https://www.halvorsen.blog

Week Assignment

Software Testing – Test Planning

Hans-Petter Halvorsen

Week Assignment

1. Get an overview of **Software Testing** in general
2. Create a **Software Test Plan** (STP)
3. Create a **Virtual Test Environment**

CODE FREEZE

In software engineering, a freeze is a point in time in the development process after which the rules for making changes to the source code or related resources become stricter, or the period during which those rules are applied. https://en.wikipedia.org/wiki/Freeze_(software_engineering)

No Programming in Class these 2 weeks! – otherwise it is easy to lose focus on Testing

Test Planning and Execution

1. Create **Software Test Plan** (STP)
2. Create **Virtual Test Environment**

Test Planning

---

3. **Test** the Software according to STP
4. Create **Unit Tests** in Visual Studio

Test Execution

Next Week

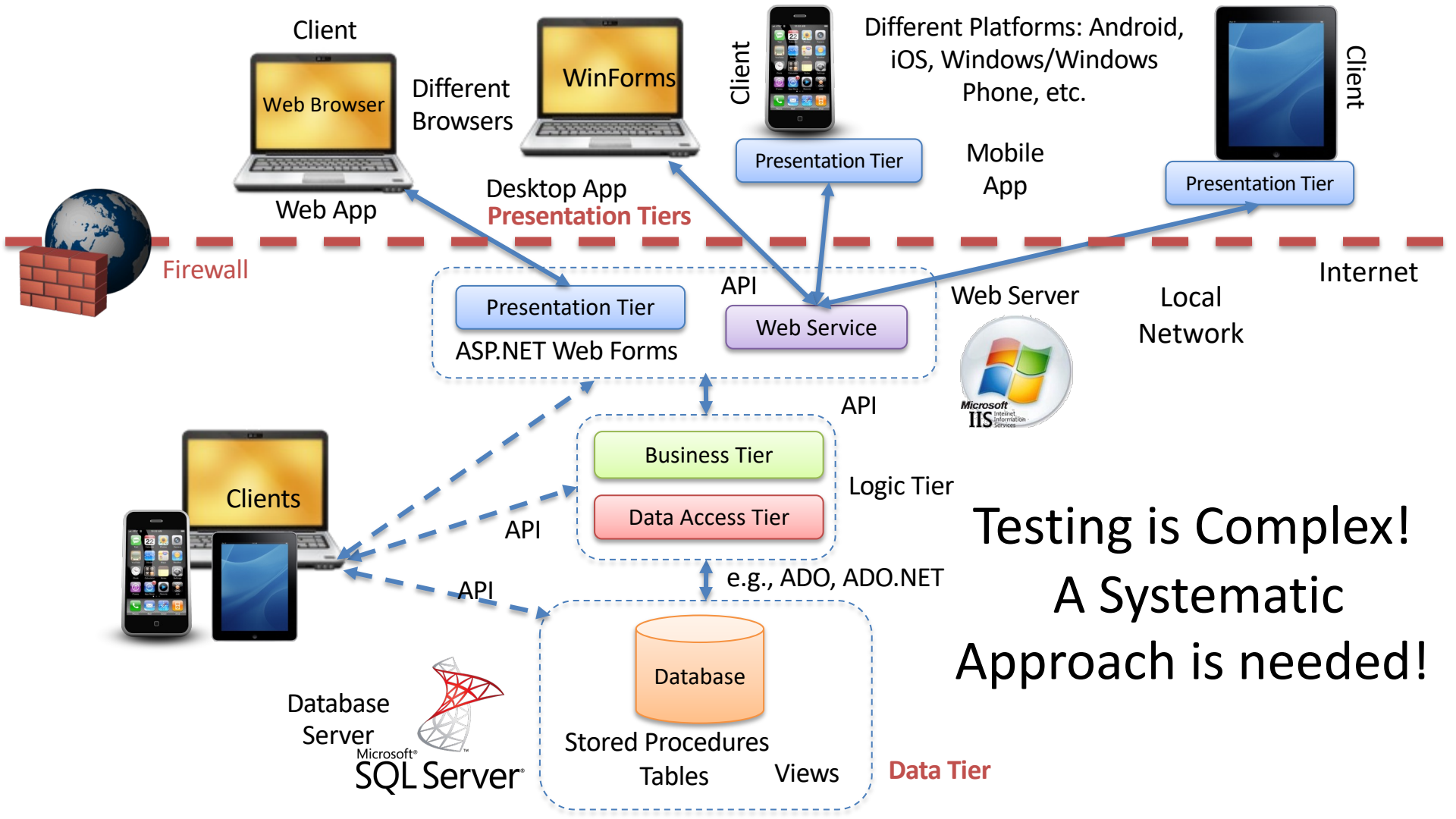
Software Testing

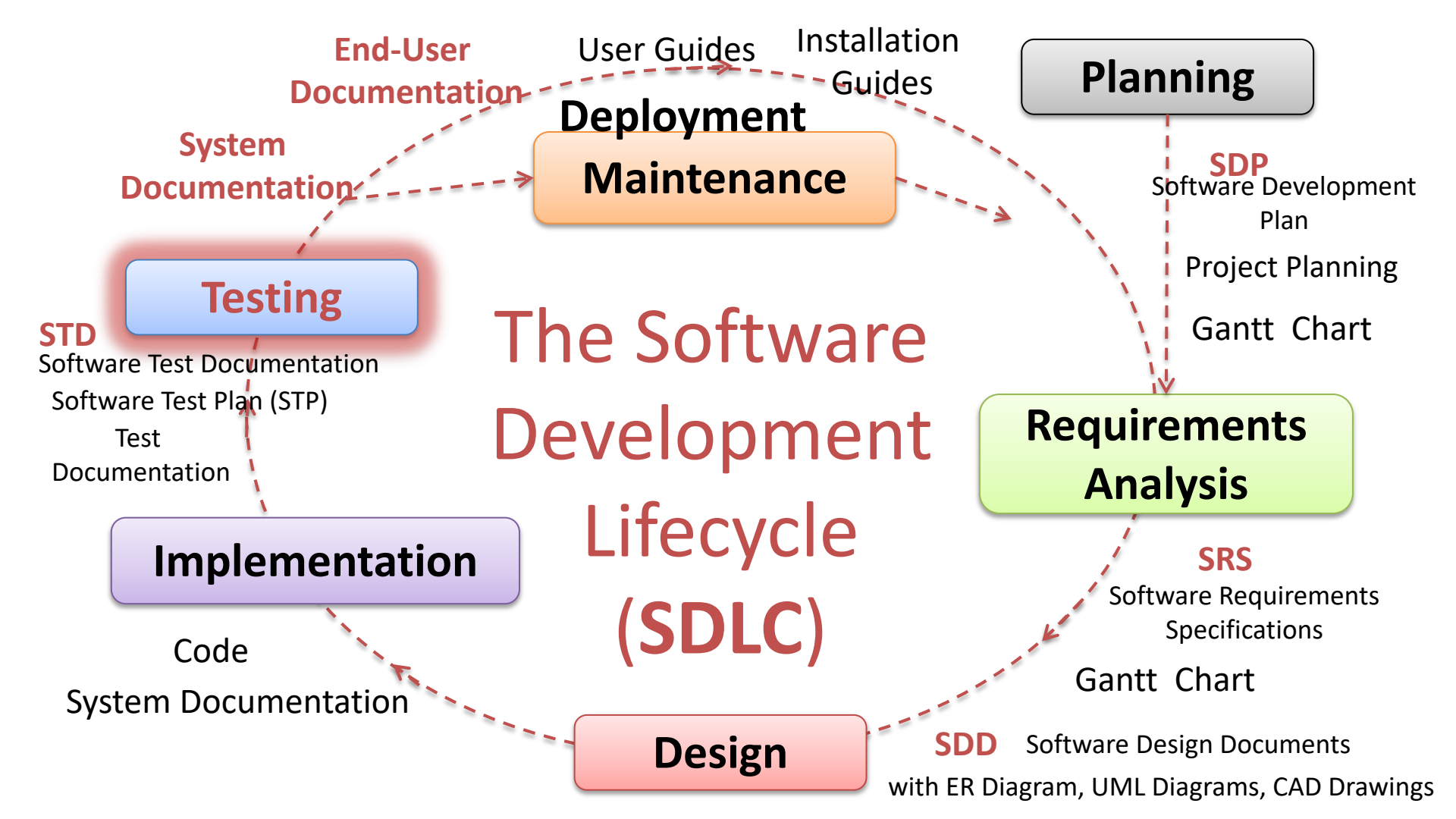
Hans-Petter Halvorsen

Table of Contents

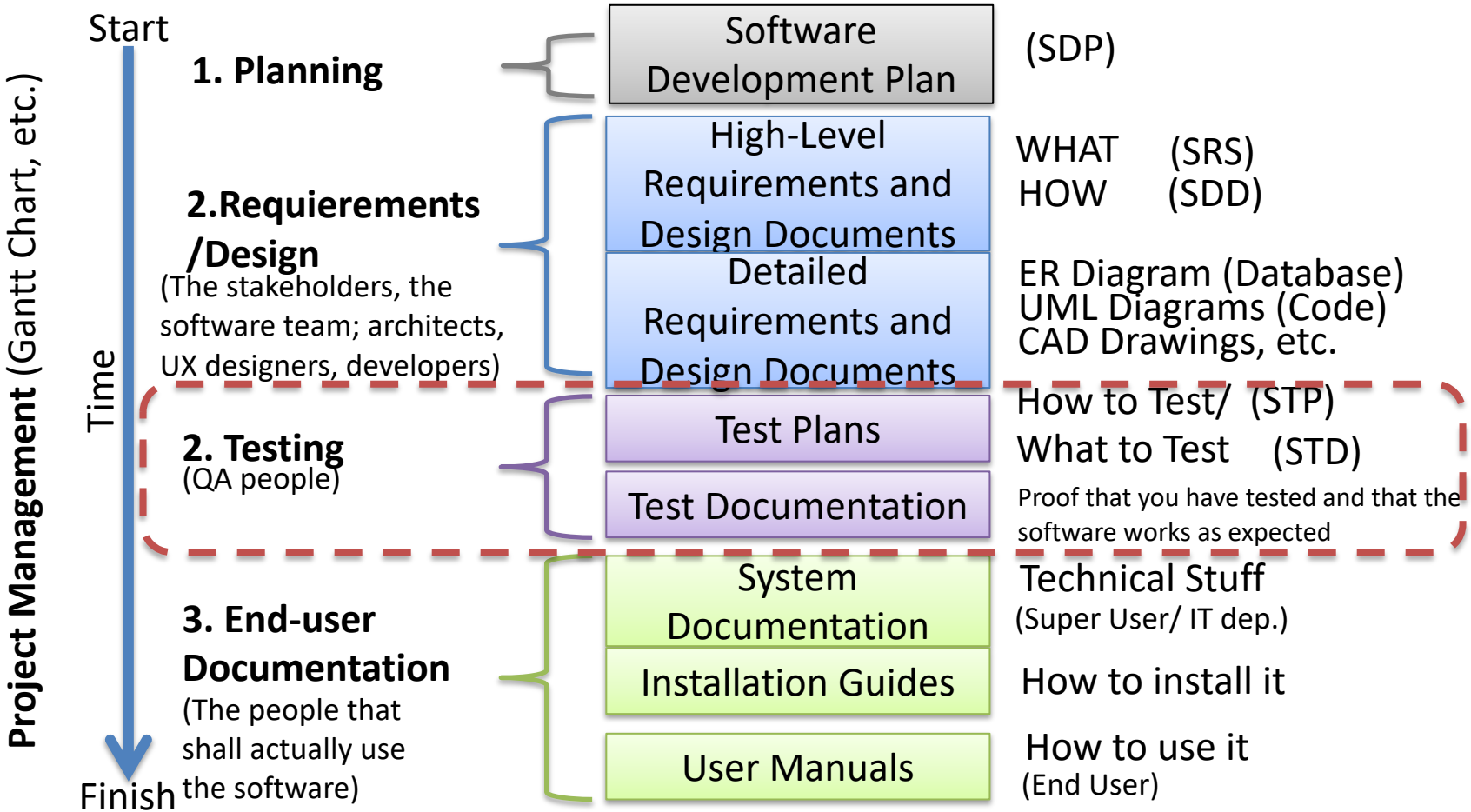
Why Testing?

- Make sure the software fulfills the **Requirements** from the Customers (Software Requirements Specification, SRS)
- Make sure the Software don't contain critical **Bugs**
- Make sure the software can be **installed** at the customer. The customer don't have Visual Studio!
- Make sure the software are **user-friendly** an intuitive to use
- Make sure the software is **robust** and has acceptable **performance** (so it don't crash when more than 1 person are using it, the database contain lots of data, etc.)


Client
Web Browser
Different Browsers
WinForms
Client
Different Platforms: Android, iOS, Windows/Windows Phone, etc.
Client
Presentation Tier
Mobile App
Presentation Tier
Web App
Desktop App
Presentation Tiers
Firewall
Internet
Presentation Tier
ASP.NET Web Forms
API
Web Service
Web Server
Local Network
Microsoft IIS
API
Business Tier
Data Access Tier
Logic Tier
Clients
API
API
e.g., ADO, ADO.NET
Database
Database Server
Microsoft SQL Server
Stored Procedures
Tables
Views
Data Tier
Testing is Complex!
A Systematic
Approach is needed!


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

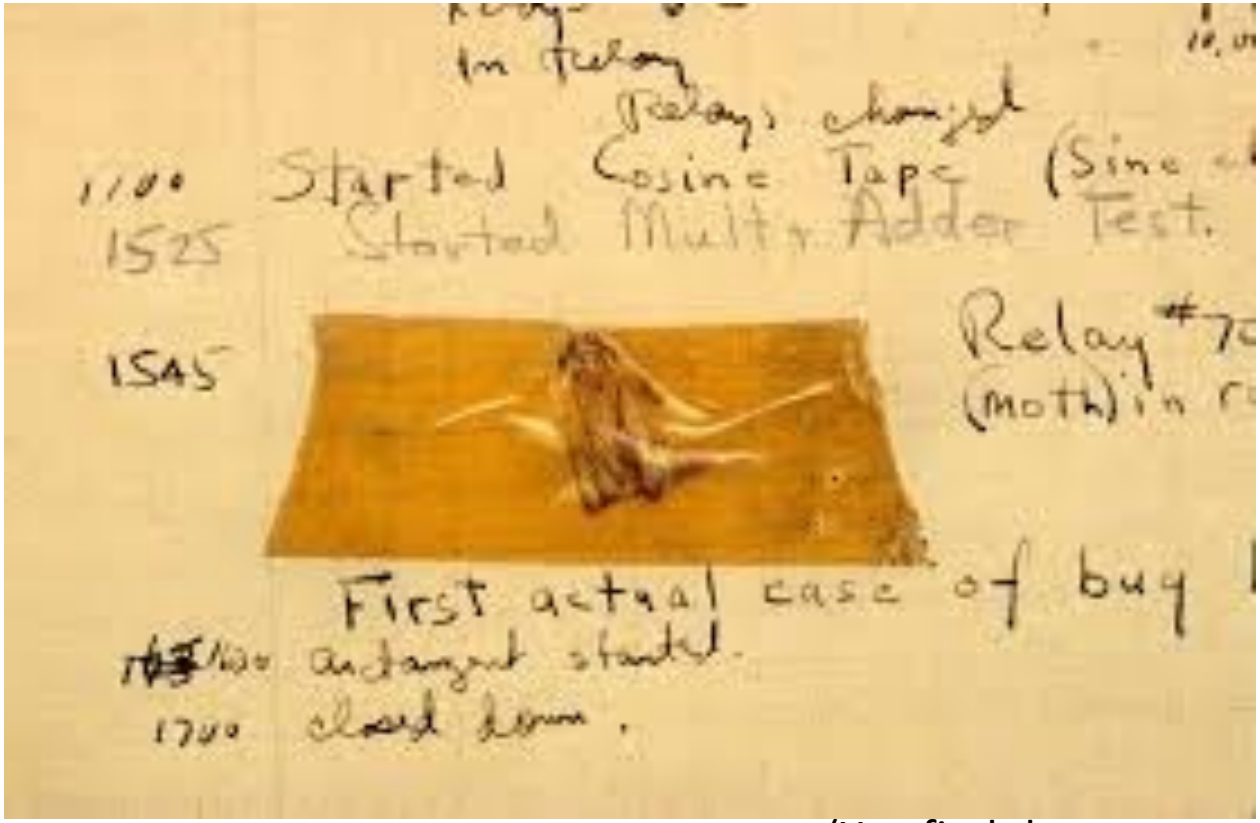


Main purpose of Testing: Find Bugs!!

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

Tsui, Frank et al. (2016). *Essentials of Software Engineering*. 4 ed. Jones & Bartlett Learning

What is this?



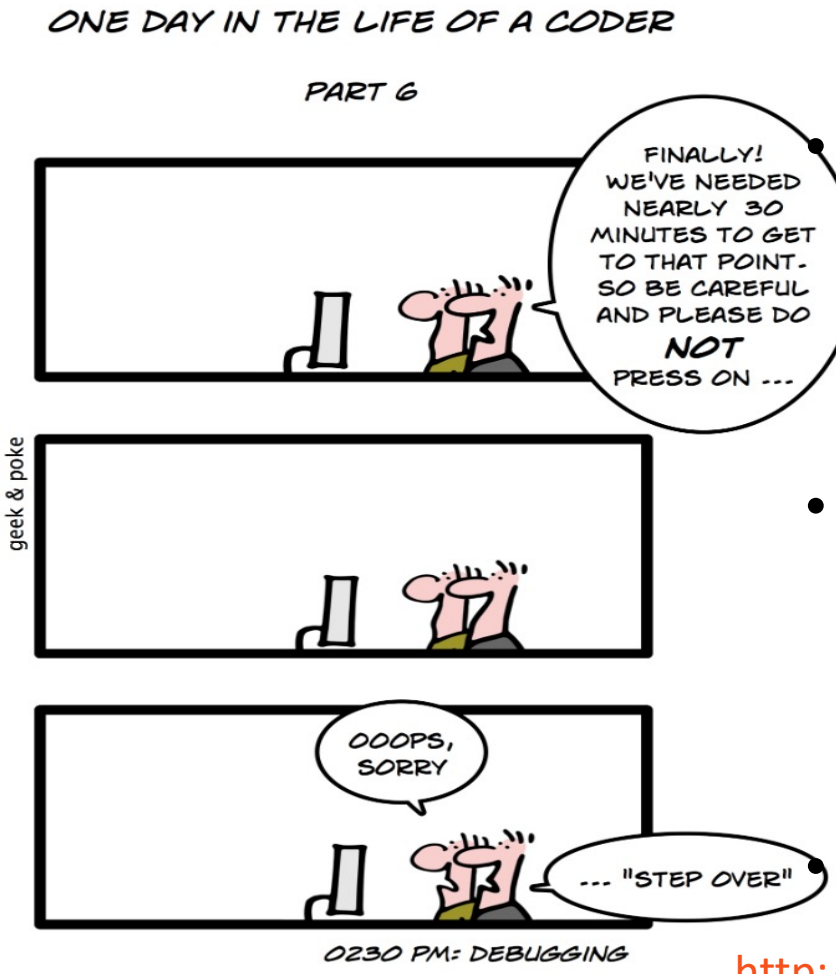
(You find the answer on the next slide)

The first Bug ever found

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.

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

What is Bugs



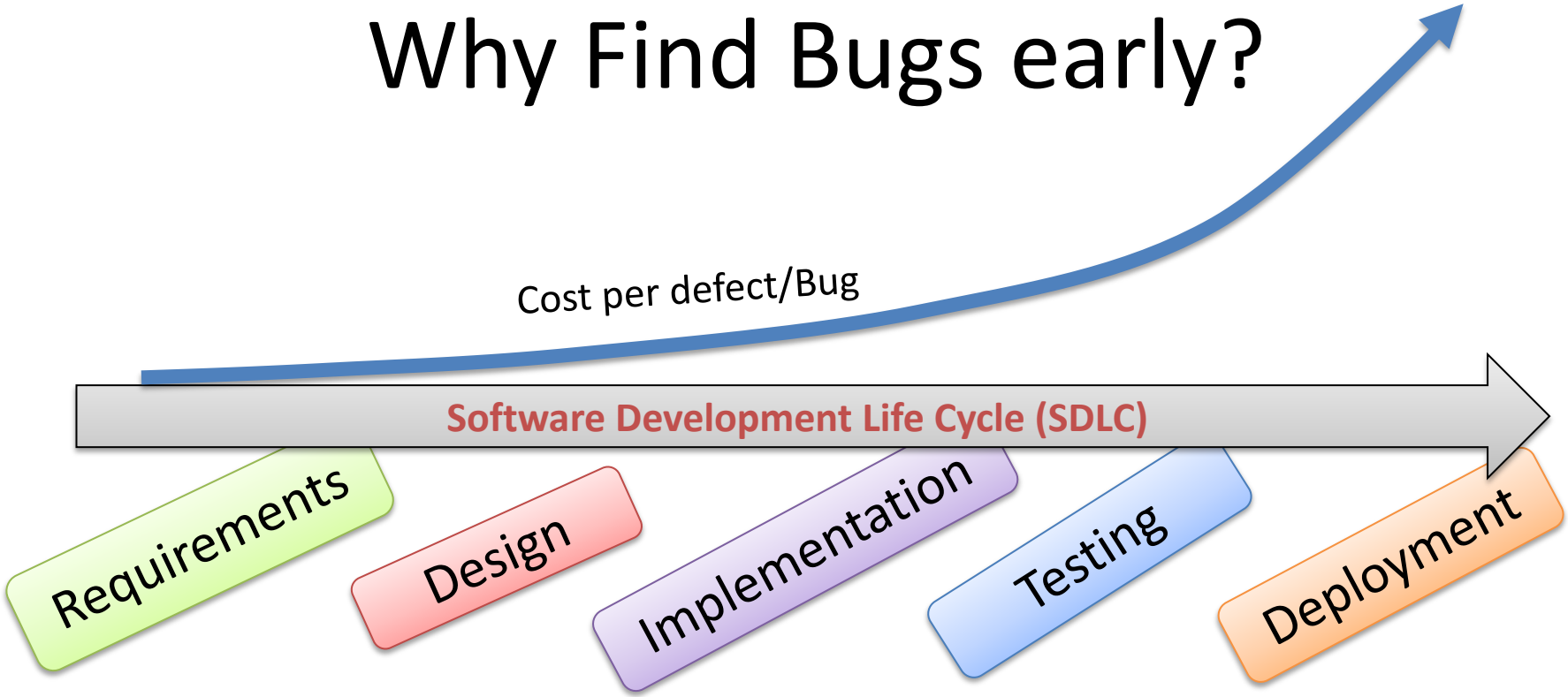
A software bug is an error, flaw, failure, or fault in a computer program or system that produces an incorrect or unexpected result, or causes it to behave in unintended ways

- 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

http://geek-and-poke.com

Why Find Bugs early?



Software Testing

"If you don't know how your code works, it does not work – you just don't know it yet"

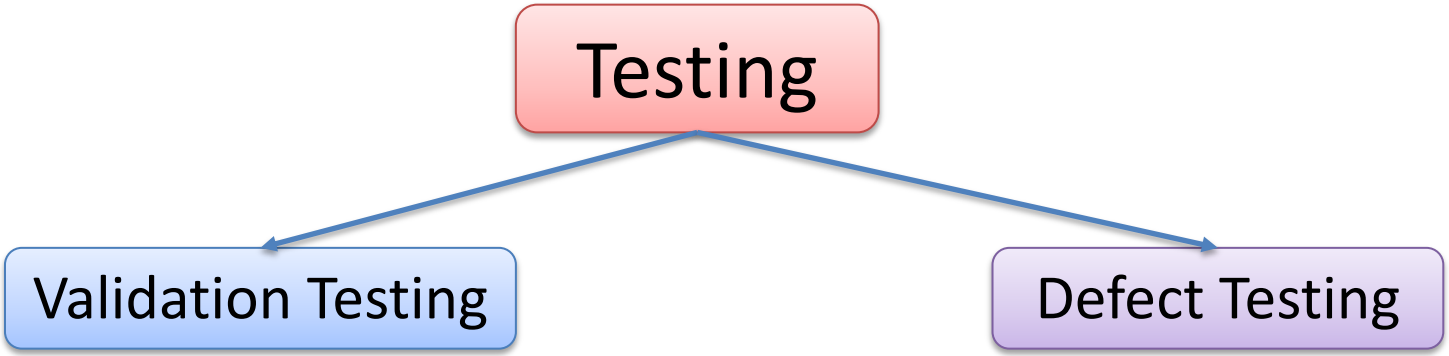
"50% of the Software Development is about Testing your Software"

http://geek-and-poke.com

Testing

“Testing can only show the presence of errors, not their absence”

Dijkstra, 1972



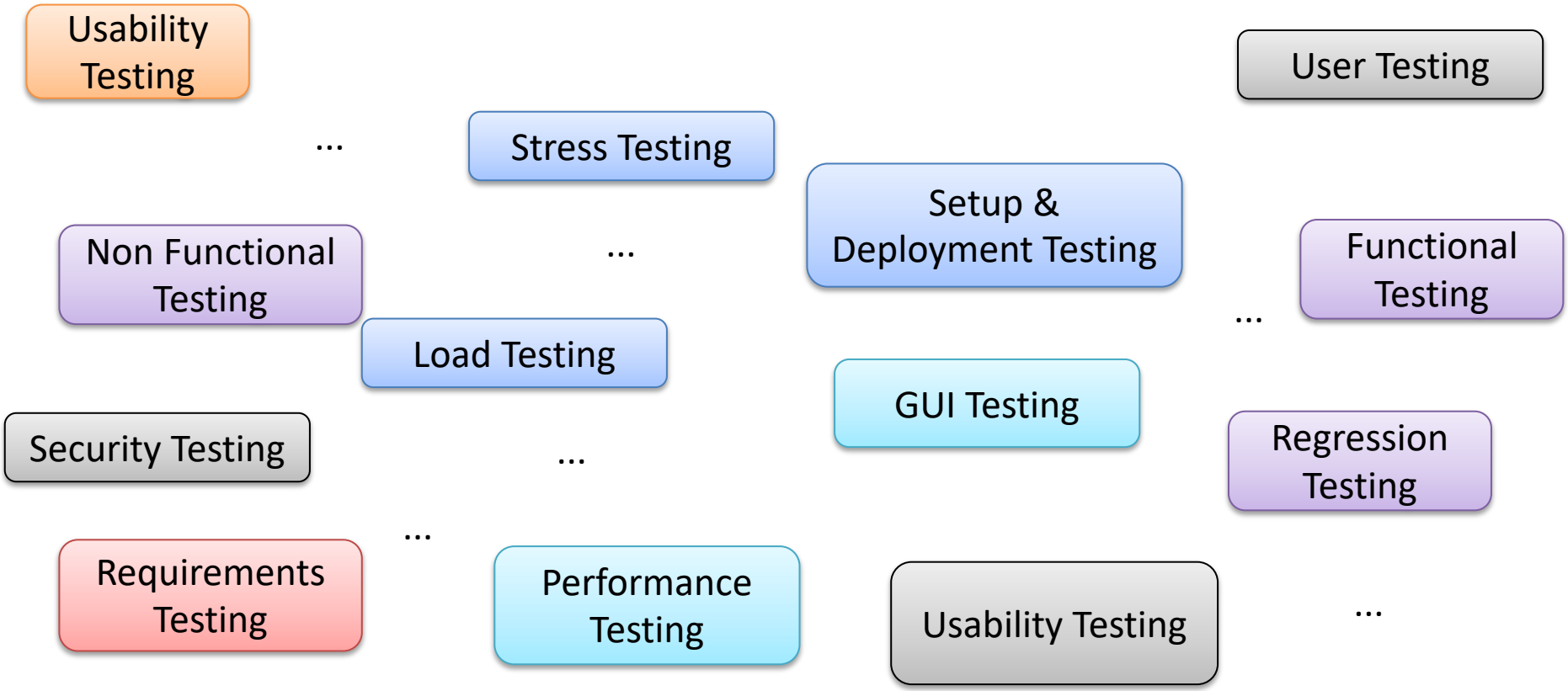
Demonstrate to the Developer and the Customer that the Software meets its Requirements.

Custom Software:
There should be at least one test for every requirement in the SRS document.
Generic Software:
There should be tests for all of the system features that will be included in the product release.

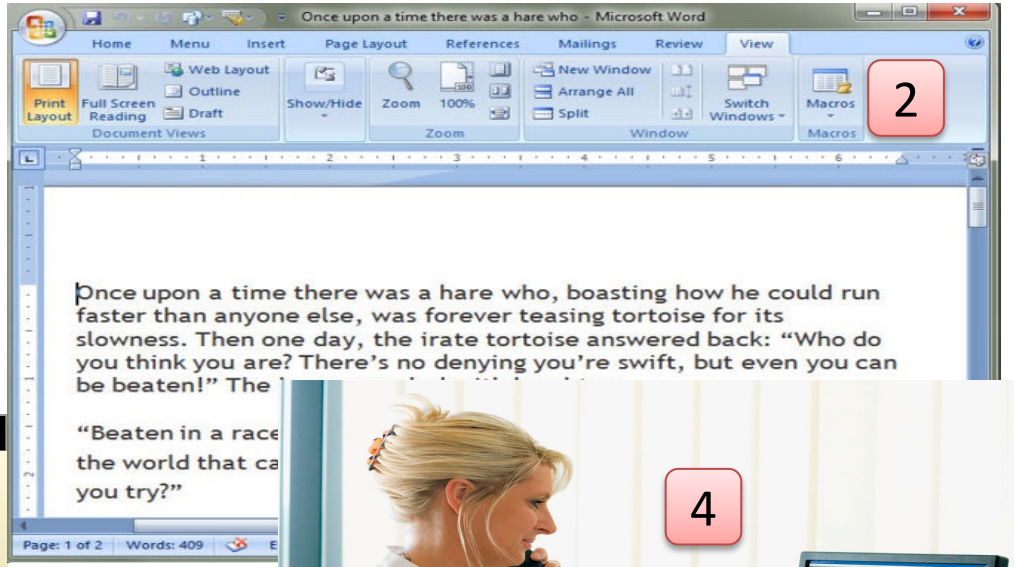
Find inputs or input sequences where the behavior of the software is incorrect, undesirable, or does not conform to its specifications.
These are caused by defects (bugs) in the software.

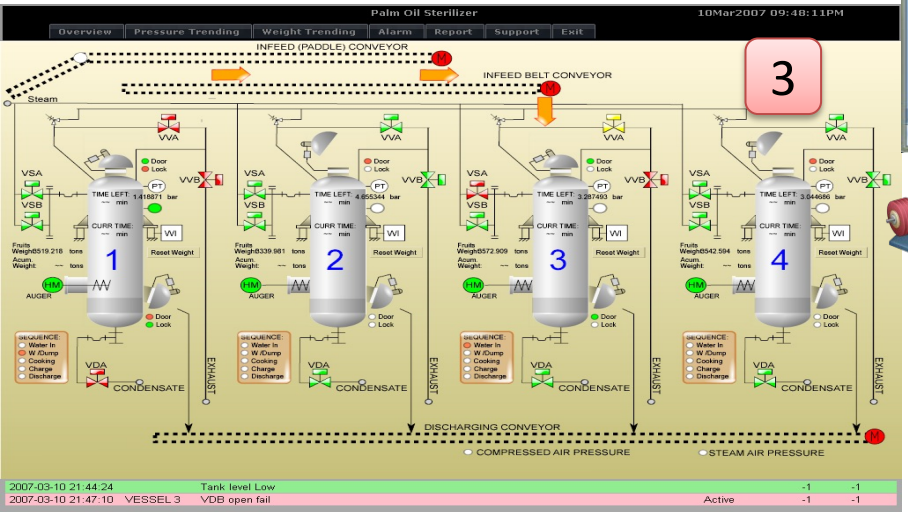
I. Sommerville, *Software Engineering*, 10 ed.: Pearson, 2015.

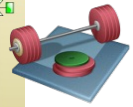
Types of Testing



Different Systems Needs Different Testing







Why?

Videos about Testing

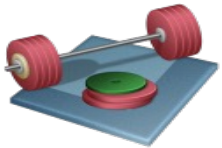
- Guru99.com: http://www.guru99.com/software-testing.html
- NTNU: http://video.adm.ntnu.no/pres/511de3f0ac5b5
- ... (search for Testing on YouTube)

7 Principles of Testing

YouTube

https://www.youtube.com/watch?v=rFaWOw8bIMM

⏰ 5min



Watch this Video

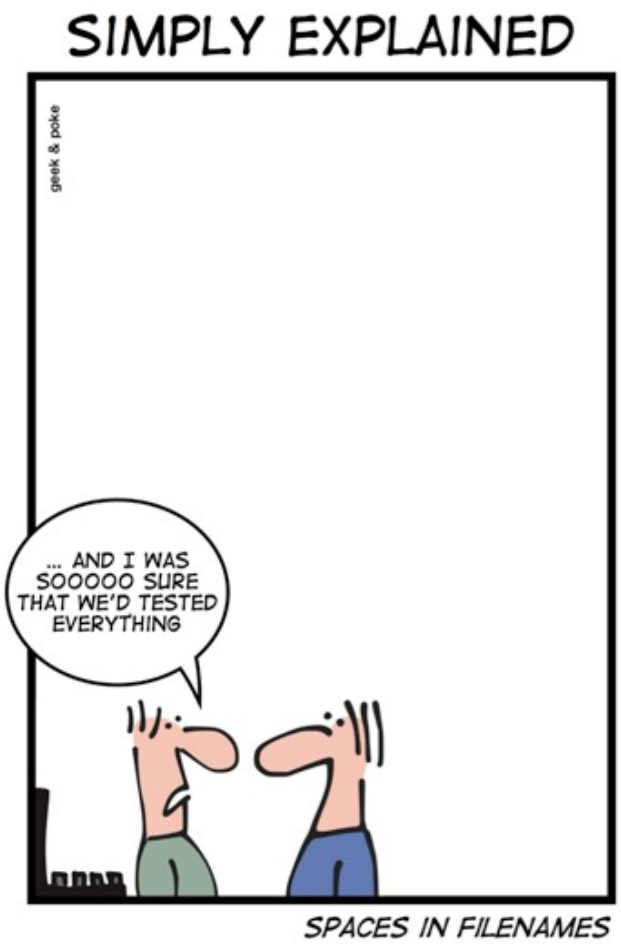
7 Principles of Testing

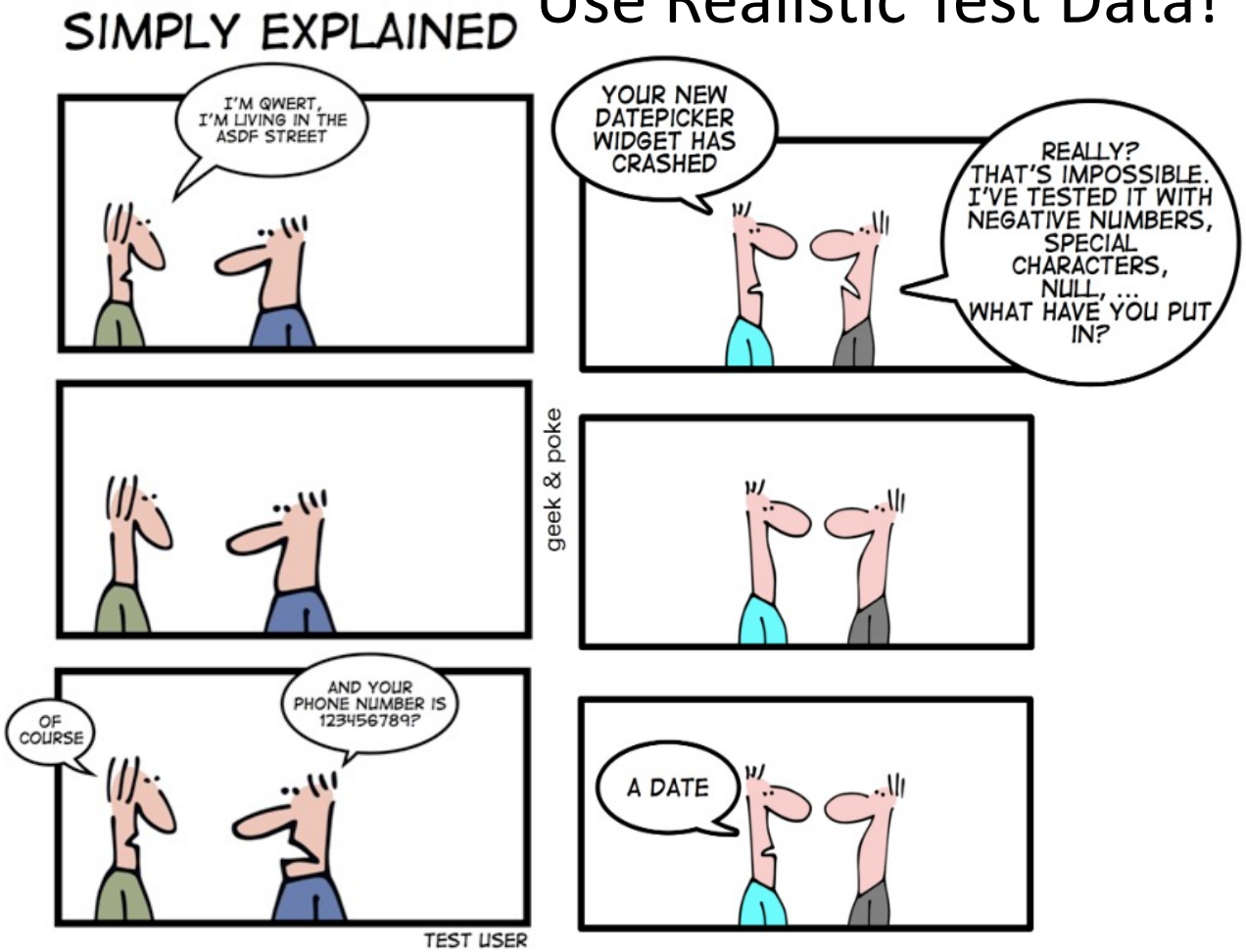
1. **Testing shows the presence of Bugs**: Software Testing reduces the probability of undiscovered defects remaining in the software but even if no defects are found, it is not a proof of correctness.
2. **Exhaustive Testing is impossible**: Testing everything is impossible! Instead we need optimal 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

Use Realistic Test Data!





http://geek-and-poke.com

Spørsmål ifm Testing

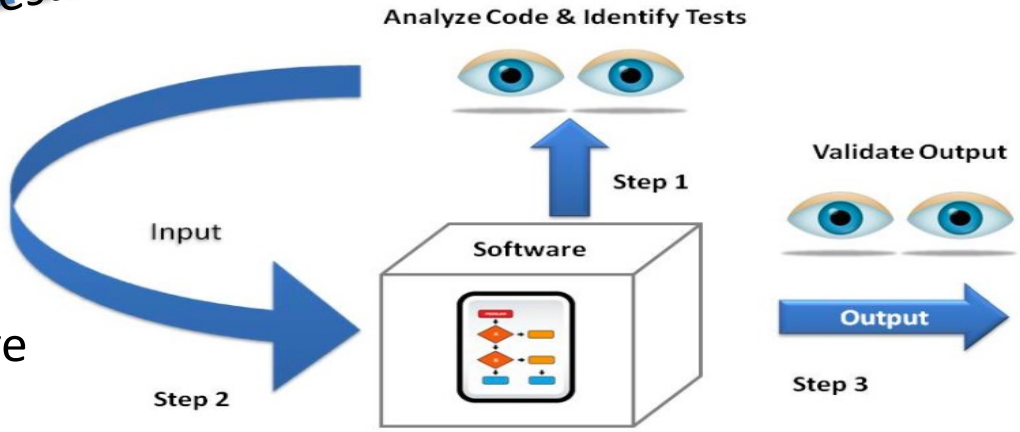
- Hvorfor må vi teste programvare?
- Når kan vi begynne å teste programvare?
- Hva trengs for å utføre testen?
- Hvordan kan / bør vi teste?
- Hvem er best til å utføre testen?
- Hvor er det behov for testing?

Test Categories

Black-box vs. White-box Testing

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

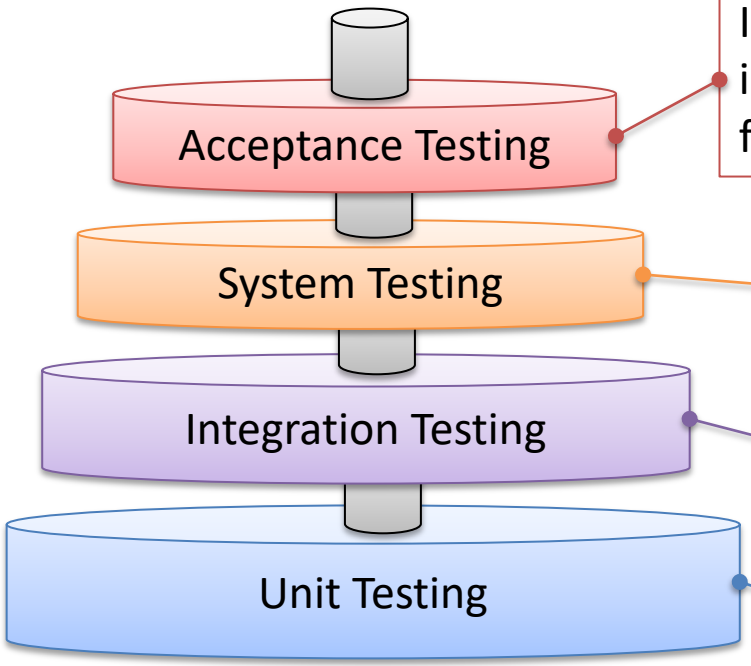
Grey-box Testing



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)

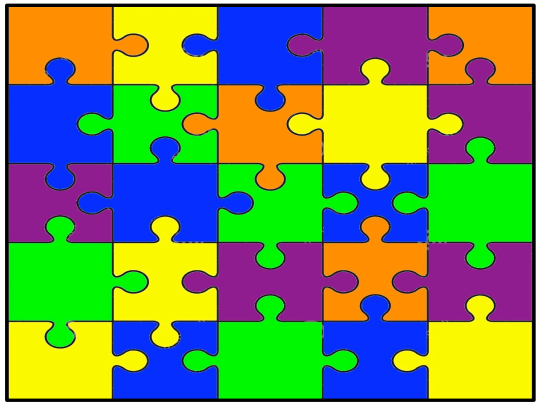
Any module, program, object separately testable

Levels of Testing

Unit Testing: Test each parts independently and isolated

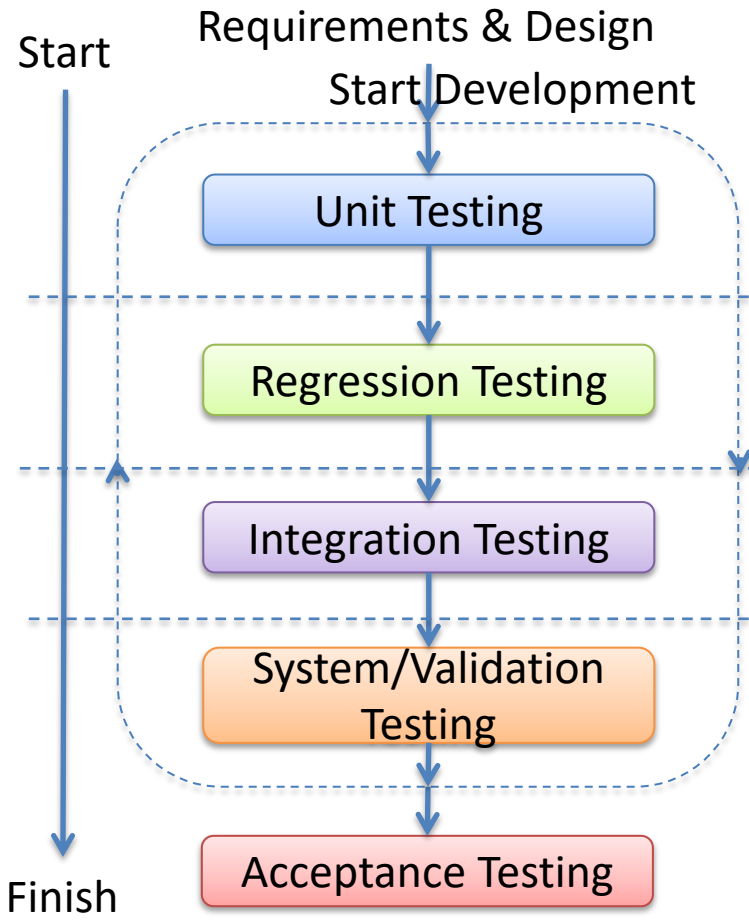
Integration Testing: Make sure that different pieces work together. Test the Interfaces between the different pieces. Interaction with other systems (Hardware, OS, etc.)

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)

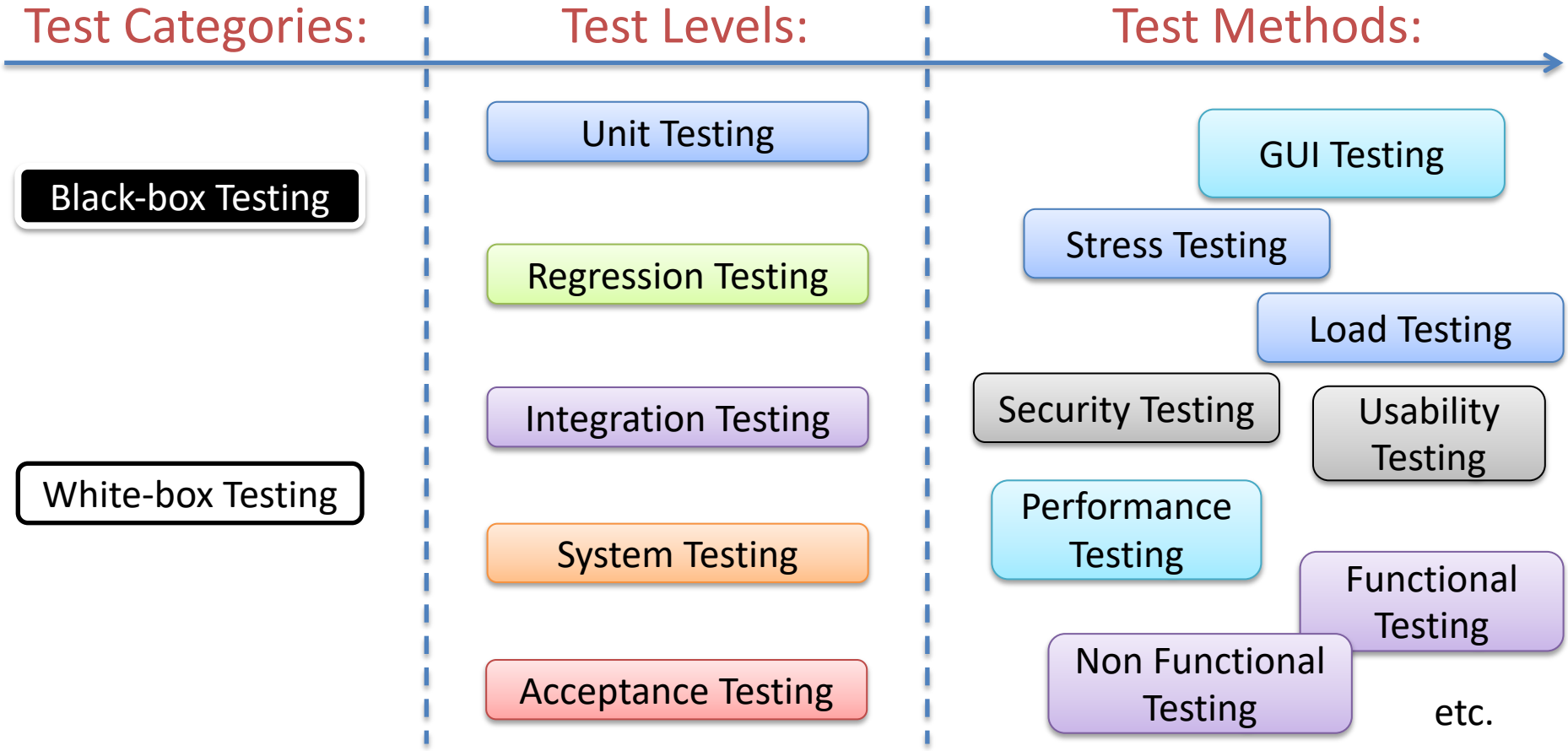
Regression testing is testing the system to check that changes have not "broken" previously working code. Both Manually & Automatically (Re-run Unit Tests)

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



80 – 20 Rule

- It takes 20% of the time to finish 80% of your application -> Prototype (80% finished) (The “fun” part, we stop here)
- It takes 80% of the time to finish the last 20% (minor adjustments, stability and performance improvements, bug fixing, etc.) (The “boring” part)
- 80% of the users only use 20% of the features
- 80% of performance improvements are found by optimizing 20% of the code
- 80% of the bugs are found in 20% of the code

http://swreflections.blogspot.no/2013/11/applying-8020-rule-in-software.html

Software Test Plan

Hans-Petter Halvorsen

Table of Contents

Software Test Plan (STP)

- Create a **Software Test Plan (STP)**
- The content in the STP may differ depending on the Project (what kind of software you are creating, the size, etc.), see examples on the next slides
- Upload the STP to Teams/Azure DevOps & your Web Site

See Next Slides for more details...

Create Software Test Plan (STP)

Team:_____

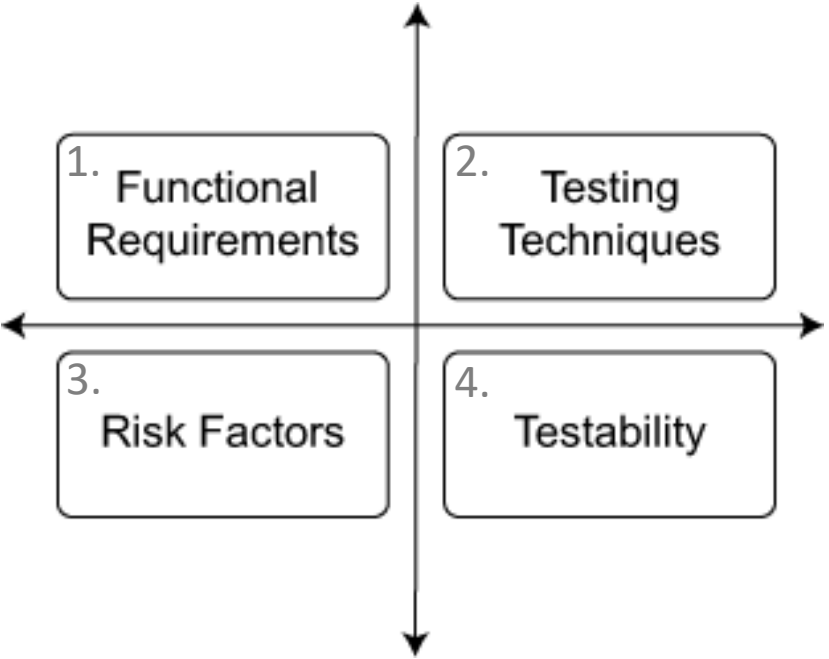
Create a Software Test Plan (STP) document

- ❑ Introduction
  - ❑ Test Software (Azure DevOps, ...)
- ❑ Test Resources
  - ❑ Test Personnel and Responsibilities. Test Manager
  - ❑ Test Environment and Test Hardware
- ❑ Overview of different Test Types
  - ❑ Validation Testing and Defect Testing
  - ❑ Unit Testing, Regression Testing, Integration Testing, System Testing, Acceptance Testing
- ❑ Test Strategies
  - ❑ What to test
  - ❑ How to test
  - ❑ When to test. Test Schedule
- ❑ Test Cases. Can, e.g., be Excel sheets
- ❑ Test Documentation - How shall tests be documented?

Test Planning

- To maximize the effectiveness of resources spent on testing, a systematic approach is required
- Testing should be well planned and organized
- Testing should be documented. Typically, the Customer requires that the Testing of the Software is well documented
- A Software Test Plan (STP) should be created

Software Test Plan – Key Factors



Software Test Plan Key Factors:

1. *Identify Key Functionalities*: The important functionalities that the product should have to ensure its success.
2. *Testing Techniques and Tools*: The different testing techniques and tools that could be leveraged for testing the various product functionalities.
3. *Risk Factors Approach*: The various risk factors which need to be considered and their impact to the product.
4. *Testability of Features*: The testability of the product and areas of the product which may not be testable.

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

Test Cases Example

If Test Cases Fails, report Bugs in DevOps

Tester: _____ , Date: _____

Test Case	OK	Failed	Expected Behavior	Description/Comments
The Login Procedure works			...	
User Data Saved in the Database			...	
Etc.			...	

The Foundation for the Test Cases are the Software Requirements (found in the SRS document)

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

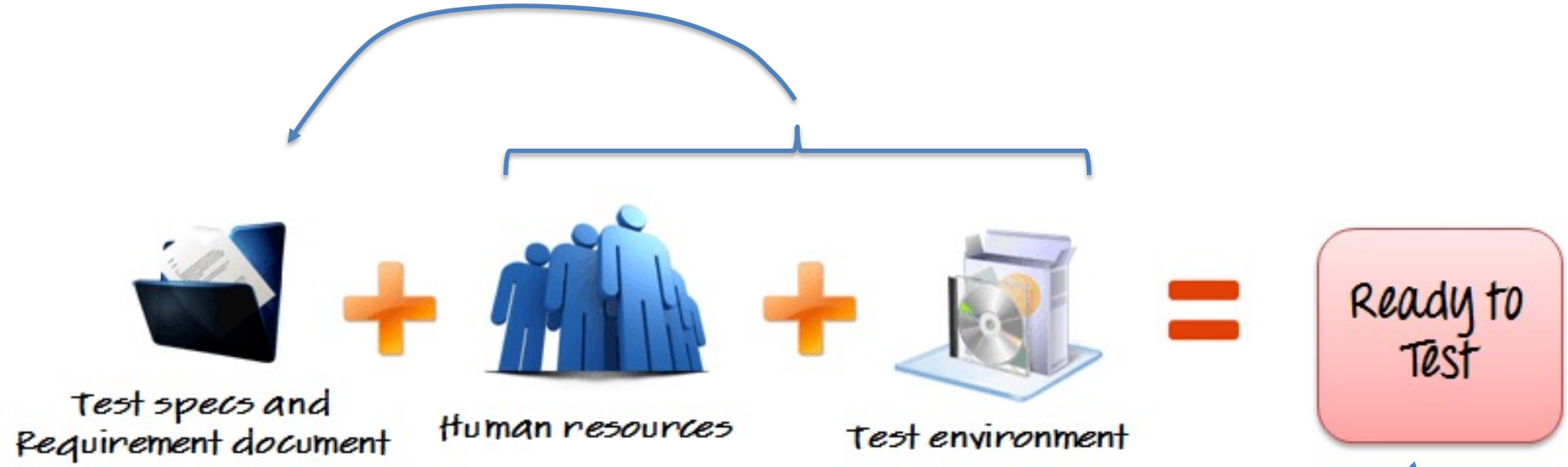
Test Cases Example

Examples of Fields in a Test case document:

- *Test Case Id*—A unique sequential number to identify each test case.
- *Short Description*—A brief description of the functionality tested in a few words.
- *Description*—A more detailed description of the functionality in a few sentences.
- *Pre-requisites*—The data that would be required to set up or the environment should be available.
- *Test Steps*—A set of logical steps to test a particular functionality of the application. The test steps could be written for positive as well as negative tests.
- *Test Data*—The data that are required to test the particular flow.
- *Status*—The pass/fail of the test case execution.
- *Remarks*—Any comments or information which the tester would like to share for future reference.

SN	SRS Ref No.	Test Case	Test Case ID	Test Condition	Expected Results	Actual Result	Status	Remarks
1	1.1, 1.2	Anonymous User	1-TC-1	Anonymous user tries to click on links provided on Login Page	User is able to access the login page containing text and links			
	1.1, 1.2	Anonymous Users	1-TC-2	Anonymous user tries to login with some username, password	Access to product is granted only with an "Authorizied Client" username and password			
	1.1, 1.2	Client User	1-TC-3	Client user tries to click on links provided on Login Page	User is able to access the login page containing text and links			
2	1.1, 1.2, 1.3	Client User	1-TC-4	Client user enters Approved username and Password	Upon enterning an approved username and password the user wil be granted access to the product. The user will be redirected to home page screen.			
	1.1, 1.2	Client User	1-TC-5	Clent user logs on to the website and click on the Home page Link.	Clicking on Homepage link wil redirect user loged on client Homepage			
	1.1, 1.2	Client User	1-TC-6	After login into the application, from any screen, user clicks on Home From global header	User will be redirected to Client Homepage			

These things need to be specified in the STP



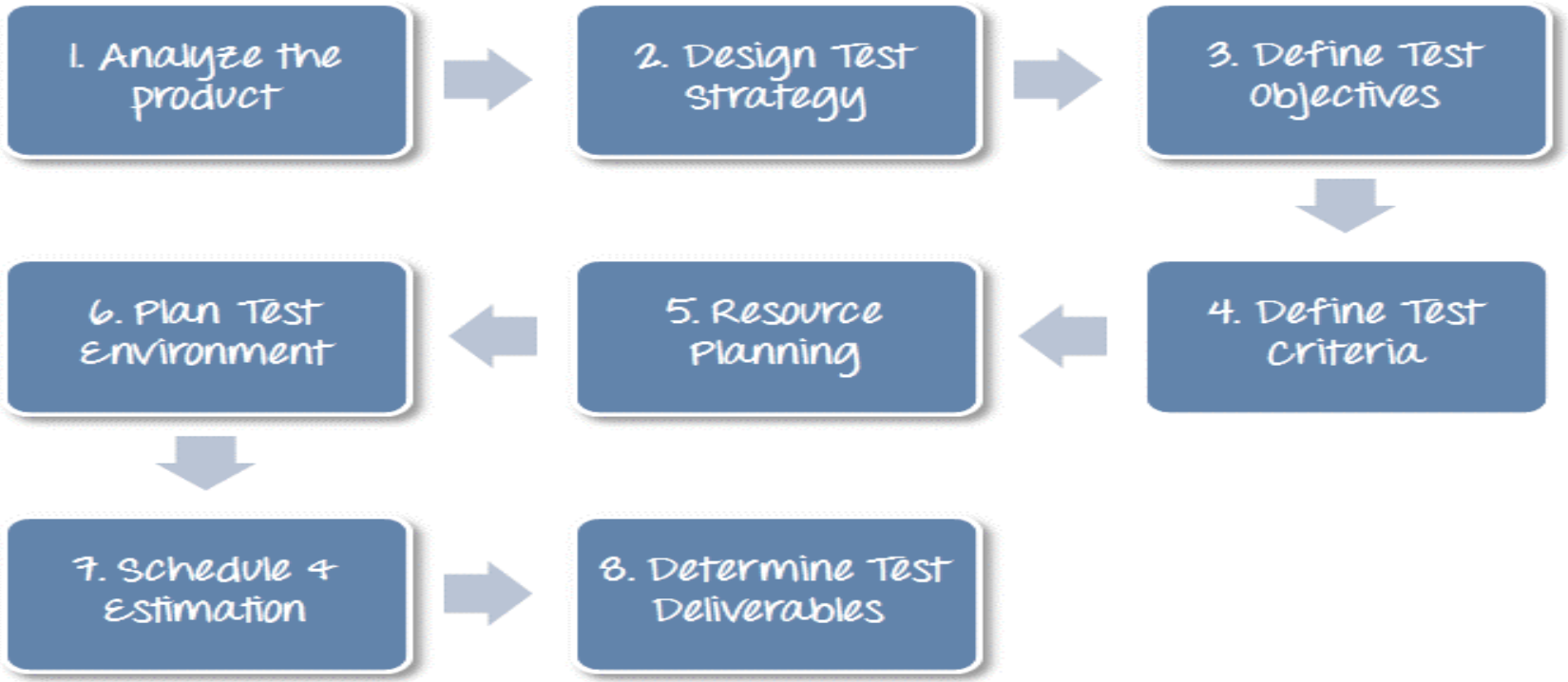
Software Test Plan (STP)

We will create a **Virtual Test Envionment** using VMware Player

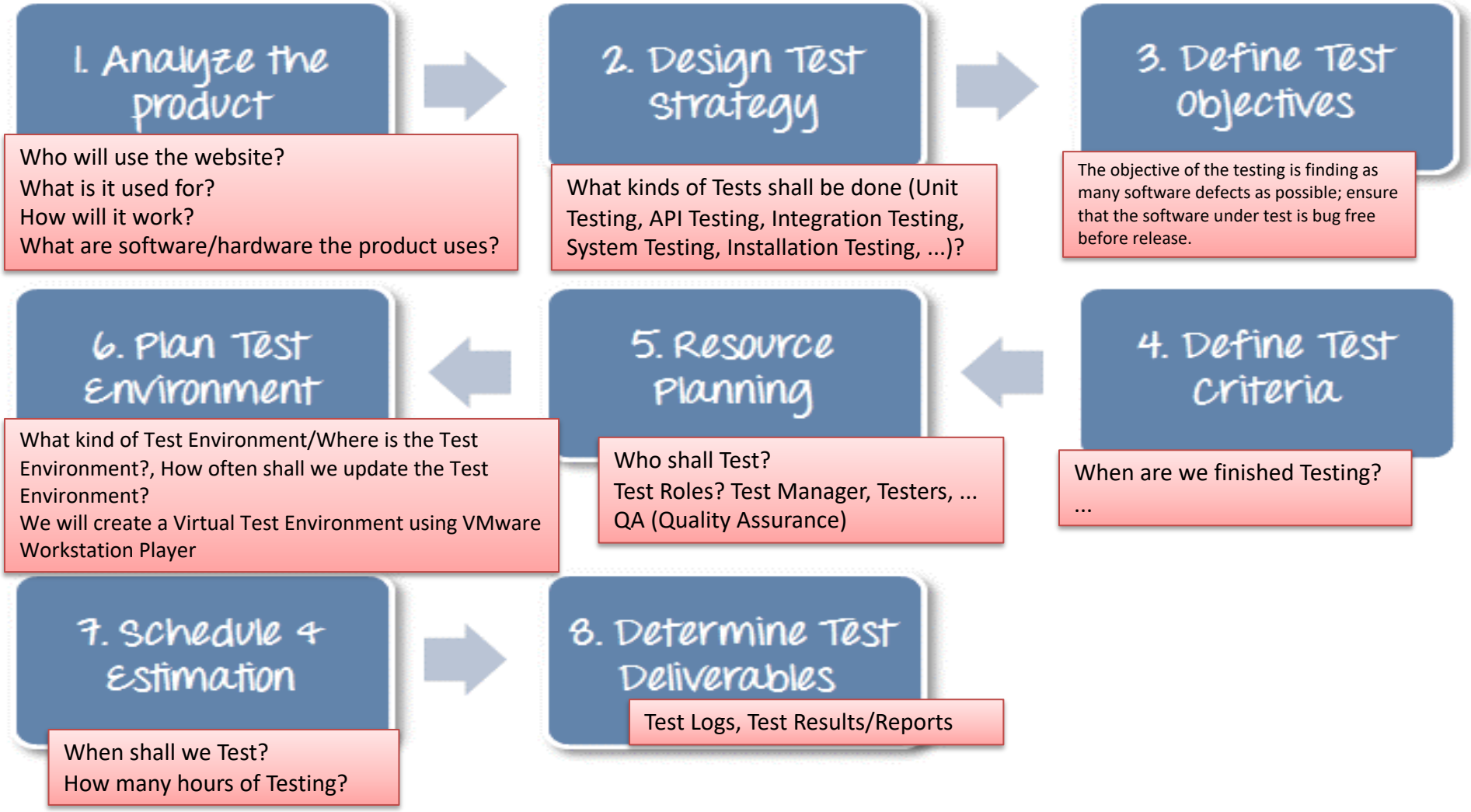
We start creating Test Environment on Friday and start Testing next week, Report Bugs in Azure DevOps, etc.

We start creating a STP and Virtual Test Environments this week

How to make a Test Plan



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


1. Analyze the product
Who will use the website?
What is it used for?
How will it work?
What are software/hardware the product uses?
2. Design Test Strategy
What kinds of Tests shall be done (Unit Testing, API Testing, Integration Testing, System Testing, Installation Testing, ...)?
3. Define Test Objectives
The objective of the testing is finding as many software defects as possible; ensure that the software under test is bug free before release.
4. Define Test Criteria
When are we finished Testing?
...
5. Resource Planning
Who shall Test?
Test Roles? Test Manager, Testers, ...
QA (Quality Assurance)
6. Plan Test Environment
What kind of Test Environment/Where is the Test Environment?, How often shall we update the Test Environment?
We will create a Virtual Test Environment using VMware Workstation Player
7. Schedule & Estimation
When shall we Test?
How many hours of Testing?
8. Determine Test Deliverables
Test Logs, Test Results/Reports

Appendix D in Essentials of Software Engineering

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 (...)

Essentials of Software Engineering, Frank Tsui; Orlando Karam; Barbara Bernal, 3 ed., Jones & Bartlett Learning

Test Plan Example #2

Reference: Software Testing Tutorial, tutorialspoint.com

See PDF document on Course schedule

- A Test Plan outlines the strategy that will be used to test an application, the resources that will be used, the test environment in which testing will be performed, the limitations of the testing and the schedule of testing activities.
- Typically the Quality Assurance Team Lead will be responsible for writing a Test Plan.
- A Test Plan will include the following:
  - Introduction to the Test Plan document
  - Assumptions when testing the application
  - List of test cases included in Testing the application
  - List of features to be tested
  - What sort of Approach to use when testing the software
  - List of Deliverables that need to be tested
  - The resources allocated for testing the application
  - Any Risks involved during the testing process
  - A Schedule of tasks and milestones as testing is started

Example #3

How to Create a Test Plan

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

Test Plan Template:
http://download.guru99.com/random_download/download_file.php?file=guru99/TestPlan.doc

Test Plan Example:
http://download.guru99.com/random_download/download_file.php?file=guru99/Test_Plan_Guru99.pdf

Example #4

Another Test Plan Example

separately tested but only when cleanly designed. As will be seen in Chapter 26, on unit testing, OO testing includes method, class, and package testing.

25.6 DOCUMENTING TESTS

It requires significant time to decide what to test, how to test, when to do so, and with what data. In addition, test results must be analyzed to determine what defects they uncovered. We therefore treat each test as an item of value. Test procedures, test data, and test records are maintained; tests are reused or modified where possible. Examples of test documentation can be found in Chapters 26 and 27.

25.7 TEST PLANNING

To maximize the effectiveness of resources spent on testing, a systematic approach is required and a plan is devised. Recall that the goal is to detect as many errors as serious a level as possible with the resources available. Typical planning steps are shown in Figure 25.8 and elaborated in the rest of this section.

25.7.1 Organize "Unit" vs. Non-Unit Tests

The limits of what constitutes a "unit" have to be defined by the development team. For example, do they include the testing of packages, or is this to be considered another type of testing?

1. Define "units" vs. non-units for testing
2. Determine what types of testing will be performed
3. Determine extent
   - Do not just "test until time expires"
   - Prioritize, so that important tests are definitely performed
4. Document
   - Individual's personal document set included?
   - How/when to incorporate all types of testing?
   - How/when to incorporate in formal documents?
   - How/when to use tools/test utilities?
5. Determine input sources
6. Decide who will test
   - Individual engineer responsible for some (units)?
   - How/when inspected by QA?
   - How/when designed and performed by third parties?
7. Estimate resources
   - Use historical data if available
8. Identify metrics to be collected
   - Define, gather, use
   - For example, time, defect count, type, and source

Figure 25.8 A plan for testing

- When the system has not been able to find another defect in 5 (10? 30? 100?) minutes of testing
- When all nominal, boundary and out-of-bounds test examples show no defect
- When a given checklist of test types has been completed
- After completing a series of targeted coverage (e.g., branch coverage for unit testing)
- When testing runs out of its scheduled time

Figure 25.9 Stopping criteria for testing

For object-oriented development projects, a common sequence of unit testing is to test the methods of each class, then the classes of each package, and then the package as a whole. If we were building a framework, we would test the classes in each framework package first and then move on to the application packages, because the latter depend on the former. Once the "units" and non-unit tests have been identified, they must be organized and saved in a systematic manner.

25.7.2 Determine the Extent of Testing

Since it is impossible to test for every possible situation, the *extent* of testing should be considered and defined in advance. For example, if a banking application consists of withdrawals, deposits, and queries, unit testing could specify that every method should be tested with an equal amount of legal, boundary, and illegal data; or perhaps, due to their sensitivity, withdrawal and deposit methods are tested three times as extensively as query methods, and so on. Test cases are selected both from normal expected operation, as well as from those judged most likely to fail. *Stopping criteria* are established in advance; these are concrete conditions upon which testing stops. Examples are listed in Figure 25.9.

25.7.3 Decide How Tests Will Be Documented

Test documentation consists of test procedures, input data, the code that executes the test, output data, known issues that cannot be attended to yet, and efficiency data. Test drivers and utilities are used to execute unit tests, and these are documented for future use. JUnit is an example of a unit test utility (described in more detail in Chapter 26). JUnit-like and various professional test utilities help developers to retain test documentation. JUnit classes, in particular, tend to be maintained along with the application.

25.7.4 Decide How and Where to Get Test Input

Applications are developed to solve problems in a specific area, and there is often a set of test data special to the application. Examples are as follows:

- Standard test stock market data for a brokerage application
- Standard test chemical reactions for a chemical engineering application
- Standard FDA procedures for the pharmaceutical industry
- Standard test input for a compiler
- Output from previous versions of the application

The procurement process and use of such domain-specific test input must be planned.

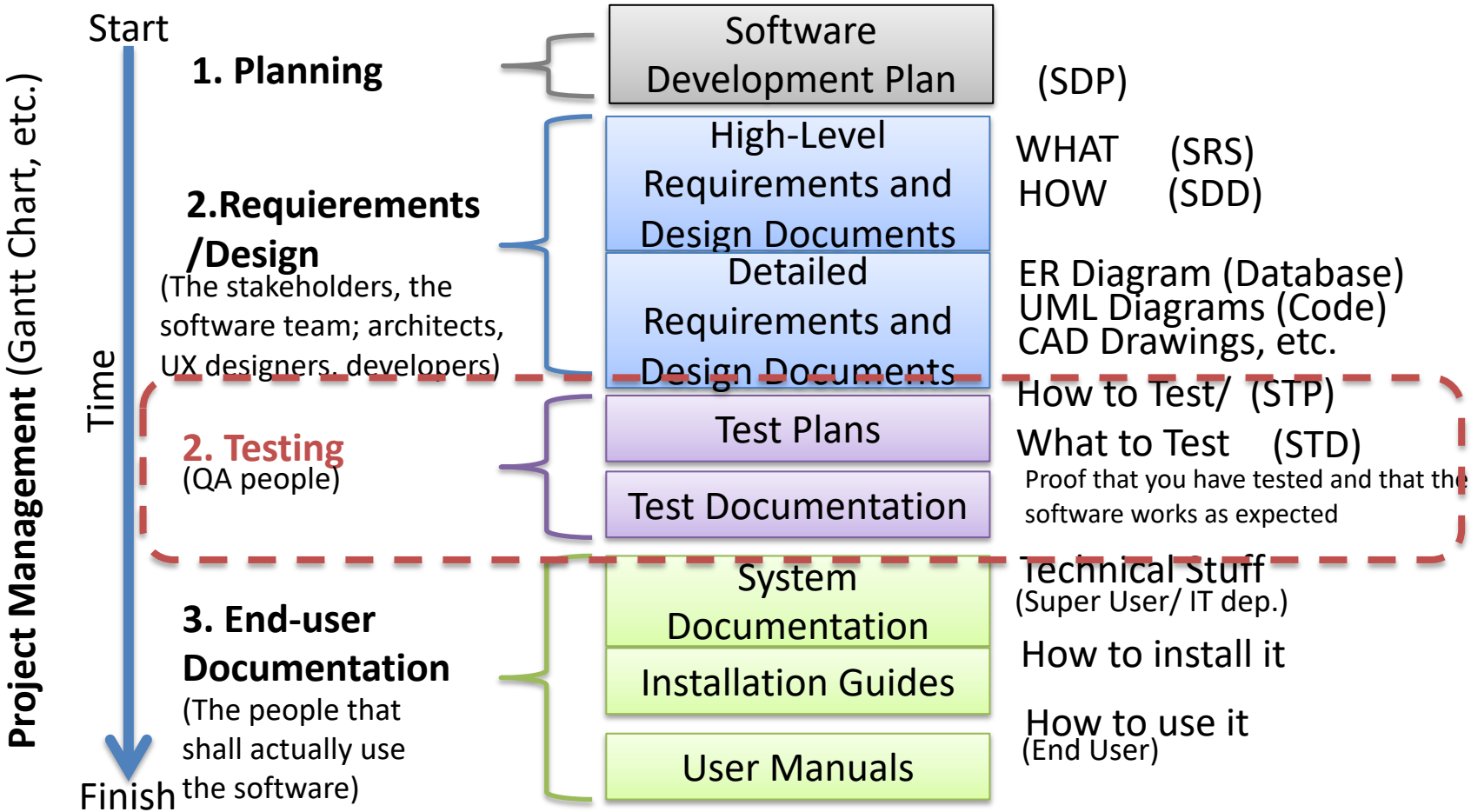
E. J. Braude and M. E.Bernstein, *Software Engineering: Modern Approaches*, 2 ed.: Wiley, 2011. (Ch. 25)

Test Plan Example

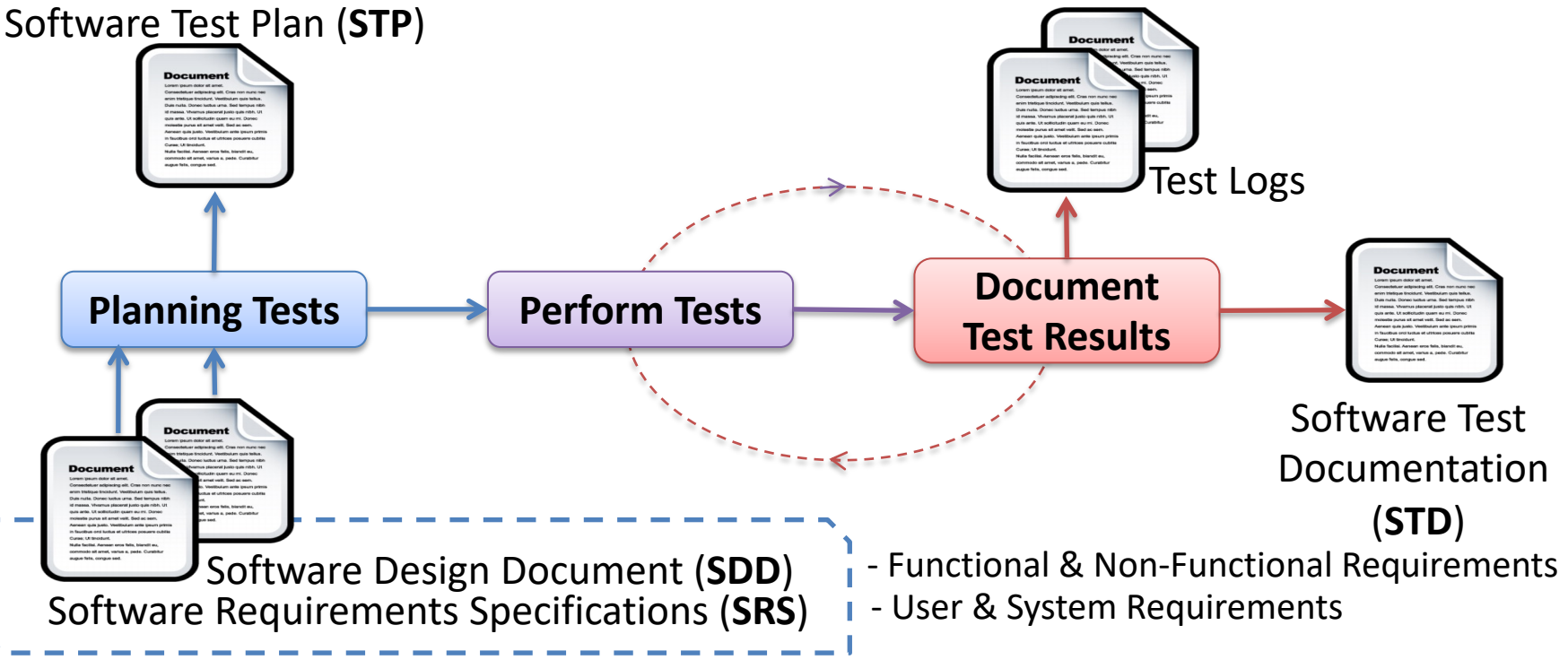
Example #5

- **1. Sample Test Plan Structure: Introduction**
  - 1.1 Background
  - 1.2 References
  - 1.3 Development Methodology
  - 1.4 Change Control Procedure
  - 1.5 Test Assumptions
- **2. Scope**
  - 2.1 Technical Overview of the Application
  - 2.2 Technical components or architecture diagram
  - 2.3 Business Overview of the application
  - 2.4 Business or Data Model Diagram
  - 2.5 External Interfaces
  - 2.6 Testability
  - 2.7 Out of Scope
- **3. Test Strategy**
  - 3.1 Features to be tested
  - 3.2 Types of testing
  - 3.3 Testing Approach
- **4. Release**
  - 4.1 Activity Guidelines
  - 4.2 Defect Tracker Setup
  - 4.3 Test case pass/fail criteria
  - 4.4 Test suspension criteria
  - 4.5 Test resumption requirements
  - 4.6 UAT Release criteria
- **5. Critical Dates**

Typical Software Documentation



Test Documentation



These documents will be the foundation for all Testing

Test Environment

Hans-Petter Halvorsen

Table of Contents

Test Environment

Note! Make sure you have enough free space on your harddrive!

- Prepare a Virtual Test Environment using **VMware Workstation Player** (Virtual Machine, VM)
- Install Windows (from Microsoft Imagine)
- Install SQL Server
- Install your Software on the Virtual Machine
- Make it ready for Testing

Note! Everyone on the Team should do this Exercise

See Next Slides for more details...

Create Virtual Test Environment

- ❑ Install **VMware Workstation Player**

You need +20Gb free space on your hard drive

Name:_____

- ❑ Install **Windows 10** (from .iso file, ~3Gb)
- ❑ Install **VMware Tools** in the VM
- ❑ Install **SQL Server** in the VM
- ❑ Activate Web Server – Internet Information Services (**IIS**) and ASP.NET
- ❑ **Backup** (Make a copy of the Folder) the Virtual Machine (VM). In that way you have a clean Test Environment you can use several times.

The first student submitting this form will receive a prize!

- ❑ **Install your Software in the VM and make it ready for Testing**
  - ❑ Create/Install your Database from a **SQL Script** (You should have one SQL Script that installs everything, such as Tables, Views, Stored Procedures, etc.)
  - ❑ Install **Desktop App** (if any), i.e., copy .exe file, etc.
  - ❑ Install **Web App**, copy files (web files, dll and other necessary files) and **deploy to Web Server** (IIS)
  - ❑ Make a Copy of your VM (Memory Stick/Hard Drive)

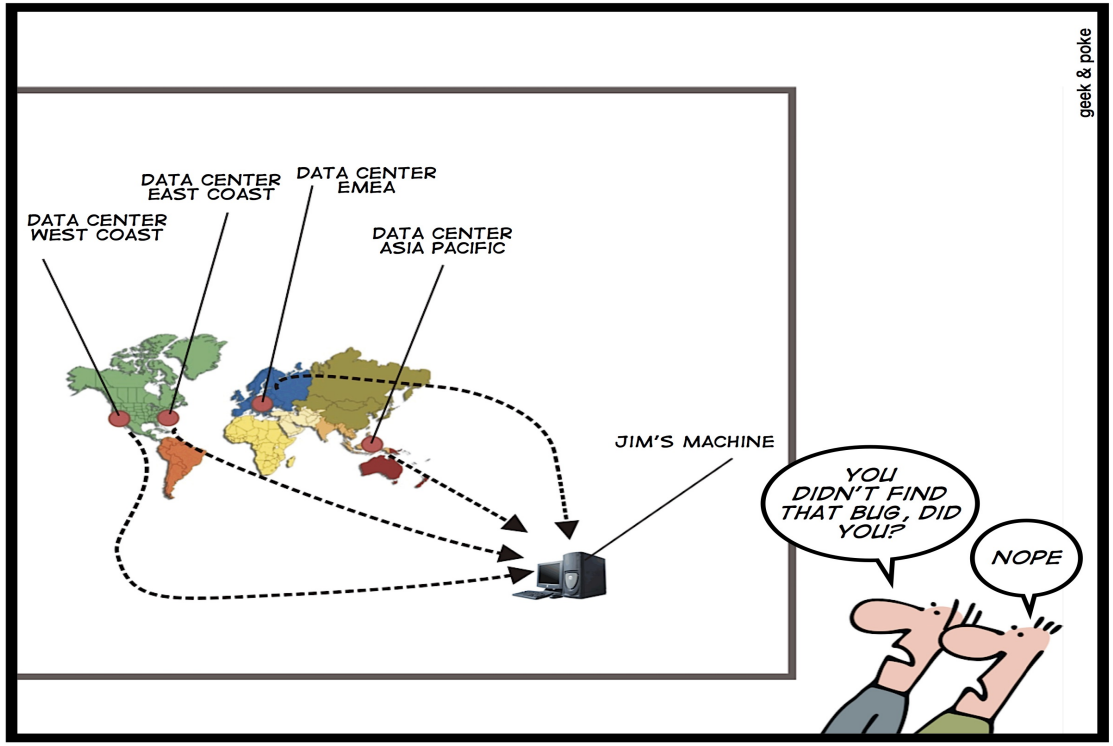
Note! Visual Studio shall not be installed in the Test Environment

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 don't 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”

SIMPLY EXPLAINED



"ON MY MACHINE IT WORKS"

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

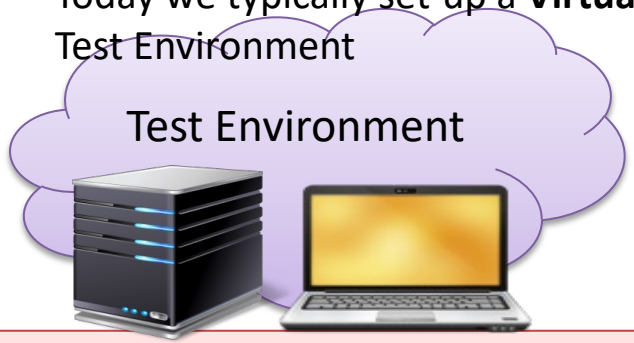
- Everything works on the Developer Computer
- The Customers Database is not the same as yours
- The Customer may not use the same OS
- The Customer may not use the same Web Browser
- The Customer do not have Visual Studio, SQL Server, etc. on their Personal Computer
- Etc.

=> Test Environment is needed!

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

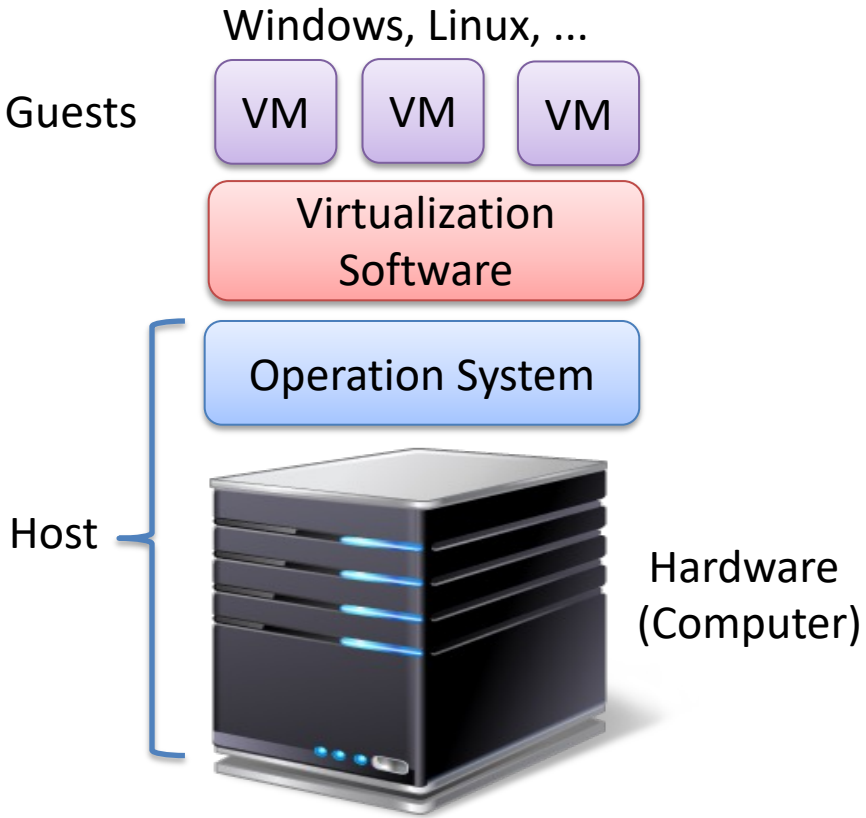
A Clean PC/Server (or a network with PCs and Servers) where you install and test your Software. Today we typically set-up a **Virtual** Test Environment

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

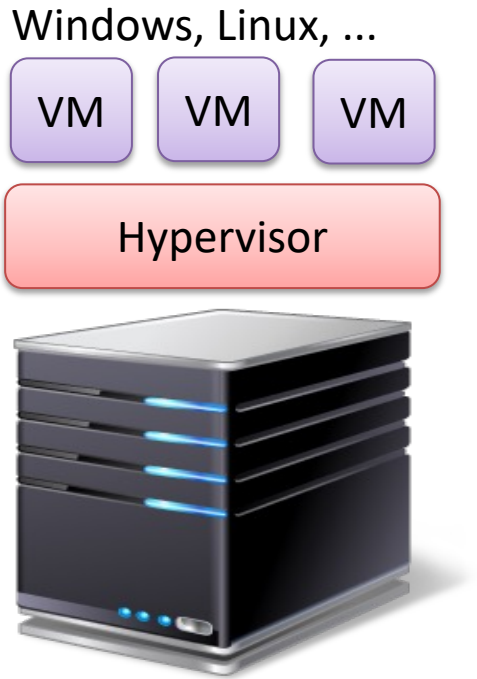


Programming environments such as Visual Studio, etc. should not be installed in this environment. You need to create .exe files etc. in order to make your software run.

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 (part of Windows)
- VirtualBox
- etc.

VMware Workstation Player

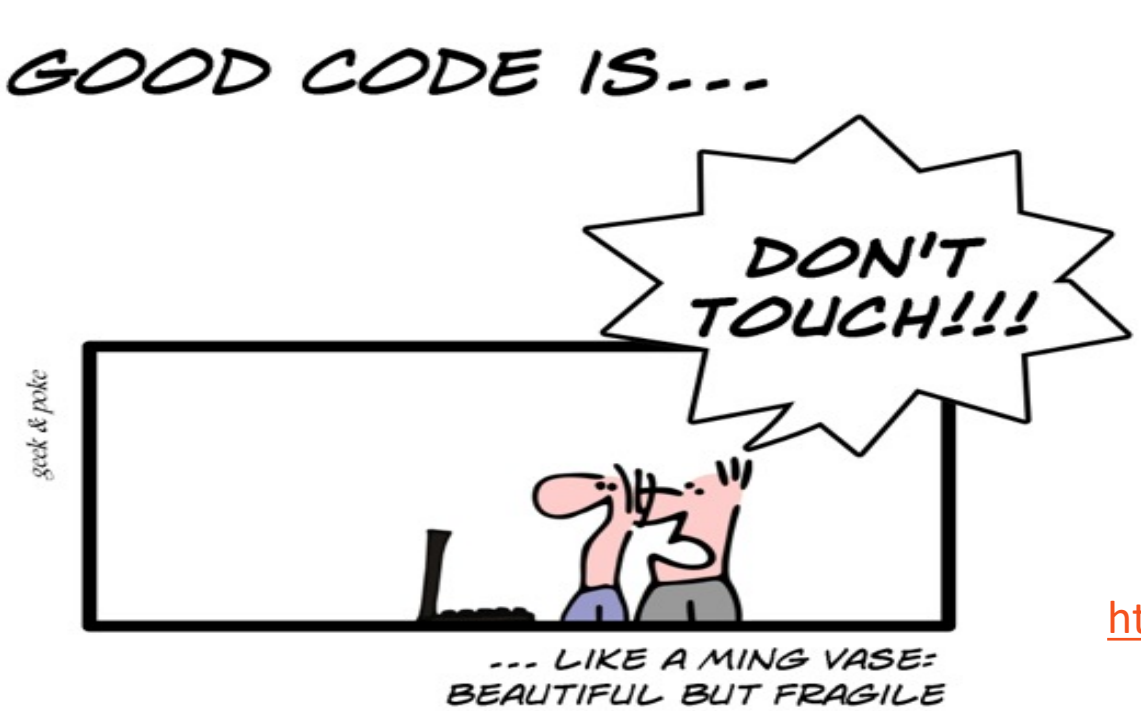
VMware Workstation Player is for personal use on your own PC. VMware Player is free of charge for personal non commercial use.

VMware is a company that has been specializing within virtualization software.

http://www.vmware.com

Test Environment - Summary

- It is important to test your software outside the Development Environment
- **To make the software available** for test personnel (nonprogrammers), the company leaders, customers, those who are creating user documentation, sales department, etc. None of these have programming experience or have Visual Studio, etc. installed
- It is important that the Customers, Testers, etc. have access to and can test the software in good time before release and deployment to Production Environment



http://geek-and-poke.com

"If your code works, but you don't know why – Then it does not work, you just don't know it yet"

Customer Perspective

- Remember – It is your Customers that are going to use your Software (and pay for it)!
- The Customer needs to be involved in the Requirements, User Experience and Testing of the Software!
- If the Customer cannot use the software, then the software becomes worthless

Hans-Petter Halvorsen

University of South-Eastern Norway

www.usn.no

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

Web: https://www.halvorsen.blog