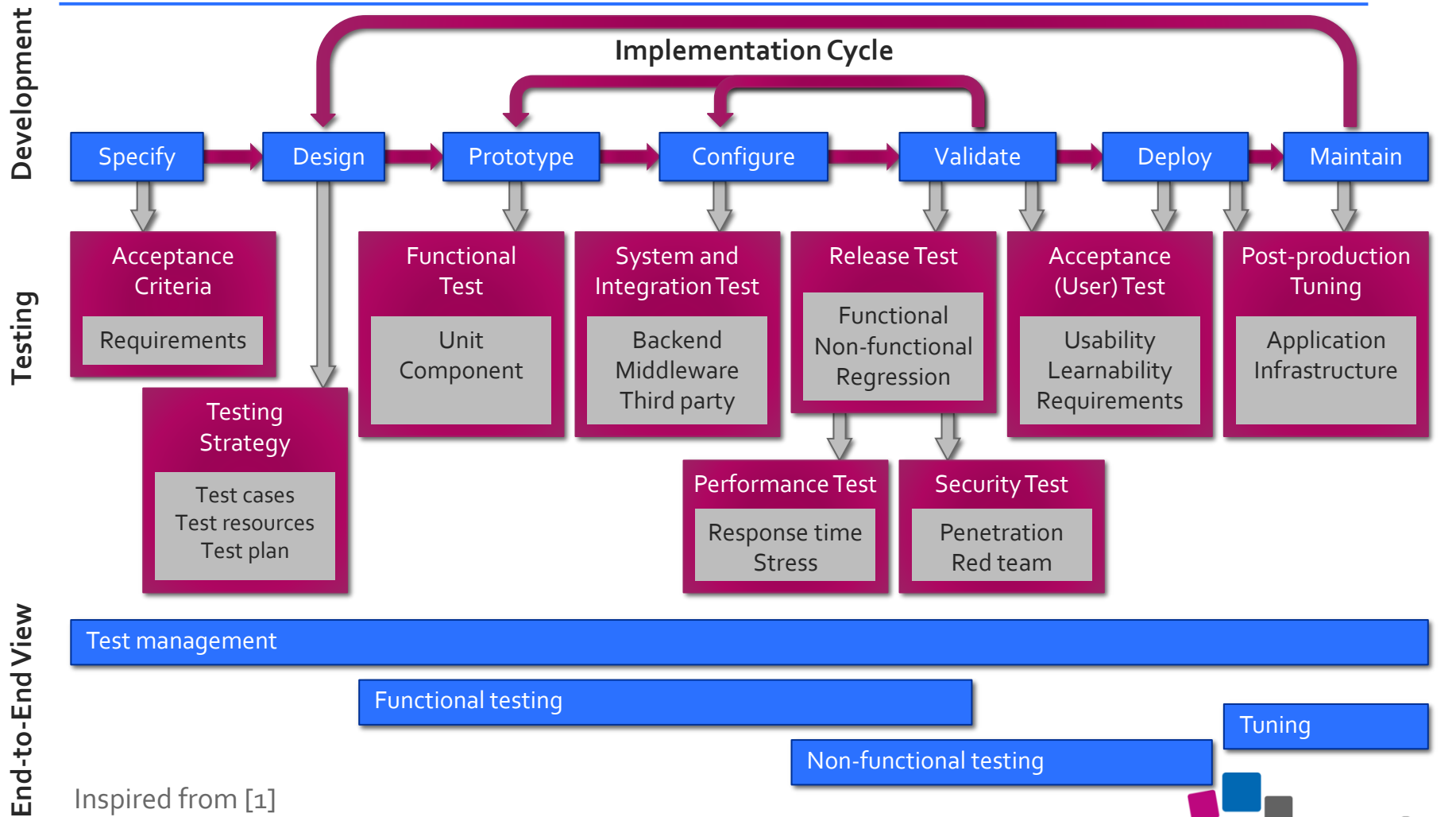 - Performance, usability, security testing

static analysis

- **Design inspections**
  - Manual inspections of design artifacts

- **Code reviews**
  - Manual inspections of code

- **Automated static code analysis**

Prevent quality issues	Detect quality issues
Repair quality issues	Keep track of quality issues

Define quality issues

lasaris

Roadmap to software testing

Development

Implementation Cycle

| Specify | → | Design | → | Prototype | → | Configure | → | Validate | → | Deploy | → | Maintain |

Testing

Acceptance Criteria
Requirements

Functional Test
Unit
Component

System and Integration Test
Backend
Middleware
Third party

Release Test
Functional
Non-functional
Regression

Acceptance (User) Test
Usability
Learnability
Requirements

Post-production Tuning
Application
Infrastructure

Testing Strategy
Test cases
Test resources
Test plan

Performance Test
Response time
Stress

Security Test
Penetration
Red team

End-to-End View

Test management

Functional testing

Non-functional testing

Tuning

Inspired from [1]

lasaris

B. Bühnová, PV260 Software Quality

Repair quality issues

- ## Functional issue
  - Code repair

- ## Reliability issue
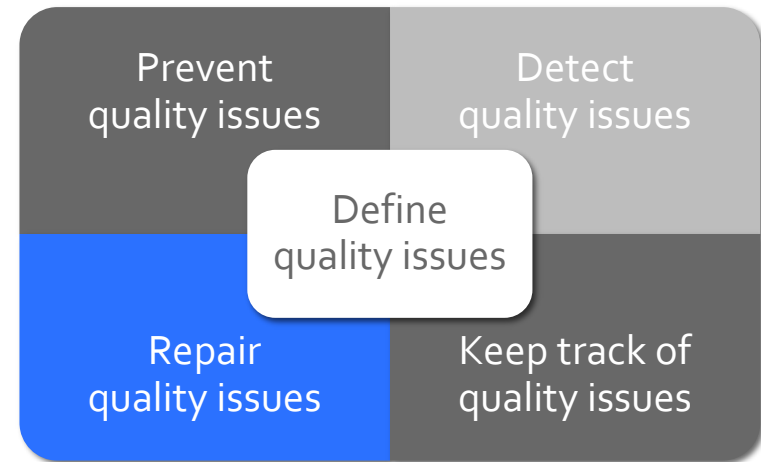 - Fault tolerance mechanisms

- ## Performance issue
  - Concurrency, effective resource utilization, identify and remove system bottlenecks

- ## Security issue
  - Identify and remove system vulnerabilities (single points of failure)

- ## Maintainability issue
  - Refactoring to clean code principles, to design patterns

tactics and patterns

Prevent quality issues	Detect quality issues
Repair quality issues	Keep track of quality issues

Define quality issues

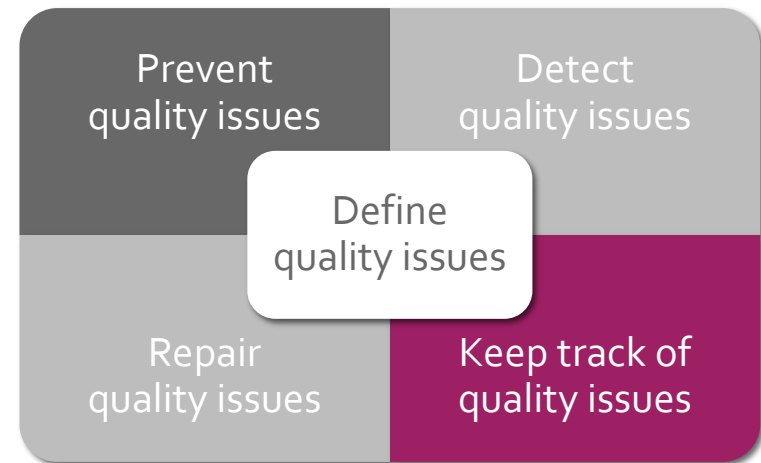lasaris

Keep track of quality issues

- Issue tracking
  - Supports the management of issues reported by customers

- Technical debt management
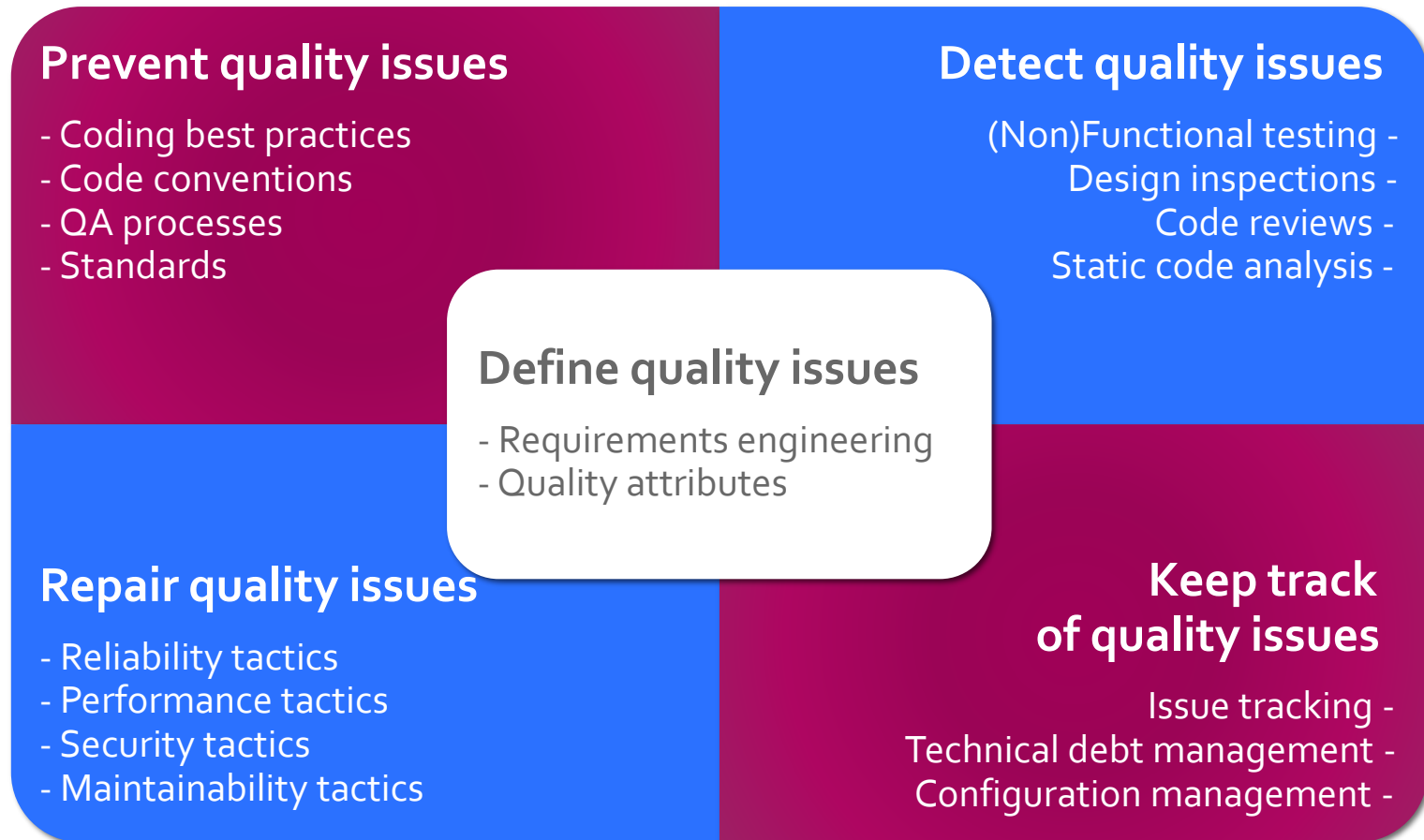 - Level of code quality degradation
  - Work that needs to be done before a particular job can be considered complete or proper

- Configuration management
  - Version management and release management
  - System integration

Prevent quality issues

Detect quality issues

Define quality issues

Repair quality issues

Keep track of quality issues

lasaris

Roadmap to QE methods – the Big Picture

Prevent quality issues

- Coding best practices
- Code conventions
- QA processes
- Standards

Detect quality issues

(Non)Functional testing -
Design inspections -
Code reviews -
Static code analysis -

Define quality issues

- Requirements engineering
- Quality attributes

Repair quality issues

- Reliability tactics
- Performance tactics
- Security tactics
- Maintainability tactics

Keep track of quality issues

Issue tracking -
Technical debt management -
Configuration management -

lasaris

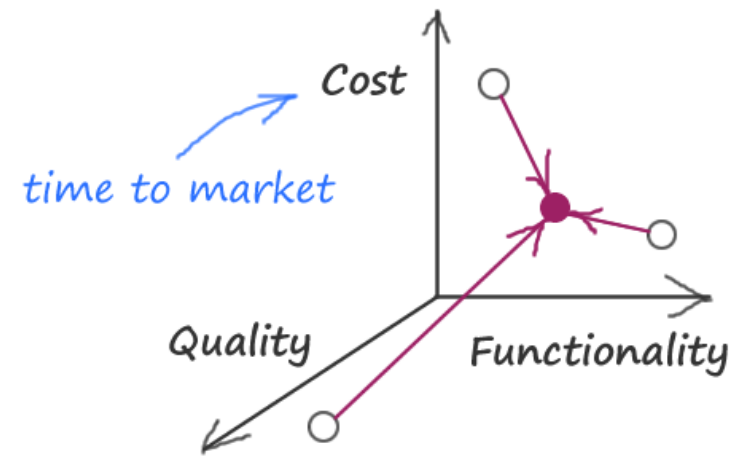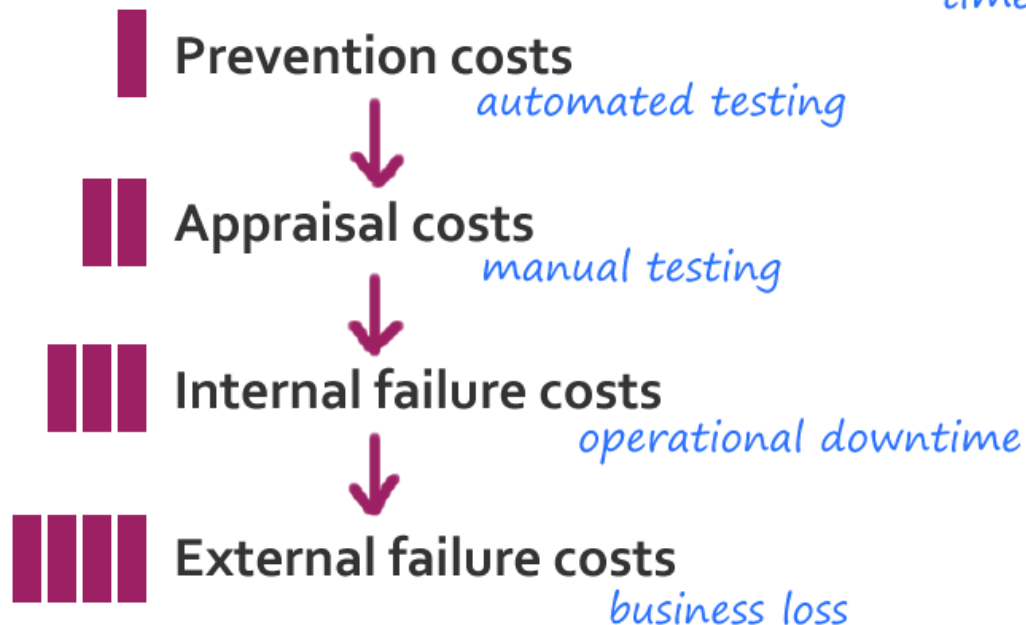B. Bühnová, PV260 Software Quality

Outline of the lecture

- Course introduction
  - Course motivation and goals
  - Course organization
  - Our team

- Roadmap to quality engineering methods
  - Define quality issues
  - Prevent quality issues
  - Detect quality issues
  - Repair quality issues
  - Keep track of quality issues
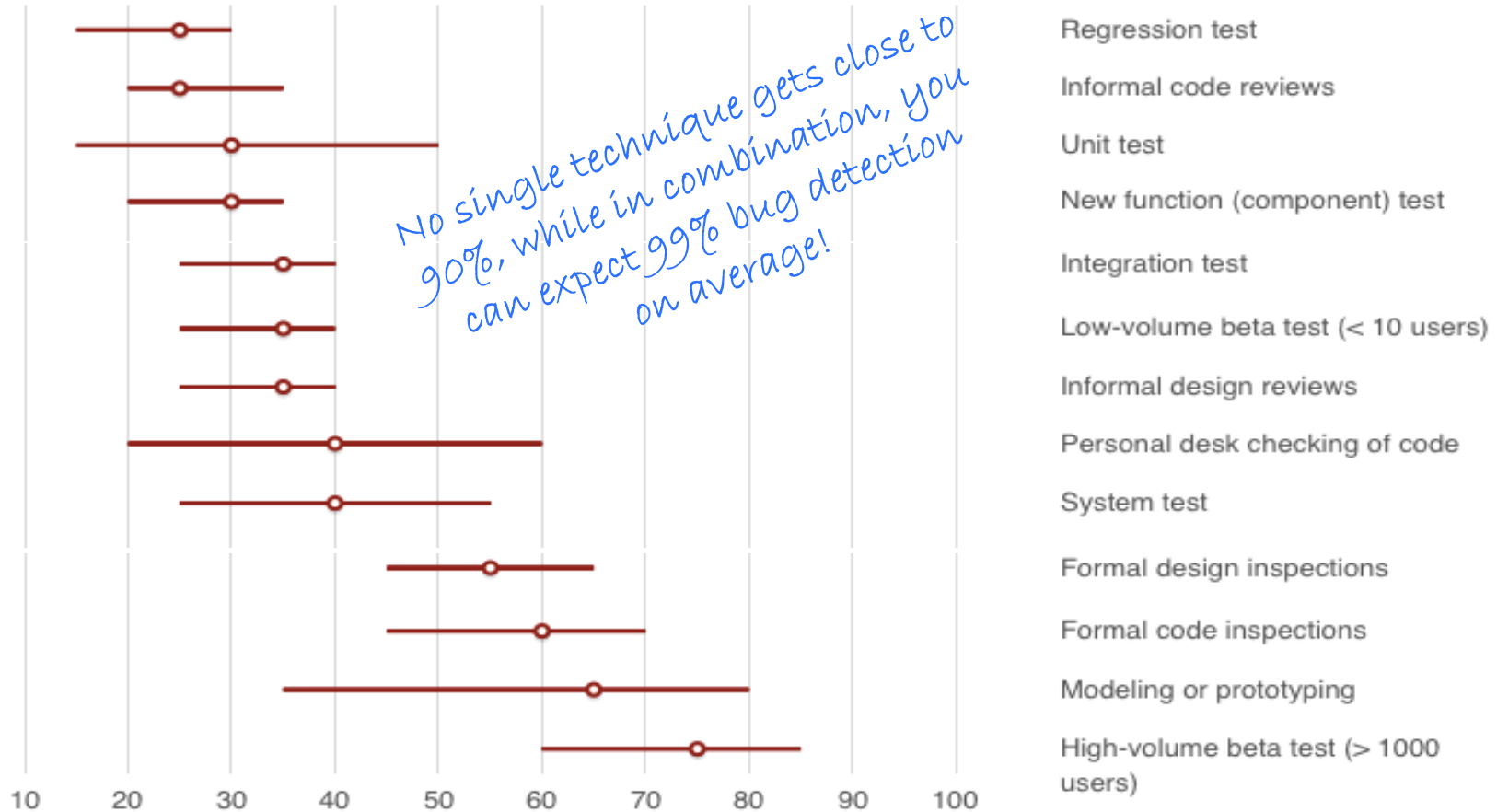
- Choose well, plan well

Choose well, plan well

- Think well about your **requirements** and the **cost of the quality**

Prevention costs
automated testing

↓

Appraisal costs
manual testing

↓

Internal failure costs
operational downtime

↓

External failure costs
business loss

Cost

time to market

Quality

Functionality

lasaris

Choose well – Combination is the key

Regression test

Informal code reviews

Unit test

New function (component) test

Integration test

Low-volume beta test (< 10 users)

Informal design reviews

Personal desk checking of code

System test

Formal design inspections

Formal code inspections

Modeling or prototyping

High-volume beta test (> 1000 users)

No single technique gets close to 90%, while in combination, you can expect 99% bug detection on average!

From [2,3], see also RebelLabs reports [4]

lasaris

B. Bühnová, PV260 Software Quality

# Plan well – The Power of Analogy

- **Airplane Servicing**
  - Requires **regular servicing** e.g. every 100,000 miles.
  - Takes place even if everything seems to work all right, because we cannot afford a failure.

Can we quantify it?

- **Technical Debt Management**
  - Introduced by Ward Cunningham
  - Analogy of **quality degradation with financial debt** – if not paid off, interests increase. One can get into trouble.
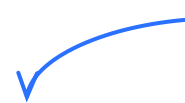
- **Sometimes it is wise to "borrow money"**
  - When one expects to have more money in the future (start-up company)
  - When one needs to act fast not to miss a market opportunity
  - When one expects money devaluation (e.g. developers will become more experienced, it will be easier to understand user needs)

lasaris

Takeaways

- **Quality engineering (QE)** is much more than **testing**, including many different methods to
  - **prevent**, **detect**, **repair** and **keep track** of quality issues

- **Combination** **of the methods** **is the key** to successful QE
  - But **choose well** and **plan well**, not all methods are best for your project!

- Make sure you understand the **needs of your customer**
  - Balance **both internal and external quality attributes** for both the present and the future

thanks for listening

Barbora Bühnová, FI MU Brno
buhnova@fi.muni.cz
www.fi.muni.cz/~buhnova

contact me

lasaris

References

- [1] Testing You Perform When You Develop a Siebel Application. Available online at http://docs.oracle.com/cd/E14004_01/books/DevDep/Overview5.html

- [2] Steve McConnell. Code Complete: A Practical Handbook of Software Construction, Second Edition. Microsoft Press, June 2004.

- [3] Kevin Burke. Why code review beats testing: evidence from decades of programming research. Available online at https://kev.inburke.com/kevin/the-best-ways-to-find-bugs-in-your-code/

- [4] RebelLabs. 2013 Developer Productivity Report. Available online at http://zeroturnaround.com/rebellabs/developer-productivity-report-2013-how-engineering-tools-practices-impact-software-quality-delivery/

- [5] Jonathan Bloom. Titanic Dilemma: The Seen Versus the Unseen. Available online at http://blog.castsoftware.com/titanic-dilemma-the-seen-versus-the-unseen/

lasaris