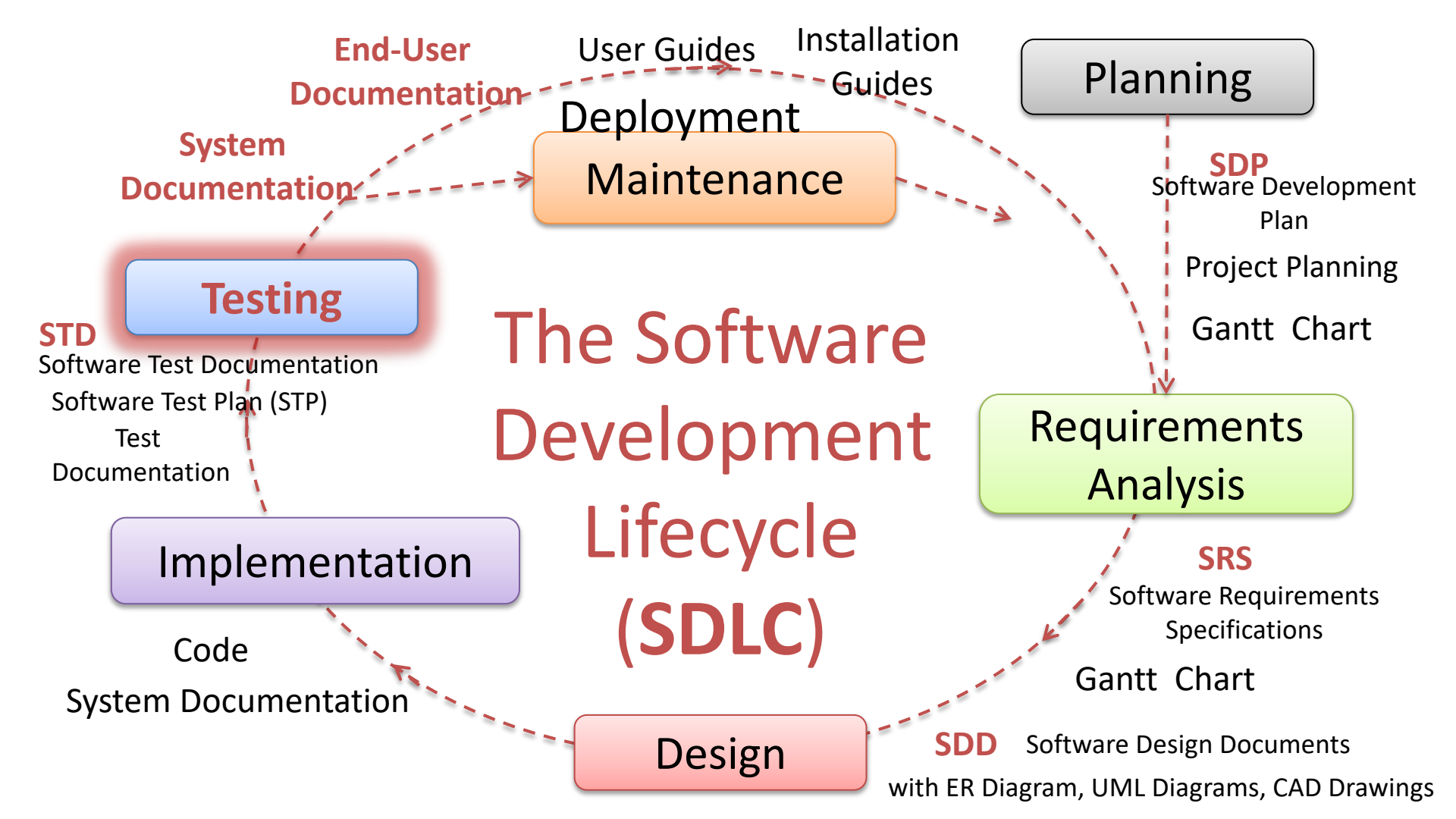
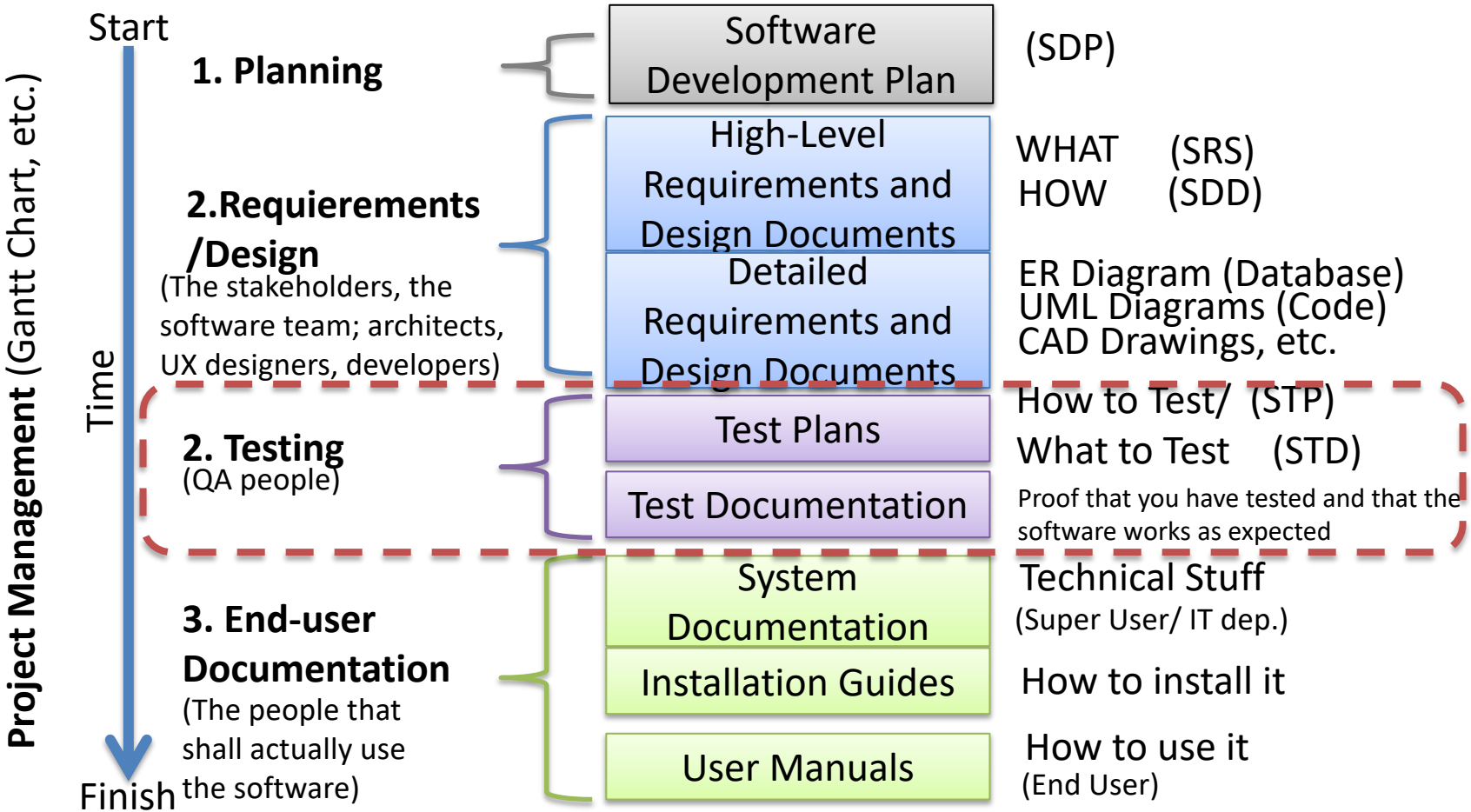
Software Testing

Hans-Petter Halvorsen, M.Sc.


End-User Documentation
User Guides
Installation Guides
Planning
Deployment
System Documentation
Maintenance
SDP
Software Development Plan
Project Planning
Gantt Chart
Testing
STD
Software Test Documentation
Software Test Plan (STP)
Test Documentation
The Software Development Lifecycle (SDLC)
Requirements Analysis
SRS
Software Requirements Specifications
Gantt Chart
Implementation
Code
System Documentation
Design
SDD
Software Design Documents
with ER Diagram, UML Diagrams, CAD Drawings

Typical Software Documentation

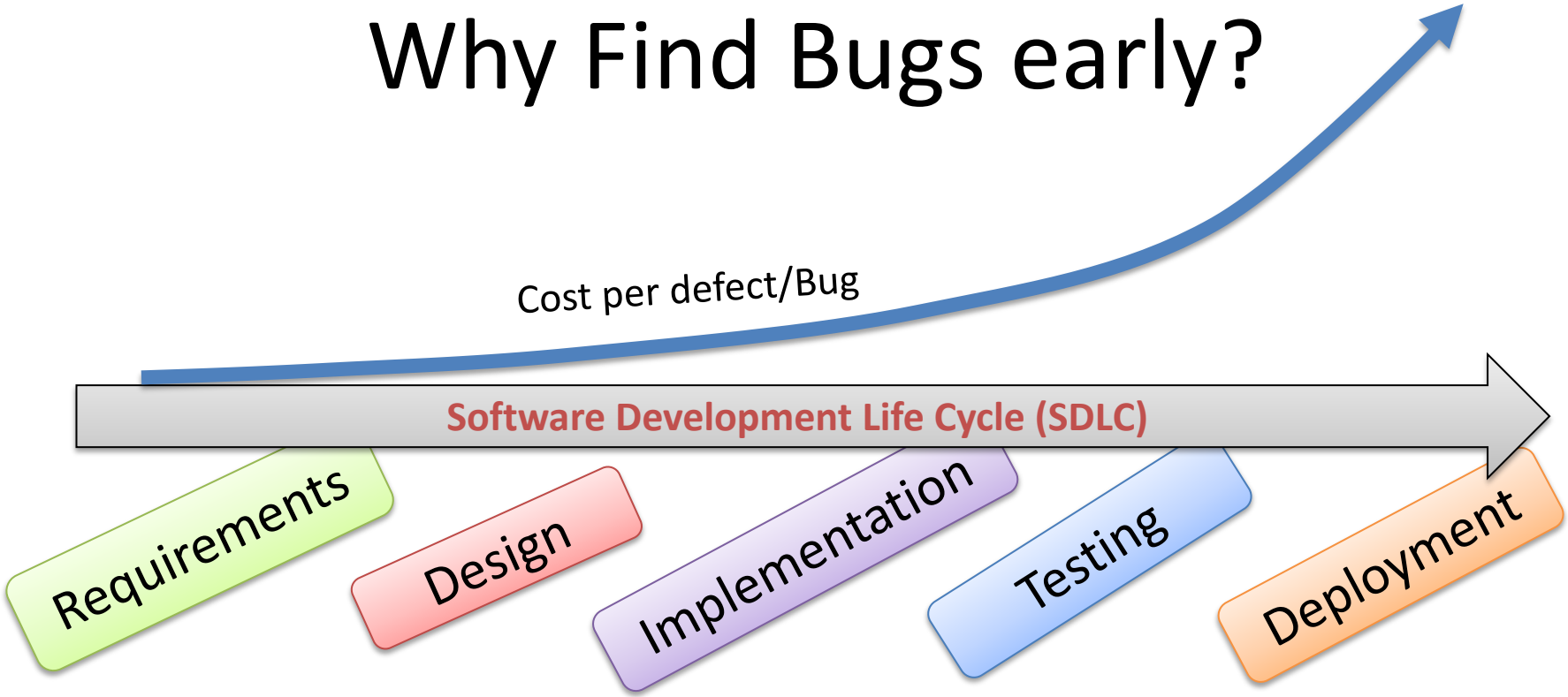


Main purpose of Testing: Find Bugs!!

- Requirements Errors: 13%
- Design Errors: 24%
- Code Errors: 38%
- Documentation Errors: 13%
- Bad-fix Errors: 12%

http://proquest.safaribooksonline.com/book/software-engineering-and-development/9781449691998/chapter-3-engineering-of-software/42?uicode=telemark

Why Find Bugs early?



The First Bug ever

They found a bug (actually a moth) inside a computer in 1947 that made the program not behaving as expected. This was the “first” real bug.

What is Bugs?

- A software bug is an error, flaw, failure, or fault in a computer program or system that produces an incorrect or unexpected result, or causes it to behave in unintended ways
- They found a bug (actually a moth) inside a computer in 1947 that made the program not behaving as expected. This was the “first” real bug.
- Debugging: Find and Remove/Fix Bugs

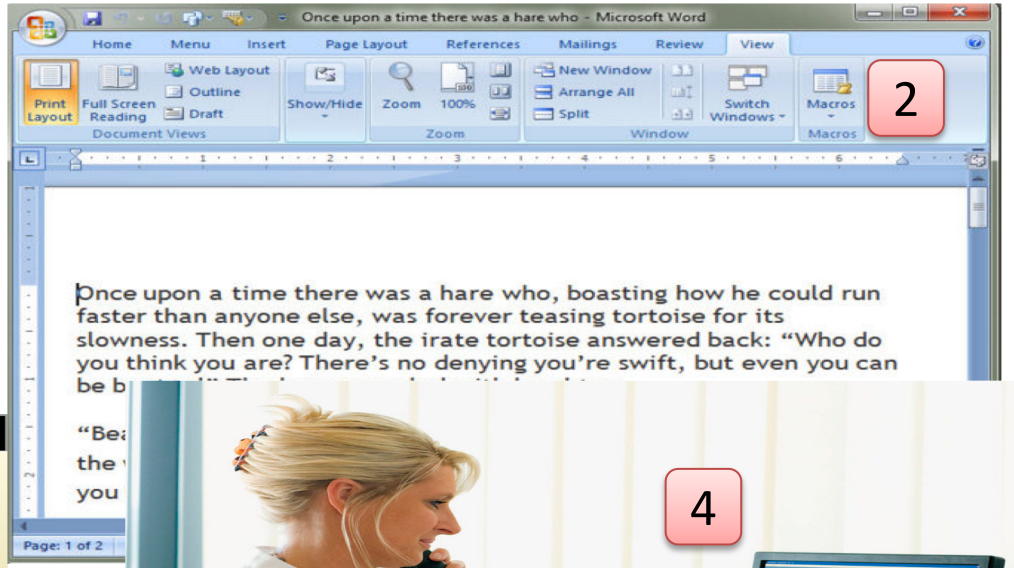
Software Testing

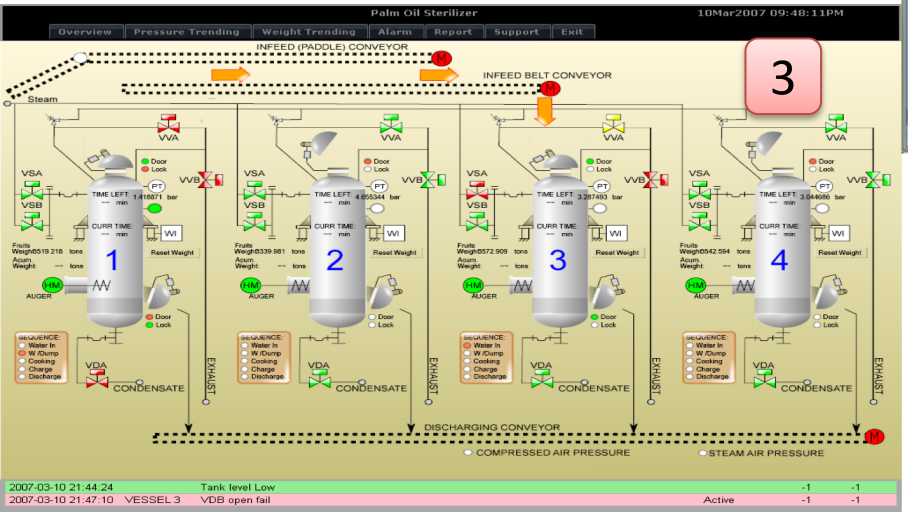
“If you don’t know how your code works, it does not work

– you just don’t know it yet”

“50% of the software development is about testing your software”

Different Systems Needs Different Testing





7 Principles of Testing

1. **Testing shows the presence of Bugs**: Software Testing reduces the probability of undiscovered defects remaining in the software but even if no defects are found, it is not a proof of correctness.
2. **Exhaustive Testing is impossible**: Testing everything is impossible! Instead we need optimal amount of testing based on the risk assessment of the application.
3. **Early Testing**: Testing should start as early as possible in the Software Development Life Cycle (SDLC)
4. **Defect Clustering**: A small number of modules contain most of the defects/bugs detected.
5. **The Pesticide Paradox**: If the same tests are repeated over and over again, eventually the same test cases will no longer find new bugs
6. **Testing is Context dependent**: This means that the way you test a e-commerce site will be different from the way you test a commercial off the shelf application
7. **Absence of Error is a Fallacy**: Finding and fixing defects does not help if the system build is unusable and does not fulfill the users needs & requirements

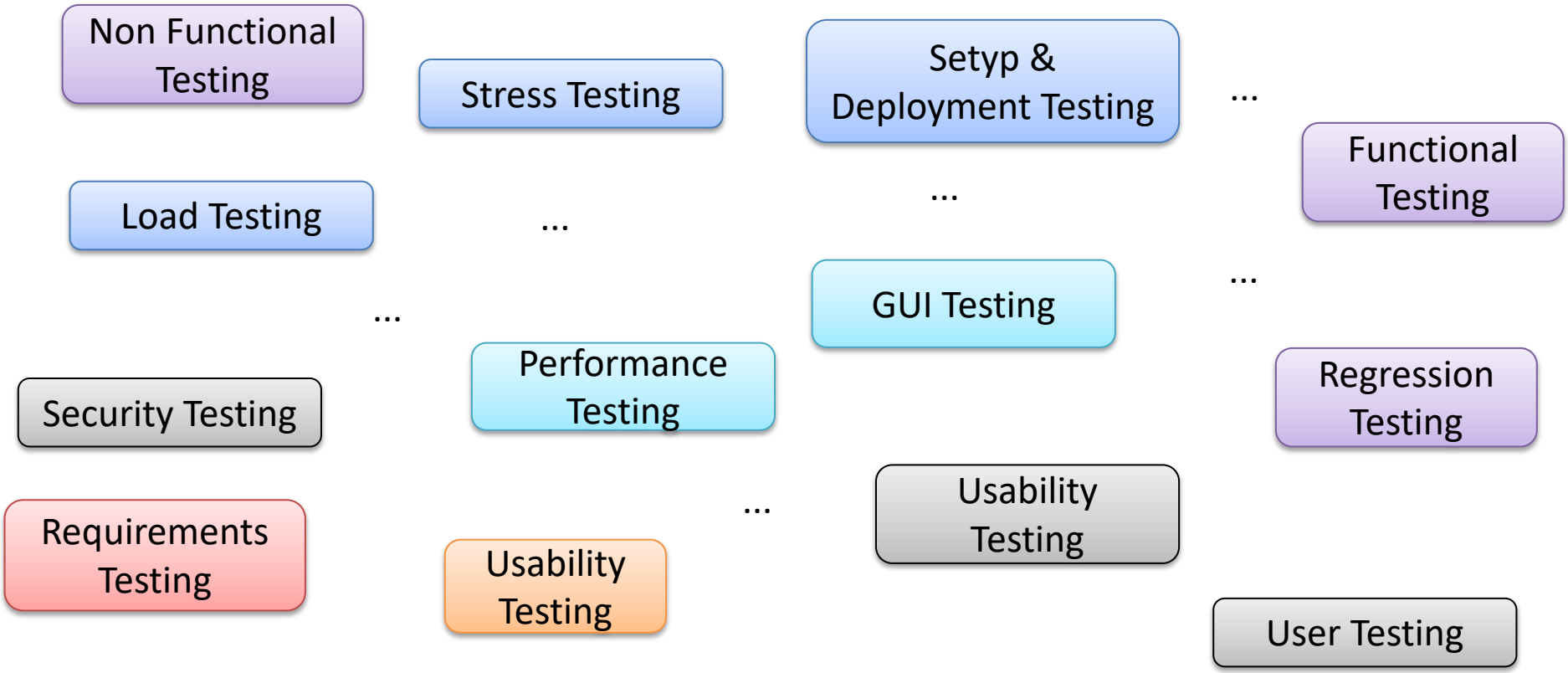
http://www.guru99.com/software-testing-seven-principles.html

http://www.testingexcellence.com/seven-principles-of-software-testing

Different Types of Testing

Hans-Petter Halvorsen, M.Sc.

Types of Testing



Who does the Testing?

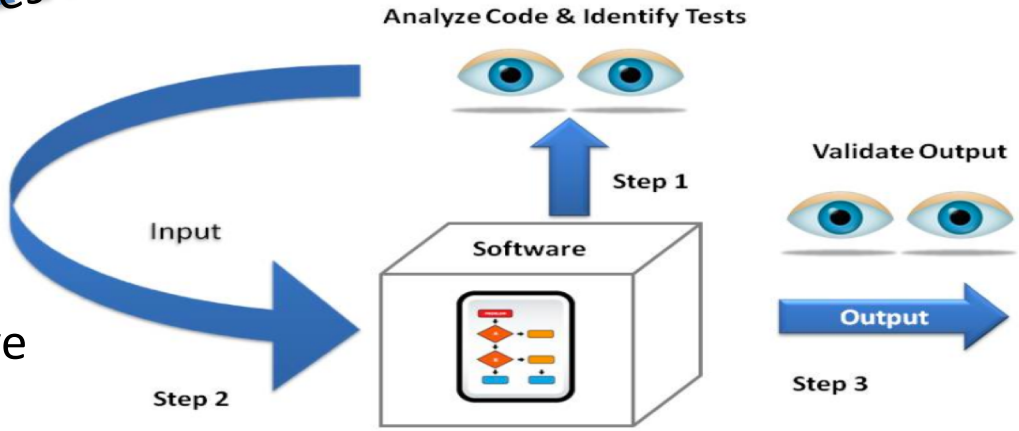
- Programmers/Developers
  - Programmers usually create test cases and run them as they write the code to convince themselves that the program works. This programmer activity related to testing is usually considered to be unit testing.
- Testers
  - A tester is a technical person whose role for the particular item being tested is just to write test cases and ensure their execution. Although programming knowledge is extremely useful for testers, testing is a different activity with different intellectual requirements. Not all good programmers will be good testers.
- End Users/Customers
  - It is a good idea to involve users in testing, in order to detect usability problems and to expose the software to a broad range of inputs in real-world scenarios.

Test Categories

Black-box vs. White-box Testing

Black-box Testing: You need no knowledge of how the system is created.

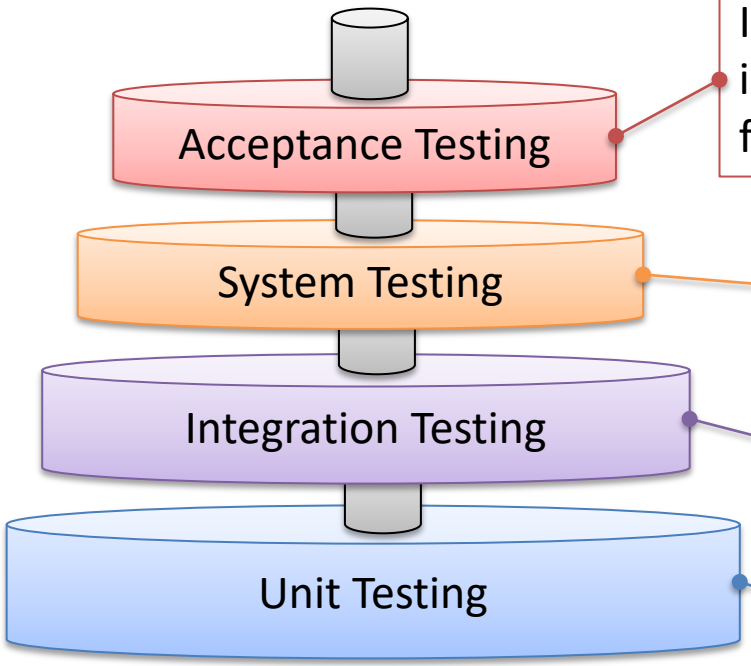
Grey-box Testing



White-box Testing: You need to have knowledge of how (Design and Implementation) the system is built

Typically done by Developers, etc

Levels of Testing



Is the responsibility of the customer – in general. The goal is to gain confidence in the system; especially in its non-functional characteristics

The behavior of the whole product(system) as defined by the scope of the project

Interface between components; interactions with other systems (OS, HW, etc)

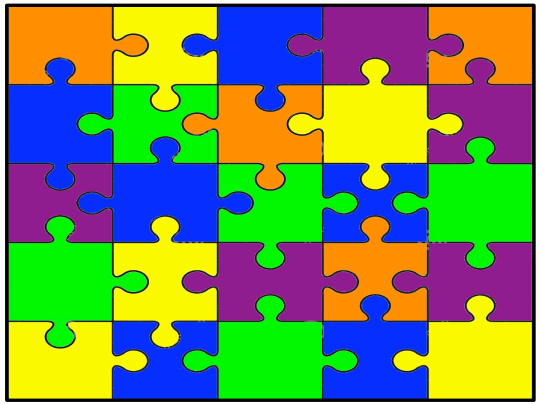
Any module, program, object separately testable

Levels of Testing

Unit Testing: Test each parts independently and isolated

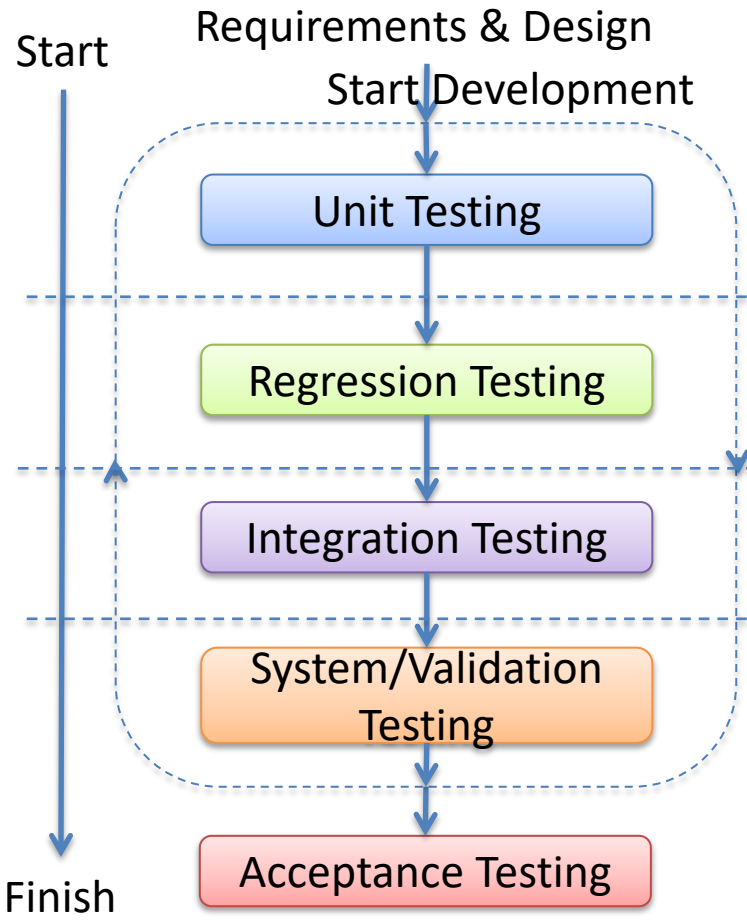
Integration Testing: Make sure that different pieces work together. Test the Interfaces between the different pieces. Interaction with other systems (Hardware, OS, etc.)

Regression Testing: Test that it still works after a change in the code



System Testing: Test the whole system

Levels of Testing



Unit Tests are written by the Developers as part of the Programming. Each part is developed and Unit tested separately (Every Class and Method in the code)

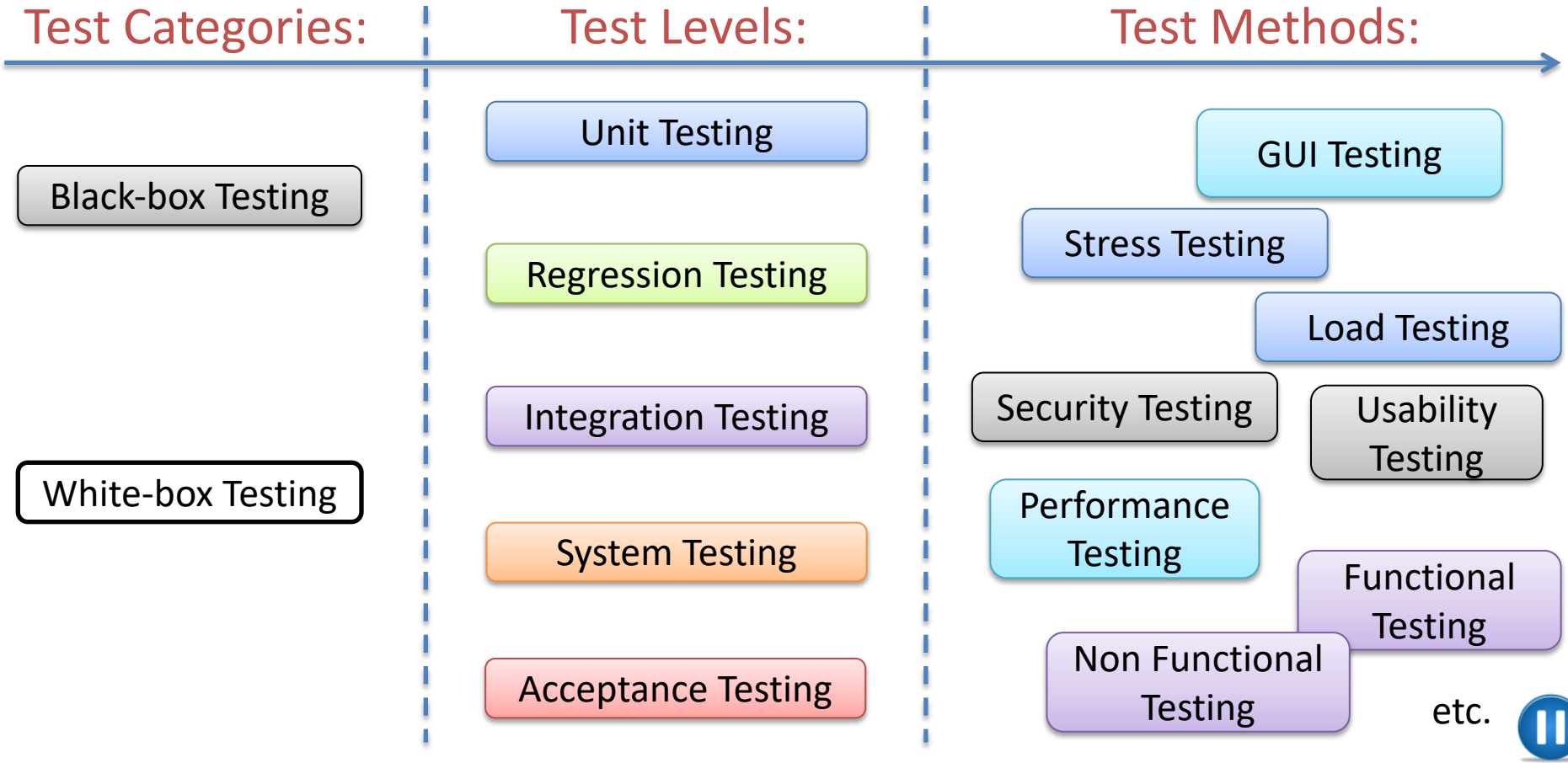
Regression testing is testing the system to check that changes have not "broken" previously working code. Both Manually & Automatically (Re-run Unit Tests)

Integration testing means the system is put together and tested to make sure everything works together.

System testing is typically Black-box Tests that validate the entire system against its requirements, i.e Checking that a software system meets the specifications

The Customer needs to test and approve the software before he can take it into use. FAT/SAT.

Testing Overview



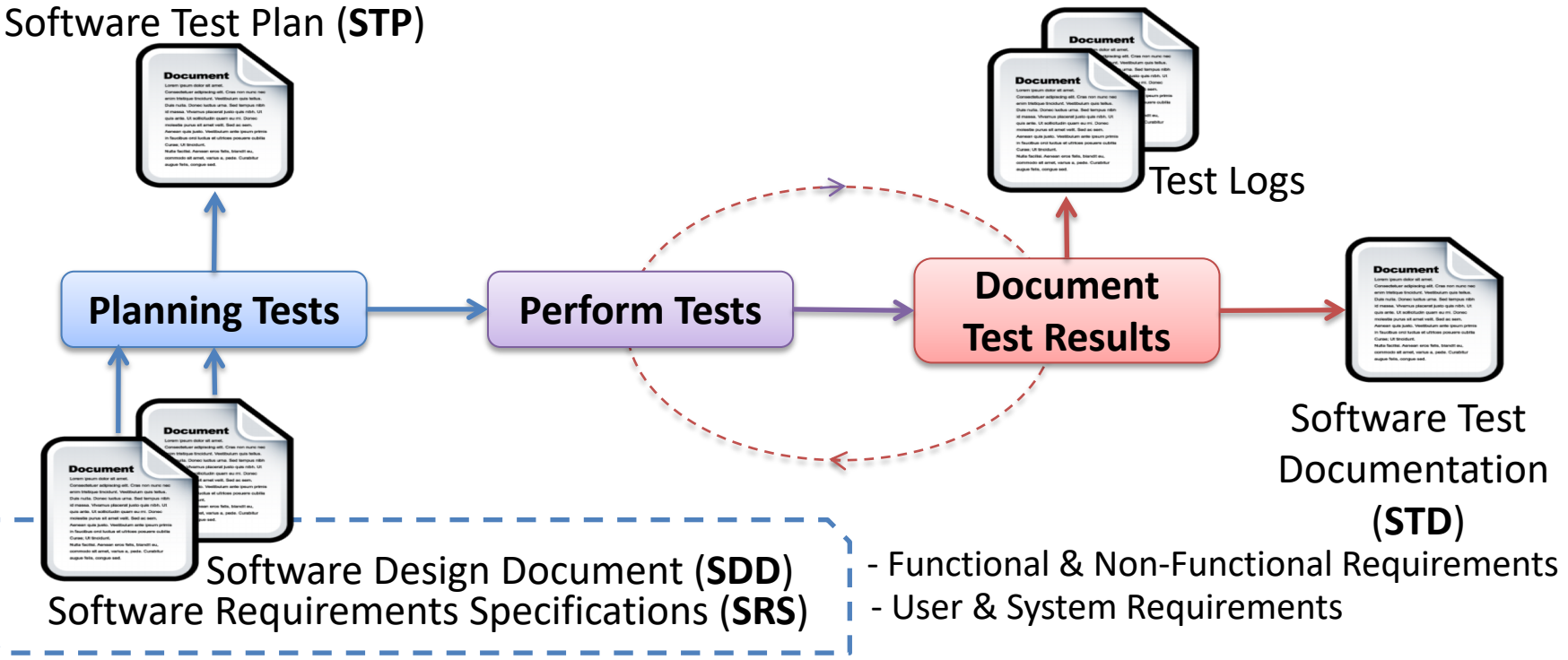
Software Test Plan (STP)

Hans-Petter Halvorsen, M.Sc.

Test Planning

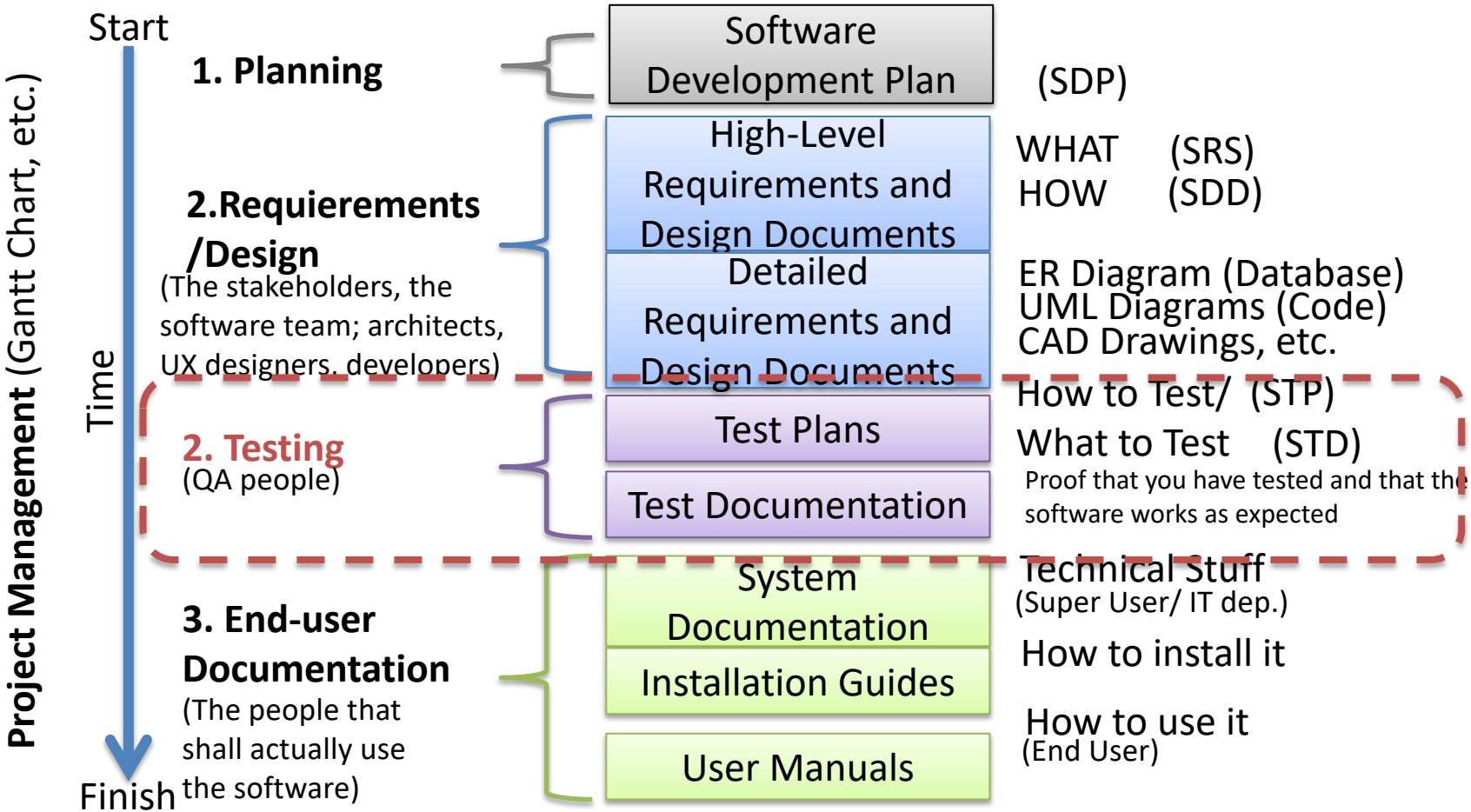
- To maximize the effectiveness of resources spent on testing, a systematic approach is required
- A Software Test Plan (STP) should be created

Test Documentation



These documents will be the foundation for all Testing

Typical Software Documentation



What is a Software Test Plan (STP)?

A Document that answers the following:

- Testing should be based on Requirements & Design Documents
- What shall we test?
- How shall we test?
- Hardware/Software Requirements
- Where shall we test?
- Who shall test?
- How often shall we test (Test Schedule)?
- How shall tests be documented?
  - It is not enough simply to run tests; the results of the tests must be systematically recorded. It must be possible to audit the testing process to check that it has been carried out correctly
- System tests: This section, which may be completely separate from the test plan, defines the test cases that should be applied to the system. These tests are derived from the system requirements specification. http://www.softwareengineering-9.com/Web/Testing/Planning.html

These things need to be specified in the STP

Test specs and Requirement document

Human resources

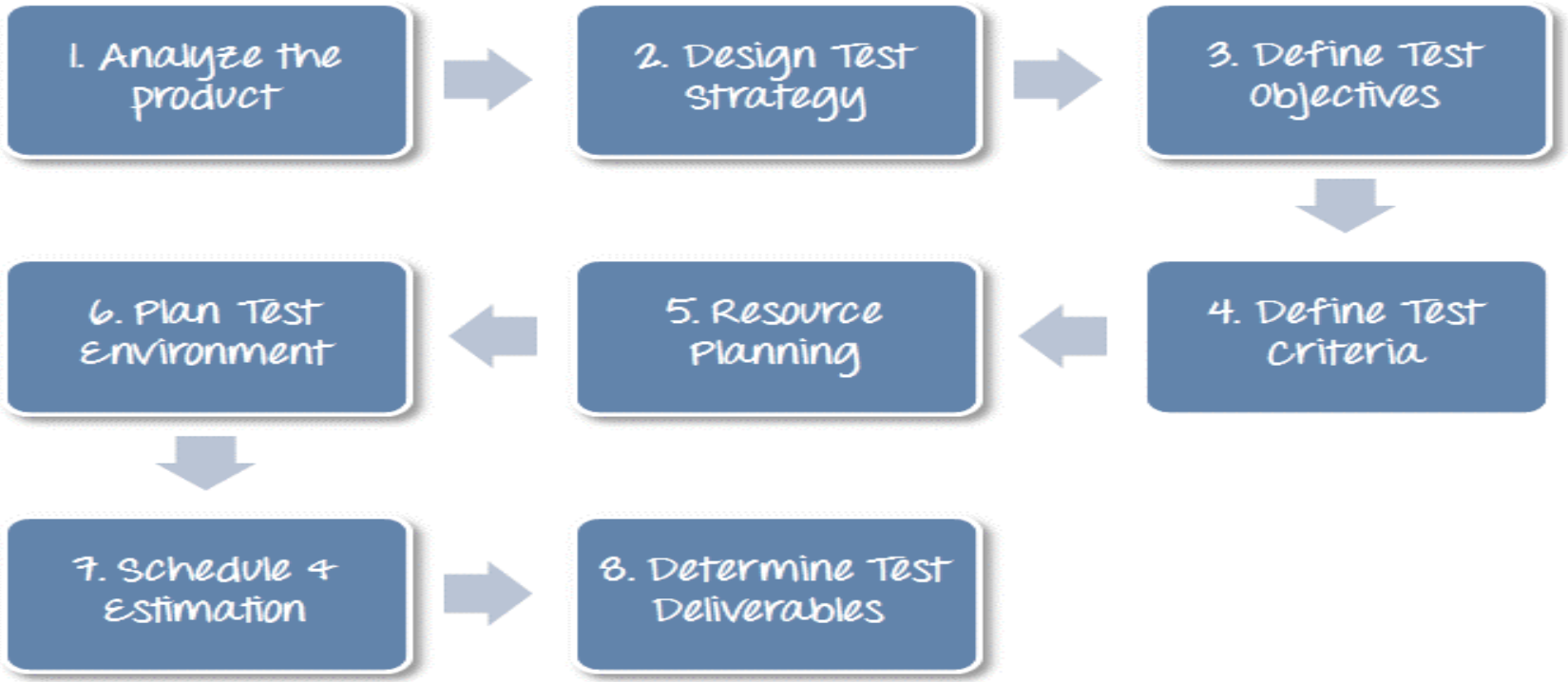
Test environment

Ready to Test

Test Plan Example

A. Goals and Exit Criteria (Quality, Robustness, Schedule, Performance Goals of the Product, ...)
B. Items to be Tested/Inspected (Executables such as modules and components, Nonexecutables such as Requirments and Design specifications, ...)
C. Test Process/Methodologies (Unit, Functional, Acceptance, Regression Tests, Black-box, White-box, Test metrics, Bug report process, ...)
D. Resources (People, Tools, Test Environment, ...)
E. Schedule (Test-case development, Test execution, Problem reporting and fixing, ...)
F. Risks (...)
G. Major Test Scenarios and Test Cases (...)

How to make a Test Plan



http://www.guru99.com/what-everybody-ought-to-know-about-test-planing.html

Test Cases List Example

Tester: _____ , Date: _____

If Test Cases Fails, report Bugs in VSO

Test Case	OK	Failed	Description
The Login Procedure works			
User Data Saved in the Database			
etc			

The Testers fill in these Lists electronically. Should be included in Software Test Documentation

Test Planning Summary

- Test planning involves scheduling and estimating the system testing process, establishing process standards and describing the tests that should be carried out.
- As well as helping managers allocate resources and estimate testing schedules, test plans are intended for software engineers involved in designing and carrying out system tests.
- They help technical staff get an overall picture of the system tests and place their own work in this context.
- As well as setting out the testing schedule and procedures, the test plan defines the hardware and software resources that are required.
- Test plans are not a static documents but evolve during the development process. Test plans change because of delays at other stages in the development process.
- Test planning is particularly important in large software system development.
- **For small and medium-sized systems, a less formal test plan may be used, but there is still a need for a formal document to support the planning of the testing process.**

http://www.softwareengineering-9.com/Web/Testing/Planning.html

Test Environment

Hans-Petter Halvorsen, M.Sc.

Why Do We Need a Test Environment?

Why cant we just use our own PC?

Why Test Environment?

- “It works on my PC” says the Developer
- Clean Environment
- On the Developers PCs we have all kind of Software installed that the Customer dont have, e.g. Development Tools like Visual Studio, etc.
- We need to test on different Platforms and Operating Systems
- Customers may use different Web Browsers
- Deployment: Test of Installation packages
- Make the software available for Testers
- etc.

“It works on my Computer”

Make sure to test your software on other Computers and Environments than your Development Computer!

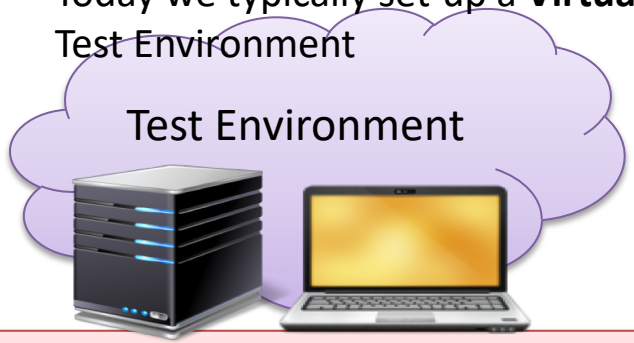
- Everything works on the Developer Computer
- The Customers Database is not the same as yours
- The Customer may not use the same OS
- The Customer may not use the same Web Browser
- The Customer do not have Visual Studio, SQL Server, etc. on their Personal Computer
- Etc.

=> Test Environment is needed!

Typically the Developers Personal Computer with Database, Web Server and Programming Software

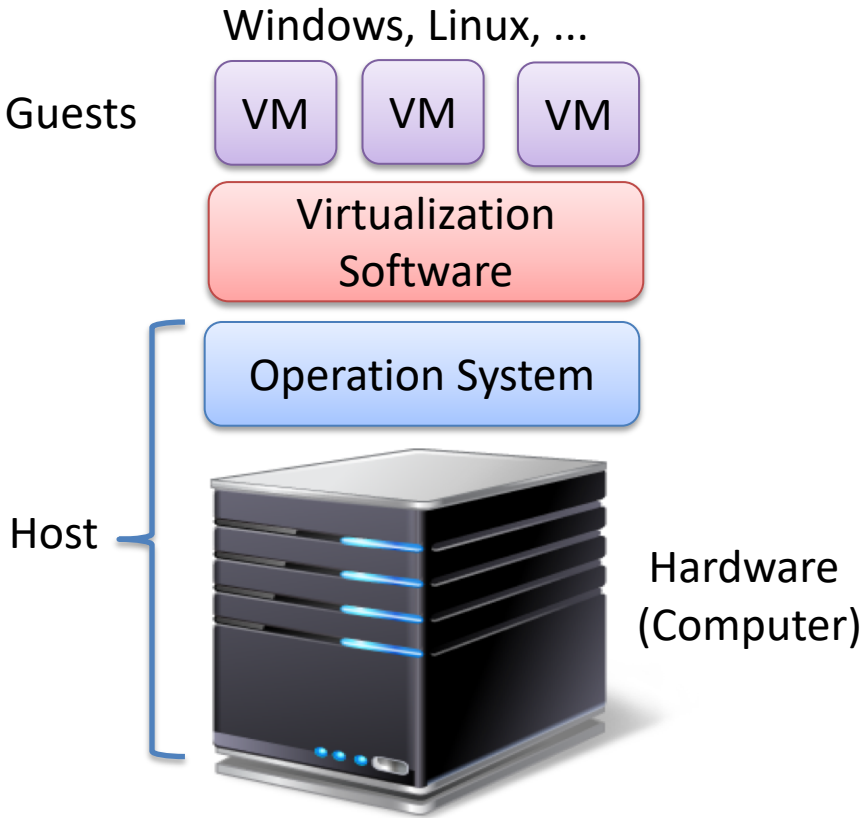
A Clean PC/Server (or a network with PCs and Servers) where you install and test your Software. Today we typically set-up a **Virtual** Test Environment

The Customers environment where you unstall the final software (Servers and Clients)

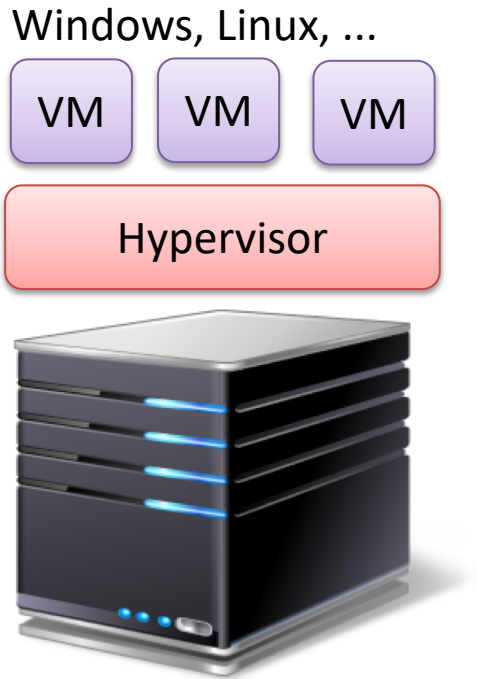


Programming environments such as Visual Studio, etc. should not be installed in this environment. You need to create .exe files etc. in order to make your software run.

Virtualization



VM = Virtual Machines



A Hypervisor can run directly on the computer without a Host OS

Virtualization Software

A lot of Virtualization Software exists. Here are some examples:

- VMware Workstation
- **VMware Workstation Player** (Free of charge and simple to use)
- VMware vSphere and vSphere Hypervisor
- VMware Fusion (Mac)
- Parallels Desktop (Mac)
- Microsoft Hyper-V
- VirtualBox
- etc.

VMware Workstation Player

VMware Workstation Player is for personal use on your own PC. VMware Player is free of charge for personal non commercial use.

VMware is a company that has been specializing within virtualization software.
http://www.vmware.com

When are you finished Testing?

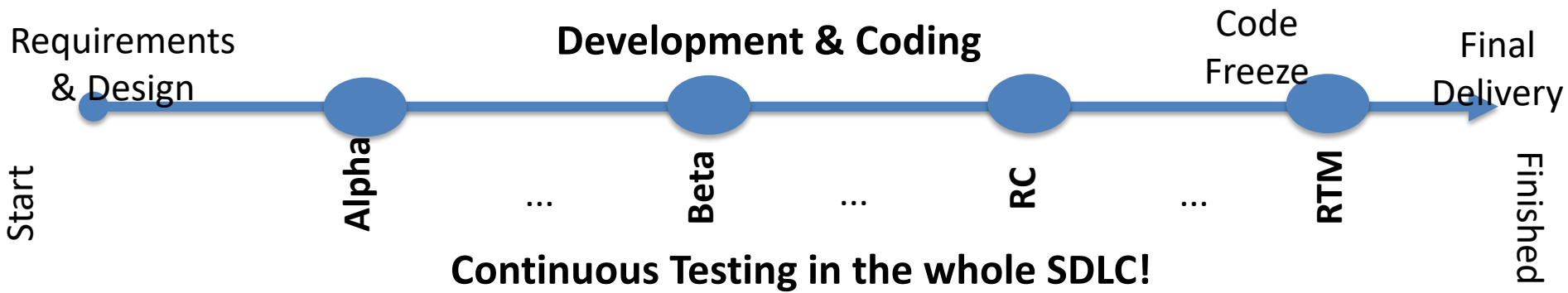
Hans-Petter Halvorsen, M.Sc.

Software Testing

“50% of the software development is about testing your software”

When are we finished with Testing?

Testing



Continuous Testing in the whole SDLC!

Increased Focus

Requirements

Increased Focus

Requirements & Functionality

Increased Focus

Functionality

Increased Focus

Software without Critical Bugs

You can never find all Bugs!

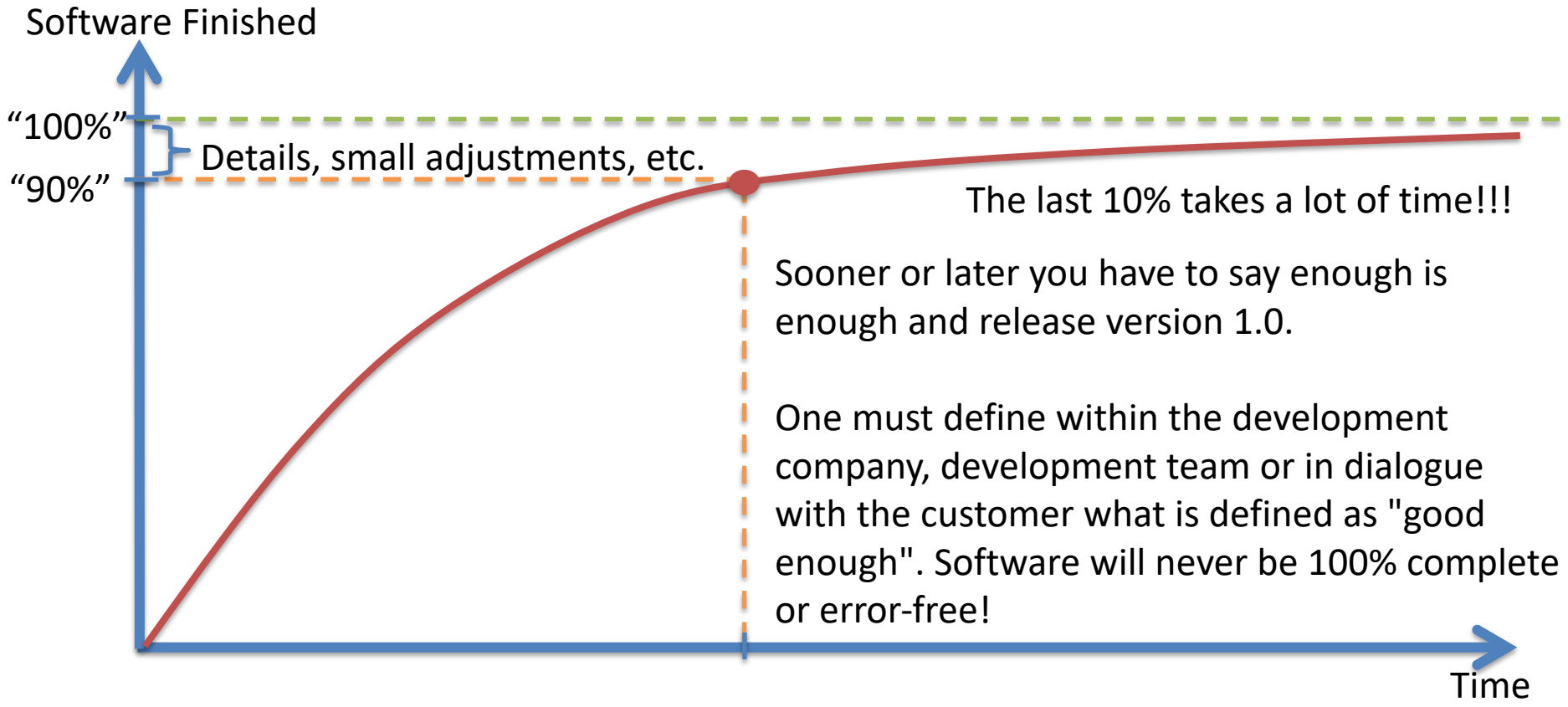
Released Software do have Bugs!

Agile/Scrum: Periodically Iterations/Sprint every 14-30 days

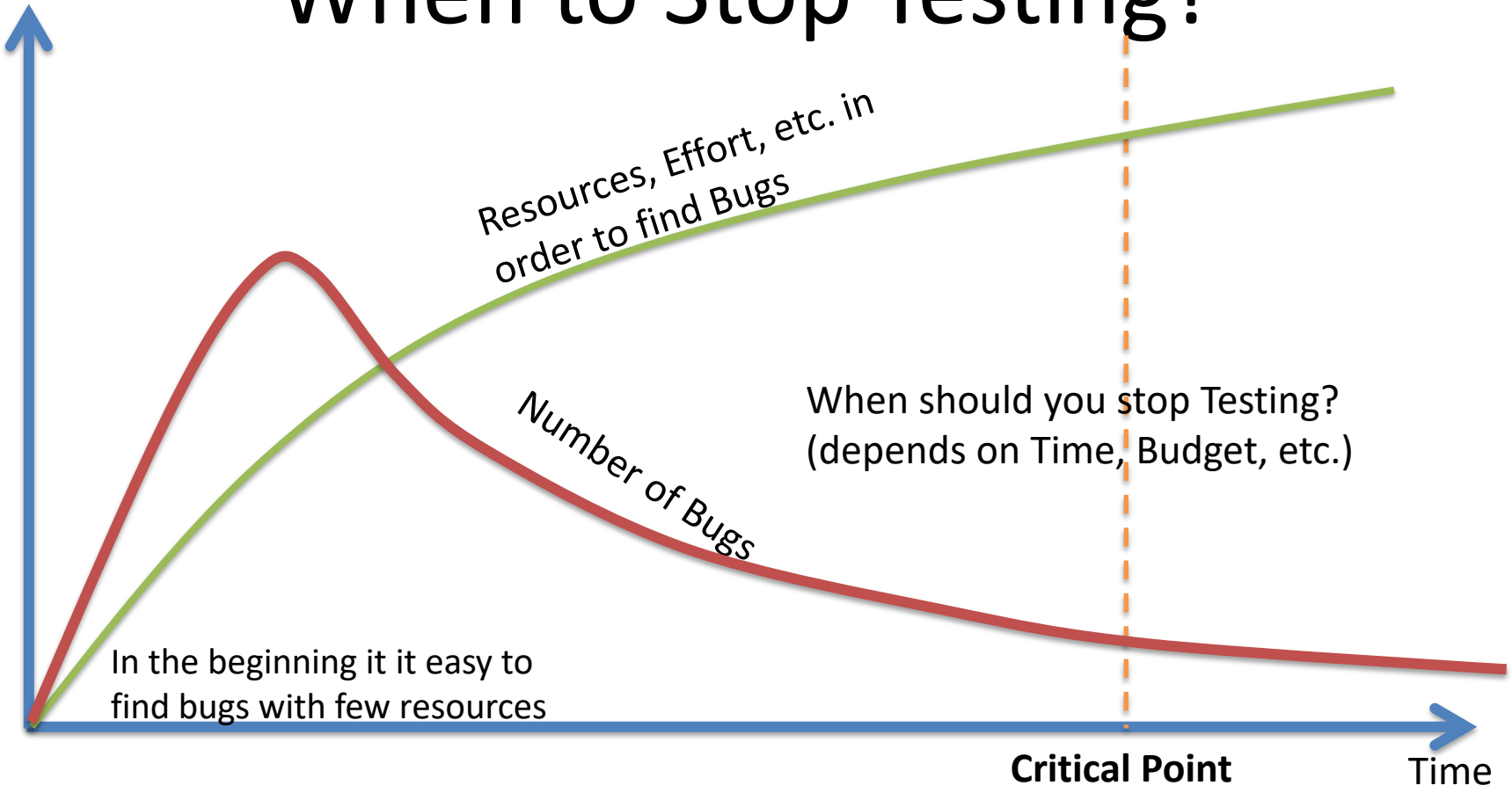
When to Stop Testing?

- A simple answer is to stop testing when all the planned test cases are executed and all the problems found are fixed.
- In reality, it may not be that simple. We are often pressured by schedule to release software product.

When to Stop Development?



When to Stop Testing?



When to Stop Testing?

- When the tester has not been able to find another defect in 5 (10? 30? 100?) minutes of testing
- All code reviews and walkthroughs have certified the code as ok
- When a given checklist of test types has been completed
- The code has passed all unit tests
- When testing runs out of its scheduled time
- ...

E. J. Braude and M. E.Bernstein, *Software Engineering: Modern Approaches*, 2 ed.: Wiley, 2011. +++

Bug Tracking Systems

Hans-Petter Halvorsen, M.Sc.

Bug Tracking Systems

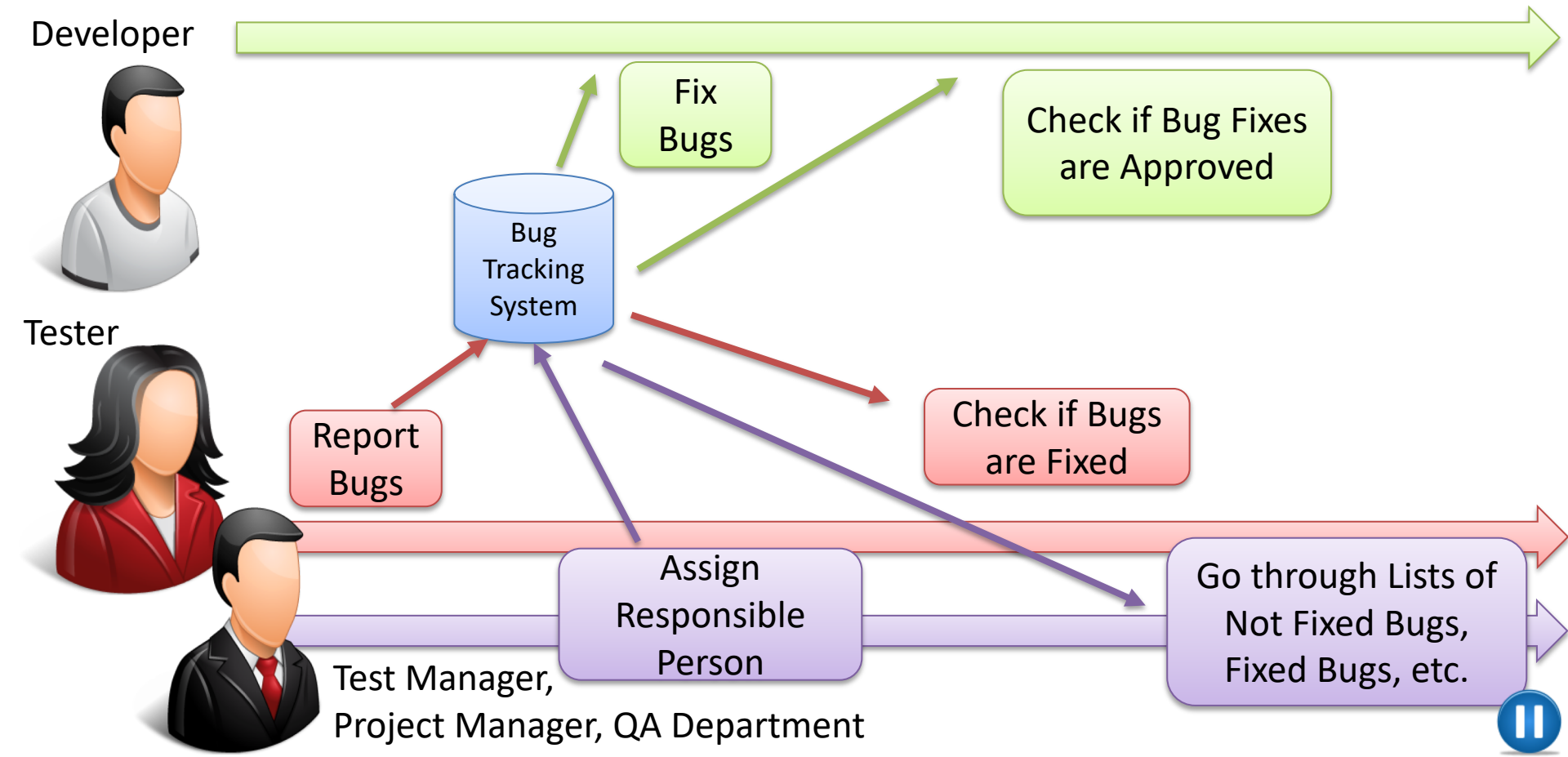
- A “bug tracking system” or “defect tracking system” is a software application that keeps track of reported software bugs in software development projects.
- It may be regarded as a type of “issue tracking system”.
- Typically bug tracking systems are integrated with other software “project management applications” – e.g., Visual studio Team Services, Jira, etc.

https://en.wikipedia.org/wiki/Bug_tracking_system

Bug Tracking Software

- Team Foundation Server/Visual Studio Team Services
- Jira
- Bugzilla
- Clearquest
- … (hundreds)

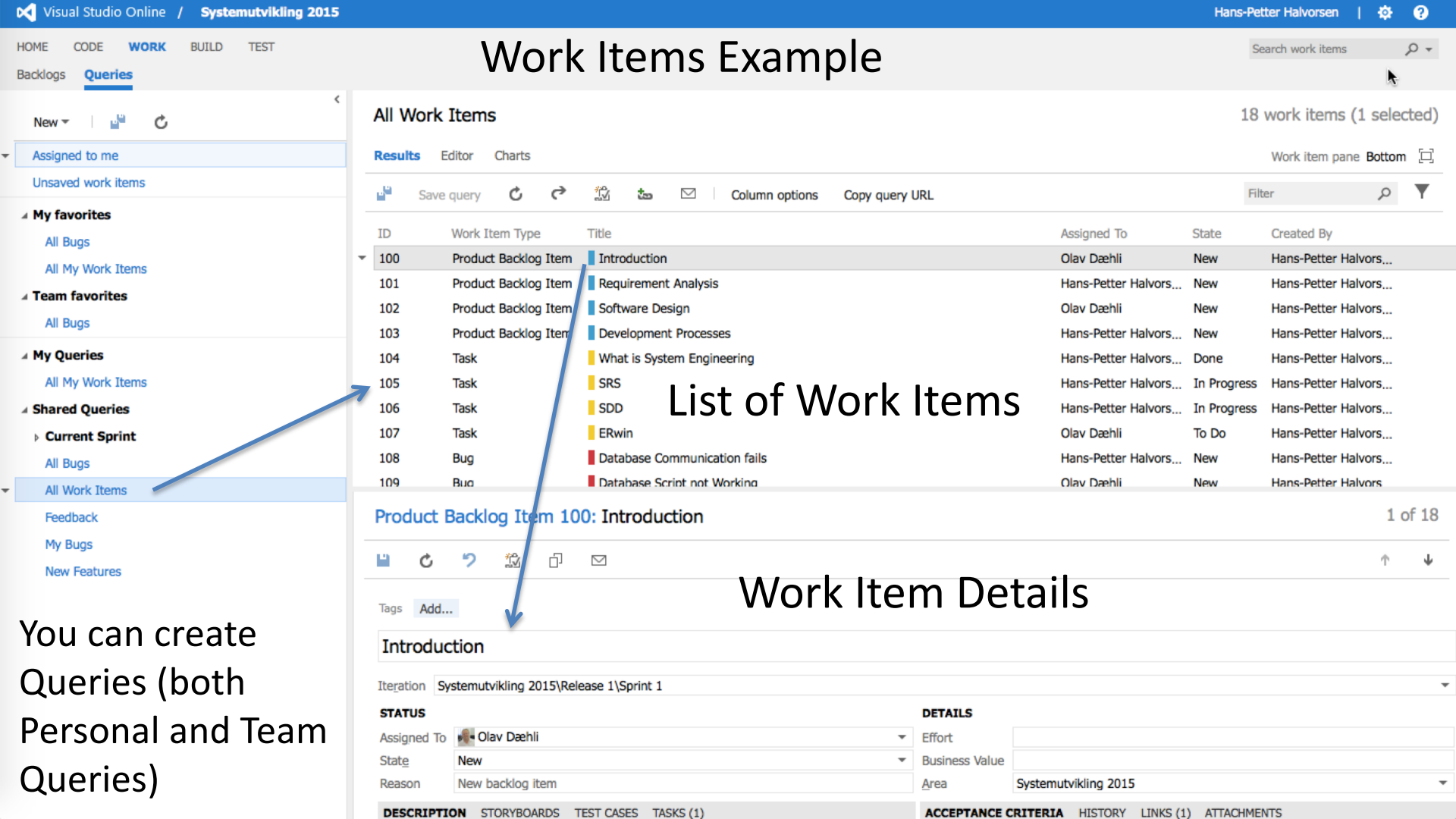
Bug Reporting and Tracking



Visual Studio Team Services

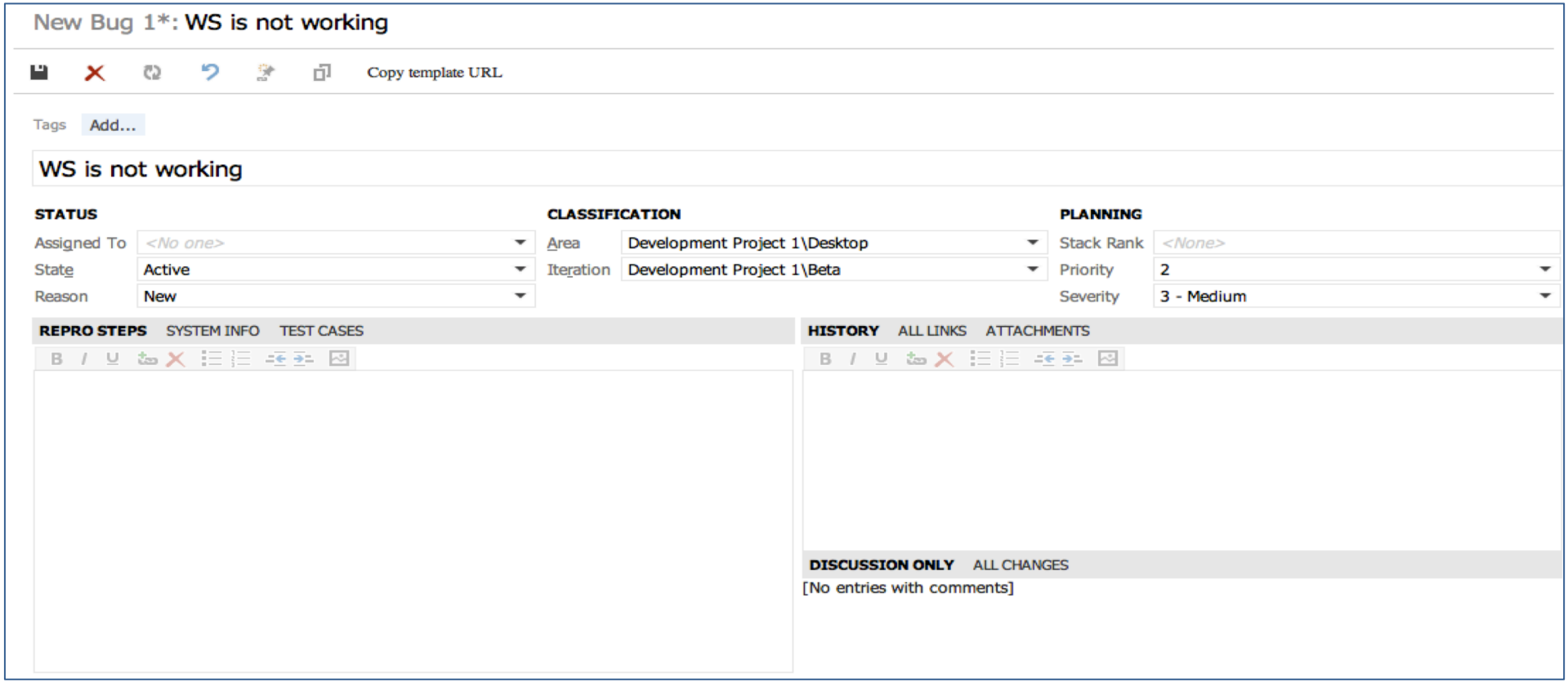
Hans-Petter Halvorsen, M.Sc.

Work Items Example



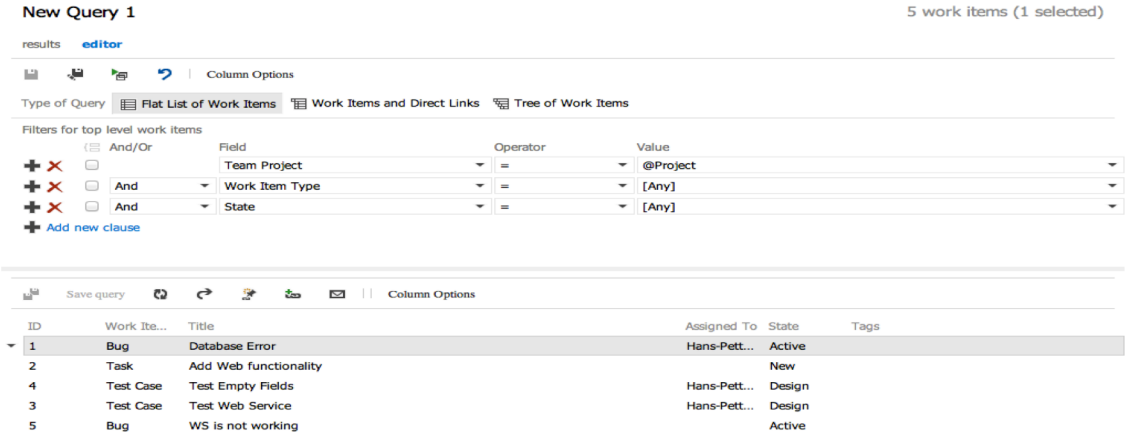
You can create Queries (both Personal and Team Queries)

Work Items – New Bug

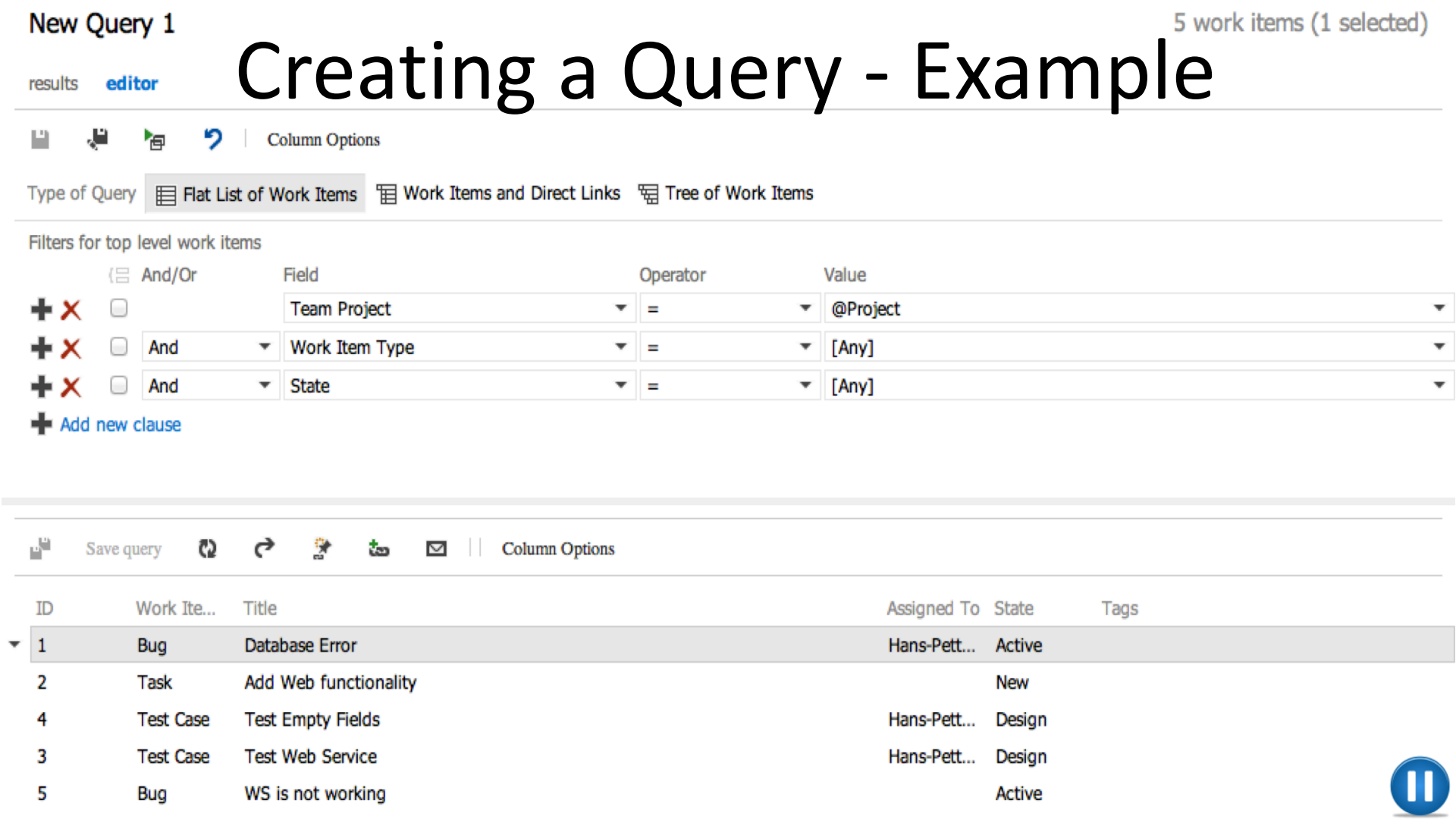


Queries

- Used to find existing Work Items
- You may create different Queries to make it easy to find the Work Items you need
- Queries may be personal or visible for everybody in the project (Team Queries)



Creating a Query - Example



Code Review & Refactoring

Hans-Petter Halvorsen, M.Sc.

What is Refactoring?

- Even when using best practices and making a conscious effort to produce high-quality software, it is highly unlikely that you will consistently produce programs that cannot be improved.
- Refactoring is
  - the activity of improving your code style without altering its behavior
  - a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior

Refactoring - Symptoms

- Coding Style and Name Conventions not followed
- Proper Commenting not followed
- Duplicated code (clearly a waste).
- Long method (excessively large or long methods perhaps should be subdivided into more cohesive ones).
- Large class (same problem as long method).
- Switch statements (in object-oriented code, switch statements can in most cases be replaced with polymorphism, making the code clearer).
- Feature envy, in which a method tends to use more of an object from a class different to the one it belongs.
- Inappropriate intimacy, in which a class refers too much to private parts of other classes.

=> Any of these symptoms (and more) will indicate that your code can be improved. You can use refactoring to help you deal with these problems.

Hans-Petter Halvorsen, M.Sc.

University College of Southeast Norway

www.usn.no

E-mail: hans.p.halvorsen@hit.no

Blog: http://home.hit.no/~hansha/