Software Development
A Practical Approach!

Hans-Petter Halvorsen

Planning

Deployment

Maintenance

Testing

The Software Development Lifecycle (SDLC)

Requirements Analysis

Implementation

Design

https://www.halvorsen.blog

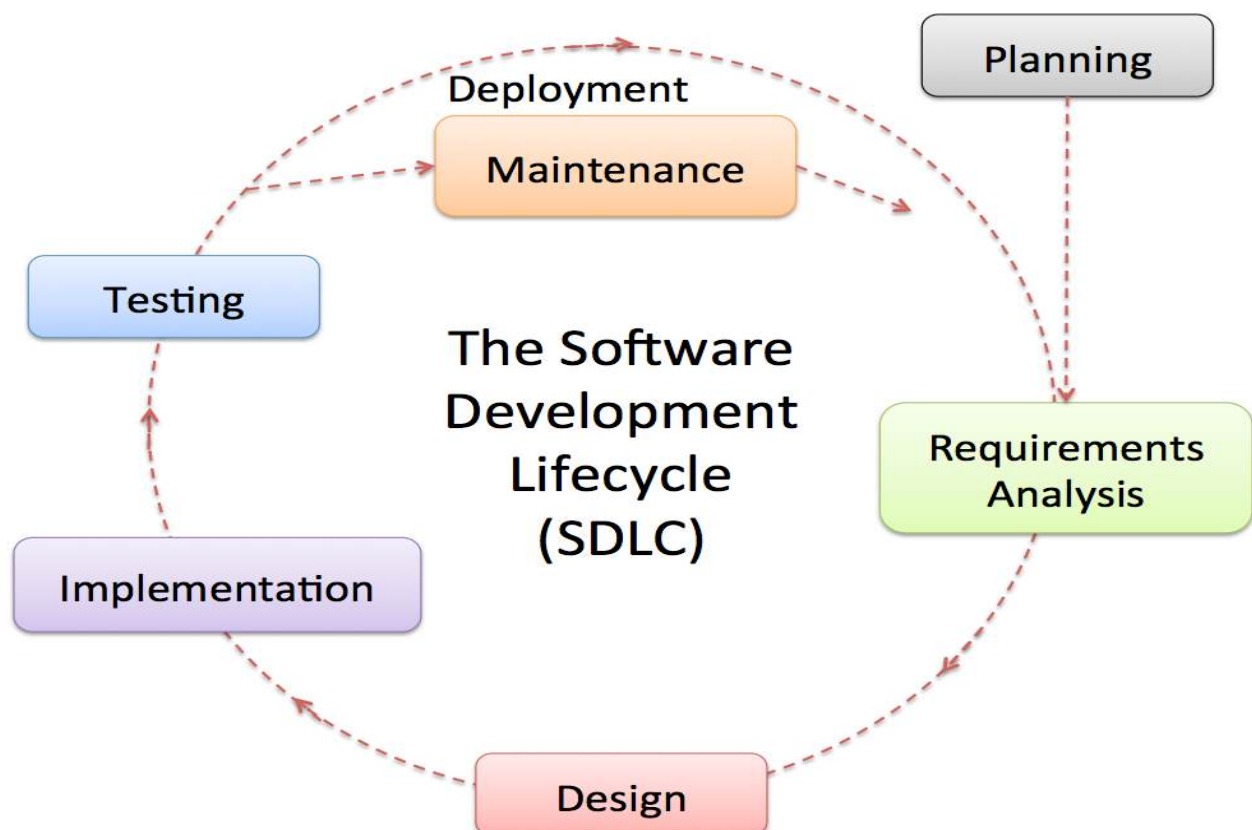# Software Development

A Practical Approach!

Hans-Petter Halvorsen

Planning

Deployment

Maintenance

Testing

The Software Development Lifecycle (SDLC)

Requirements Analysis

Implementation

Design

Software Development

A Practical Approach!

Hans-Petter Halvorsen

Preface

The main goal with this document:

- To give you an overview of what software engineering is
- To take you beyond programming to engineering software

What is Software Development? It is a complex process to develop modern and professional software today. This document tries to give a brief overview of Software Development.

This document tries to focus on a practical approach regarding Software Development.

So why do we need System Engineering? Here are some key factors:

- Understand Customer Requirements
  - What does the customer needs (because they may not know it!)
  - Transform Customer requirements into working software
- Planning
  - How do we reach our goals?
  - Will we finish within deadline?
  - Resources
  - What can go wrong?
- Implementation
  - What kind of platforms and architecture should be used?
  - Split your work into manageable pieces

- Quality and Performance
    - Make sure the software fulfills the customers' needs

We will learn how to build good (i.e., high quality) software, which includes:

- Requirements Specification
- Technical Design
- Good User Experience (UX)
- Improved Code Quality and Implementation
- Testing
- System Documentation
- User Documentation
- etc.

You will find additional resources on this web page:

http://www.halvorsen.blog/documents/programming/software_engineering/

Information about the author:

Hans-Petter Halvorsen

The author currently works at the University of South-Eastern Norway. The author has been working with Software Engineering for more than 20 years.

For more information, visit my web site:

https://halvorsen.blog

Table of Contents

Part 1 : Introduction

In this part, we discuss what software development is with some examples. We also give a brief overview of the software history.

1 Introduction

What is Software Development? It is a complex process to develop modern and professional software today. This document tries to give a brief overview of Software Development. Normally we use the terms System Engineering or Software Engineering.

Software Engineering: https://youtu.be/f4dpWVzpvUE

System Engineering: The process of analyzing and designing an entire system, including the hardware and the software.

Software Engineering: The discipline for creating software applications. A systematic approach to the design, development, testing, and maintenance of software.

A lot of systems today have a mix of hardware and software that is tightly integrated, like modern smartphones, tablets, etc. Creating such systems involves a lot of different disciplines.

Software is any set of machine-readable instructions that directs a computer's processor to perform specific operations. The term is used to contrast with computer hardware, the physical objects (processors and related devices) that carry out the instructions. Computer hardware and software require each other, and neither can be realistically used without the other, see Figure 1-1.

Figure 1-1: Hardware and Software working together

In Figure 1-2 we see a typical network and infrastructure that the software relies on.

Figure 1-2: Typical Network and Infrastructure in Software Development

Figure 1-3 we see the complexity of software development and different components that are involved.
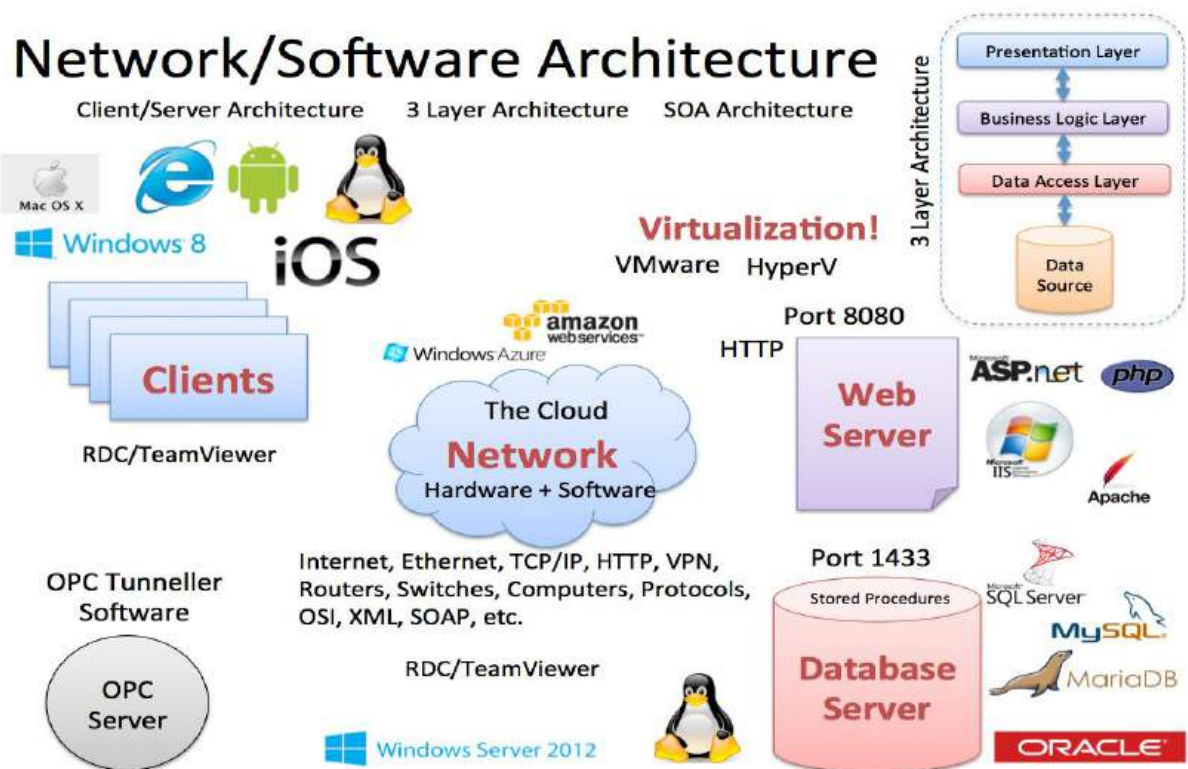
Figure 1-3: The Complexity of modern Software

In Figure 1-4 we see the different phases involved in the Software Development Lifecycle (SDLC).
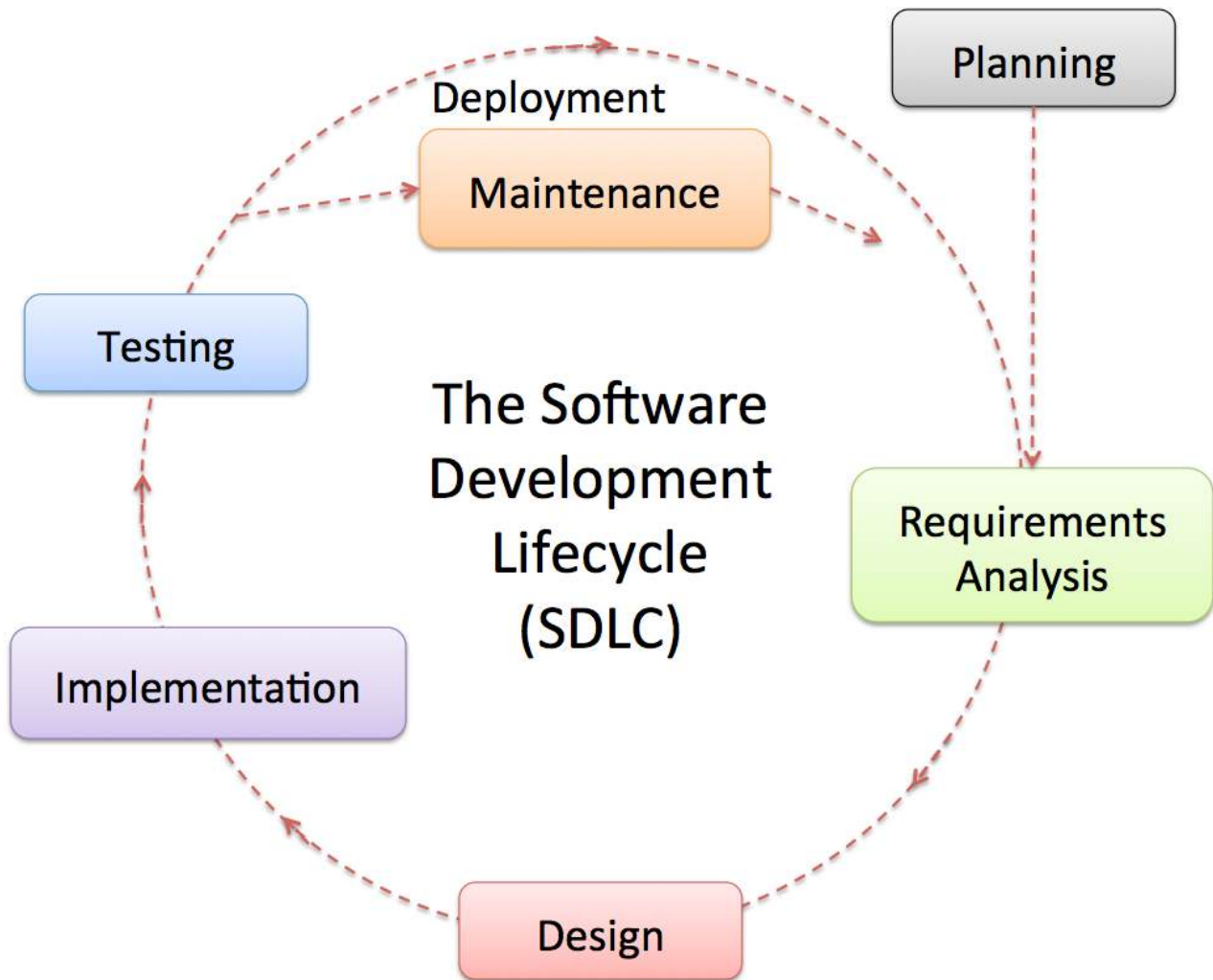
Figure 1-4: The Software Development Lifecycle

The main parts or phases in the software development process are:

- Planning
- Requirements Analysis
- Design
- Implementation
- Testing
- Deployment and Maintenance

In Figure 1-5 we see examples of some of the different activities involved in the different phases of software development.
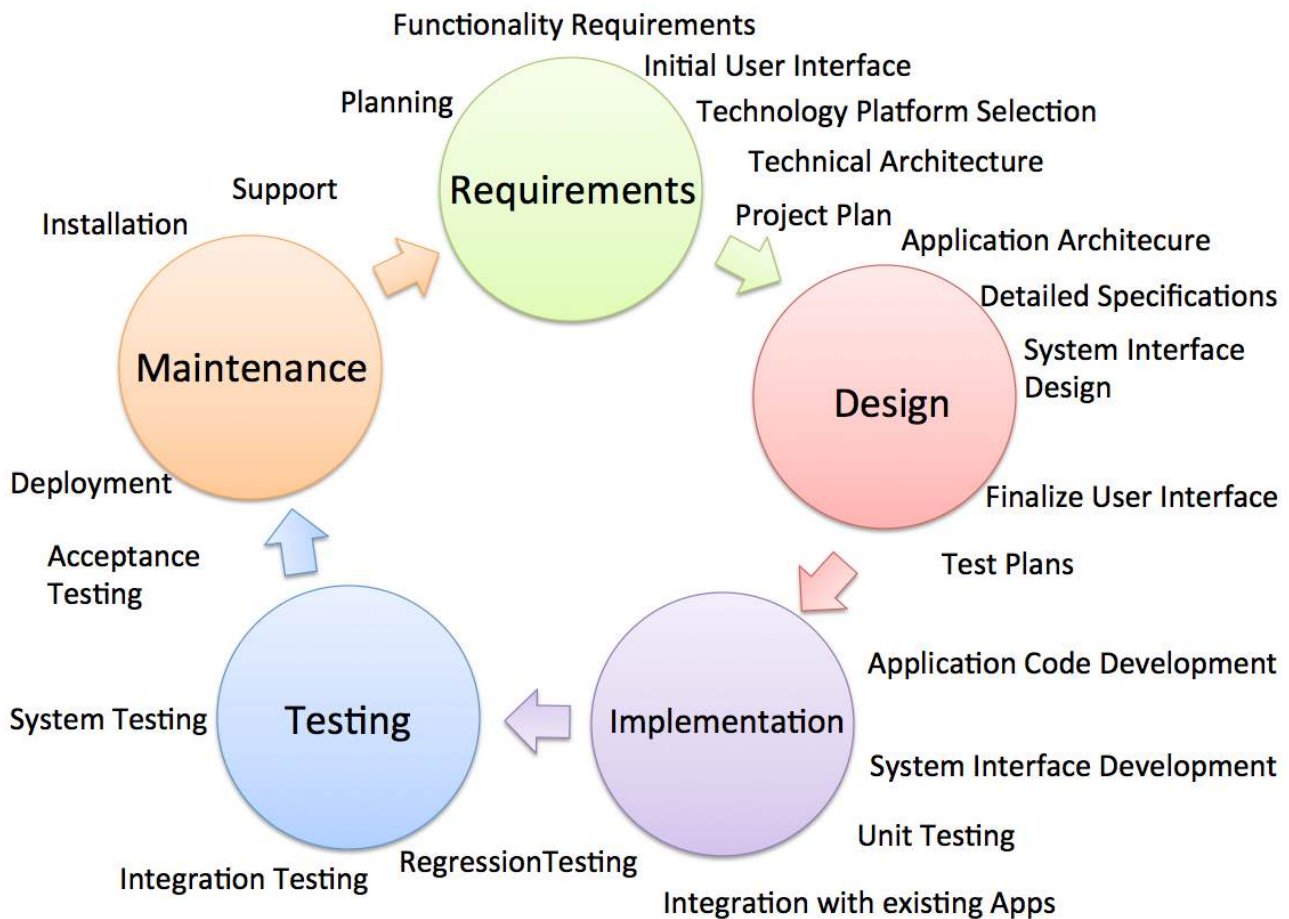
Figure 1-5: Activities involved in the different Software Phases

As you see, software development involves a lot of phases, they are executed by different disciplines and different people. We will discuss and explain all these things later in this document.
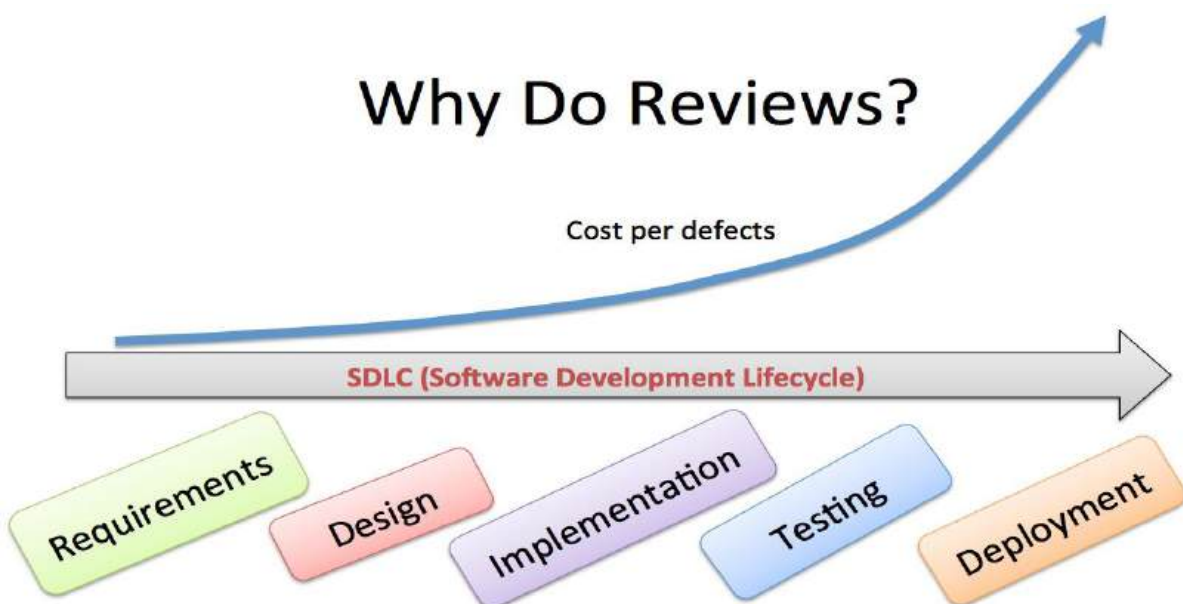
Figure 1-6: Have Reviews at all levels in the Development Cycle

1.1 Background

Software and software systems are getting more and more complex, so it is important to have the necessary "tools" in your "toolbox" to be able to create and maintain your software.

Software Development is a complex process, and it may involve a lot of money and a lot of people. Here are some examples:

- Windows 7: A Team with 1000 Developers created Windows 7

- Number of Code Lines: Real systems may have millions of code lines

- Big money: 100+ million Development Projects

- Combination of Hardware and Software: Most of the projects involve both hardware and software and integration between them.

- iPhone 1: Development period 2004-2007, 1000 Apple employees worked with the device, Estimated cost: $150 mill.

All this need structure! - Software Engineering is the Answer!

[http://geek-and-poke.com]

Project Planning and Management is important in Software Development, and we can use different approaches to deal with Software Development, which we will cover in this document.

Especially so-called Agile approaches, such as Scrum, have become very popular today.

1.2 Topics

Below we list the different topics covered in this document. The Software Development Life Cycle, shorted SDLC, involves distinct phases, such as:

- Market research
- Gathering requirements for the proposed business solution
- Analyzing the problem
- Create a plan or design for the software-based solution
- Implementation (coding) of the software
- Documentation in different ways
- Testing the software
- Deployment and Installation
- Maintenance and bug fixing
- Marketing

See also Figure 1-7 for topics involved in software development.

There are different approaches (Software Development Processes) that deal with these phases, such as:

- **Waterfall model**
- V-model
- **Agile Software Development (such as Scrum, XP, etc.)**
- Spiral model
- Rational Unified Process (RUP)
- etc.

We will learn more about these development processes later in this document.

Figure 1-7: Different Topics involved in Software Development

Software Development also involves separate roles, which are organized in different teams (Figure 1-8). Typical roles are:

- Project Manager
- System Architect
- UX Designer
- Programmer, System Developer
- Tester
- Customer

It is crucial that the separate roles and teams can work together and collaborate.

The Programmer or System Engineer must deal with the fact that there exists hundreds of different Programming Languages. Each language has pros and cons, so it is important to find out which programming language is best suited in each situation.

Figure 1-8: Distinct Roles involved in Software Development

In this document, we will learn how to build good (i.e., high quality) software, which includes:

- Requirements Specification
- Technical Design
- User Experience (UX)
- Improved Code Quality and Implementation
- Testing
- System Documentation
- User Documentation
- etc.

1.3 Tools

To create great software, as a software engineer you need a toolbox with proper tools, otherwise you will not succeed in your job (see Figure 1-9).

Your Toolbox

- PC
- Programming Language
- IDE (Integrated Development Environment)
- Frameworks
- SCC Tool (Source Code Control)
- ALM Tool (Application Lifecycle Management)
- Knowledge about Software Engineering

You cannot do a good job as a software developer without some proper tools!

Figure 1-9: The Toolbox of a Software Engineer

When working with software development it is important to have good tools. The developer needs of course to use a programming language and proper IDE (Integrated Development Environment). In addition, a so-called ALM Tool should be used. ALM is short for Application Lifecycle Management. An ALM tool typically facilitates and integrate things like:

- Requirements Management
- Architecture
- Coding
- Source Code Control (SCC)
- Testing
- Bug Tracking
- Release Management
- etc.

There exist a lot of such tools, e.g., Azure DevOps, Jira, etc.

We will take a closer look at Azure DevOps (or the online version of Azure DevOps: Azure DevOps Services) in this document. Azure DevOps from Microsoft is tightly integrated with Visual Studio.

Typically, you need to share the code with other developers or testers in your team or other teams, so it is crucial that you have tools that can be used to share your code, that makes sure that

old versions of your code will be stored, and can be restored, etc. Such a system is called a Source Code Control (SCC) system.

Your software will also contain a lot of bugs that need to be found, tracked, and fixed, etc. To handle that we need a so-called Bug Tracking system.

In Figure 1-10 we see some of the bug tracking functionality in Azure DevOps.

Figure 1-10: Bug Tracking System

In Figure 1-11 we see a typical software project with different platforms and frameworks involved.
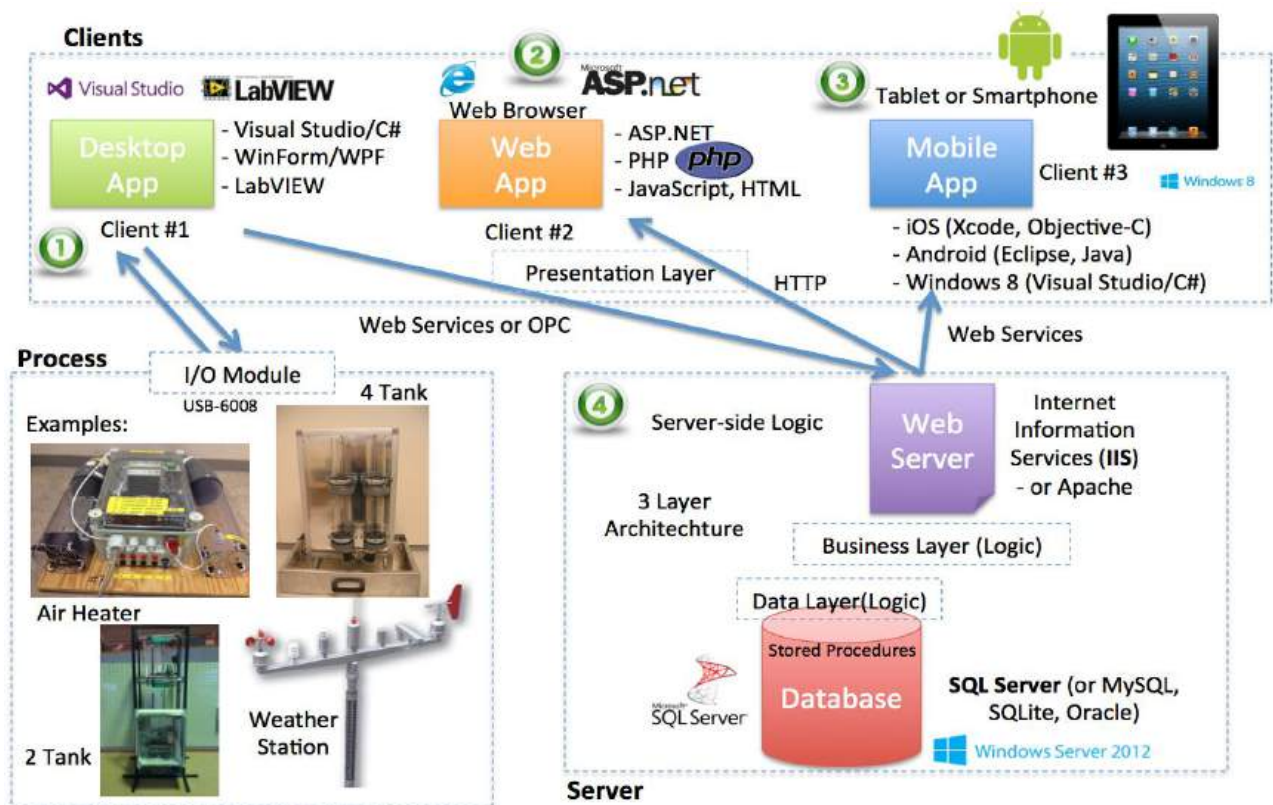
Figure 1-11: Typical Software Project with different Platforms and Frameworks involved

Typically, your software needs to be installed and be running on different devices, such as PCs, tablets, smartphones, etc. You also need to store the data, typically in a database, such as Microsoft SQL Server, MySQL, etc.

All these devices and data also need to communicate with each other over a network, either an internal network (LAN, Local Area Network) or over Internet (WAN, Wide Area Network).

All these things make it very complicated to develop, test, deploy and install such systems. That's the reality for a modern software developer.

2 Software History

2.1 Introduction

Computer and software history goes back to the beginning of the 1900 century. IBM was established in 1911, Hewlet Packard (HP) was established in 1939, the transistor came in 1947, the first Microprocessor came in 1972, etc. But "personal computing" started in 1981 with MS-DOS and the IBM PC. In 1984 came the famous Macintosh from Apple. Windows 1.0 was released in 1985.

They found a bug (a moth) inside a computer in 1947 that made the program not behave as expected. This was the "first" real bug.

In the 80s and 90s we saw the beginning of the personal computer era that started with Mac computers from Apple (Macintosh, 1984) and IBM computers from IBM (or IBM compatible computers from other vendors) with MS-DOS and later Windows installed (Figure 2-1).

Figure 2-1: The Beginning of Personal Computing

World Wide Web (WWW) was established in 1991. The first Web Browser, as we know it today, came in 1994 (Netscape). Google was established as late as 1998. Facebook was first invented in 2004. The first smartphone was released in 2007 (iPhone).

In Figure 2-2 we see some of the people that have founded and shaped the modern software industry.

Figure 2-2: Pioneers of modern Software Industry

The companies that they created still dominate the software industry today. Some of the people are still active within these companies today, either as CEOs or members of the board.

Some of the largest software companies today are:

- **Microsoft** (established 1975), Bill Gates, Paul Allen

    - MS DOS (1981), Office, Windows (1985), …

    - Employees (2012): 94.000, Revenue (2012): $74 bill.

- **Apple** (Software and Hardware) (established 1976), Steve Jobs, Steve Wozniak

    - Macintosh (1984), iPhone (1097), iPad (2012), iOS

    - Employees (2012): 72.800, Revenue (2012): $158 bill.

- **Google** (established 1986), Larry Page, Sergey Brin

- **Facebook** (established 2004), Mark Zuckerberg

- More than 1 bill. users

In addition, we can mention companies like IBM, Oracle, Samsung (more hardware than software), Amazon, SAP, Adobe, Symantec, VMware, etc.

2.2 Software Trends

The software industry has changed a lot since the 80s, and it is still changing very quickly. Figure 2-3 gives an overview of some important trends in the software industry today and tomorrow.
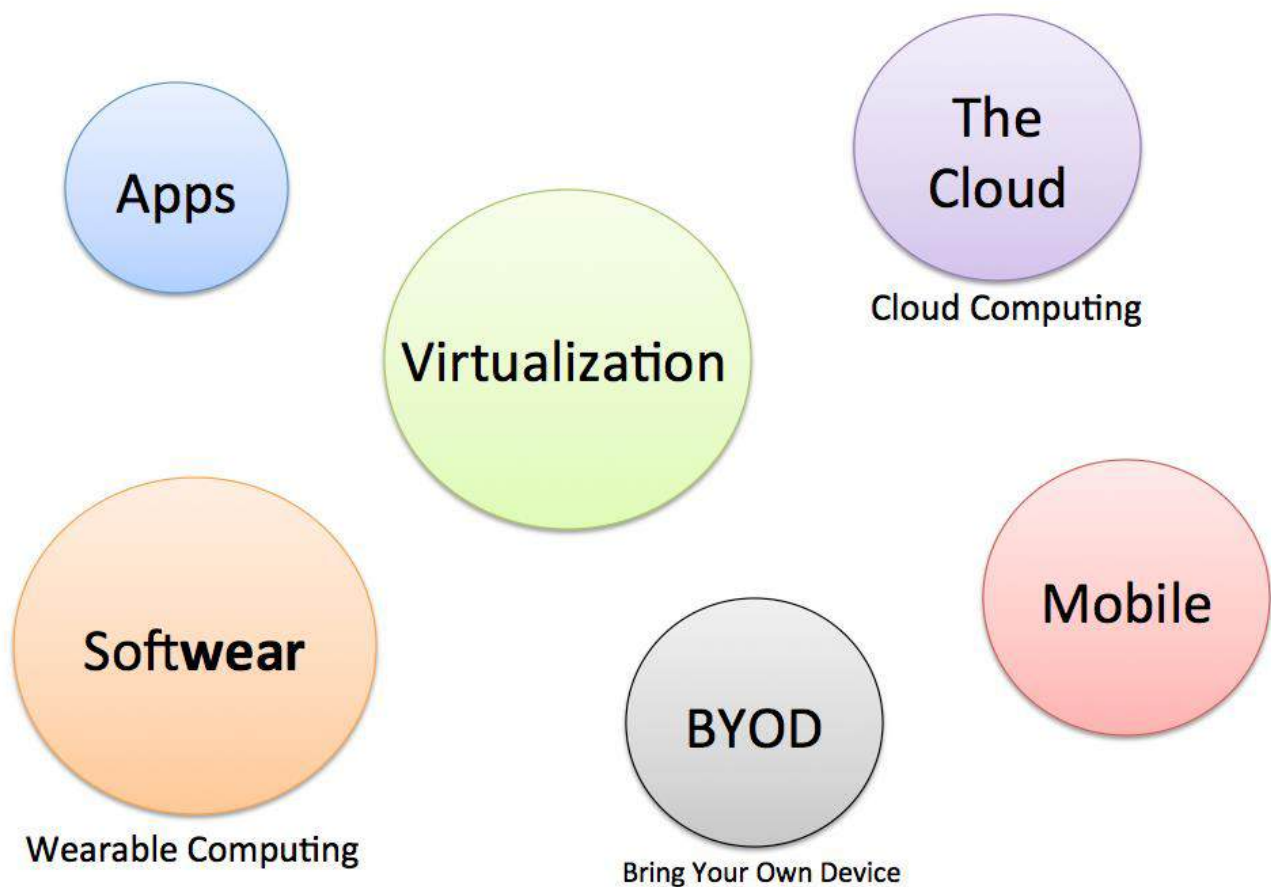
Figure 2-3: Software Trends

Apps and Mobil devices: Everybody has a mobile device today and fewer PCs are sold than ever before. Licensing: You don't buy, but lease software these days and all your information is stored in the Cloud, and some software is running in the Cloud (so-called Cloud Computing). Security challenges is very important in this case.

The companies that develop software need to face these facts and make the necessary changes to survive.

"Softwear" and Wearable Computing: Now we have watches like Galaxy Gear, the Apple Watch, Google Glass, etc.

In Figure 2-4 we see some examples of Software Trends.

Figure 2-4: Software Trends Examples

3 Software Development

In this chapter, we will give a short overview of software development, the challenges and what kind of different software categories we have, what kind of documents that are needed and created during the software development process, what kind of skills needed, etc.

🎬 Software Engineering: https://youtu.be/f4dpWVzpvUE

Terms:

Software Development (also known as application development, software design, designing software, software application development, enterprise application development, or platform development) is the development of a software product.

Software Engineering (SE) is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software.

In Figure 3-1 we see how a software application typically interacts with users, the underlying operating system and hardware.
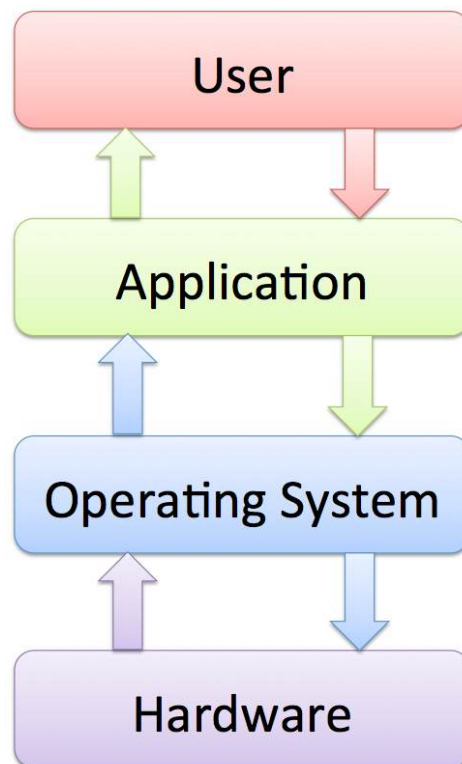
Figure 3-1: Software Interaction with Hardware and Users

3.1 Challenges

In Figure 3-2 we see some of the challenges in software development.
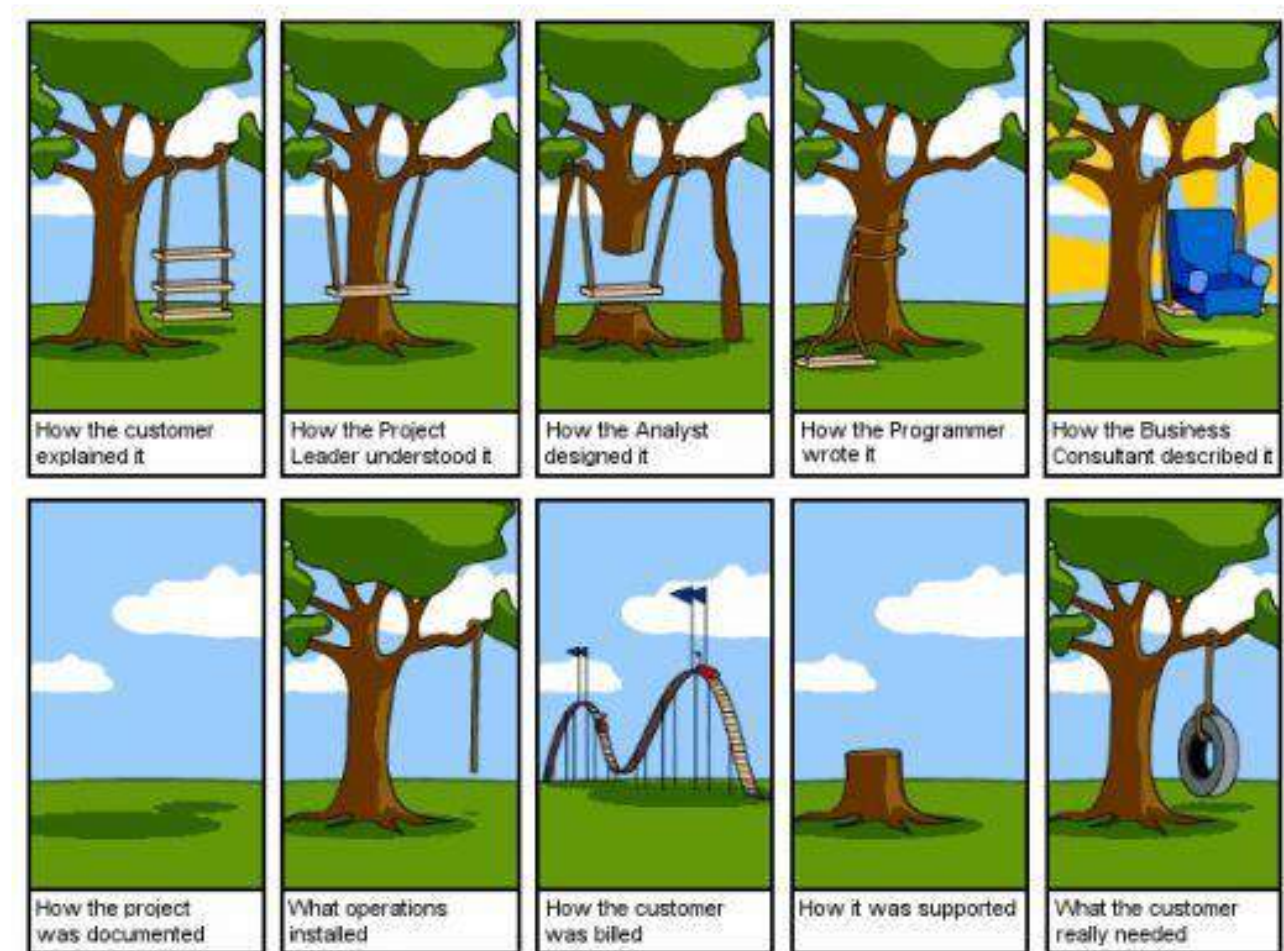
Figure 3-2: Challenges with Software Development

Collaboration and communication within the team and with stakeholders, etc. is crucial when it comes to creating good software.

Creating software is complicated. It is important to understand the customers' needs! In some way, you need to find out what the customer needs.

Market research, etc. is a good start, but in the end, you need to go much deeper to understand the customer. Most of the time the customer doesn't even know what they need.

3.2 Software Systems

In software development we have different kinds of systems, such as [1]:

- **Stand-alone applications**

- These are application systems that run on a local computer, such as a PC. They include all necessary functionality and do not need to be connected to a network.

- **Interactive transaction-based applications**

  - Applications that are executed on a remote computer and are accessed by users from their own PCs or terminals. These include web applications such as e-commerce applications.

- **Embedded control systems**

  - These are software control systems that control and manage hardware devices. Numerically, there are probably more embedded systems than any other type of system.

- **Batch processing systems**

  - These are business systems that are designed to process data in large batches. They process large numbers of individual inputs to create corresponding outputs.

- **Entertainment systems**

  - These are systems that are primarily for personal use, and which are intended to entertain the user.

- **Systems for modeling and simulation**

  - These are systems that are developed by scientists and engineers to model physical processes or situations, which include many, separate, interacting objects.

- **Data collection systems**

  - These are systems that collect data from their environment using a set of sensors and send that data to other systems for processing.

- **Systems of systems**

  - These are systems that are composed of several other software systems.

We can split the software systems into 2 main categories:

Generic products

- Stand-alone systems that are marketed and sold to any customer who wishes to buy them.

- Examples – PC software such as graphics programs, project management tools; CAD software; software for specific markets such as appointments systems for dentists.

Examples: Microsoft Office

Customized products

- Software that is commissioned by a specific customer to meet their own needs.

- Examples – embedded control systems, air traffic control software, traffic monitoring systems.

3.3 Documentation

Lots of documentation is involved in software development, see Figure 3-3. In this document, we will go through all the documentation needed in the different phases involved in software development.

Some important documents are:

- **SRS** – Software Requirements Specifications

    o A document stating what an application must accomplish

- **SDD** – Software Design Document

    o A document describing the design of a software application

- **STP** - Software Test Plan

    o Documentation stating what parts of an application will be tested, and the schedule of when the testing is to be performed

- **STD** - Software Test Documentation

    o Contents: Introduction, Test Plan, Test Design, Test Cases, Test procedures, Test Log, …, Summary

More about Software Documentation later in this document.
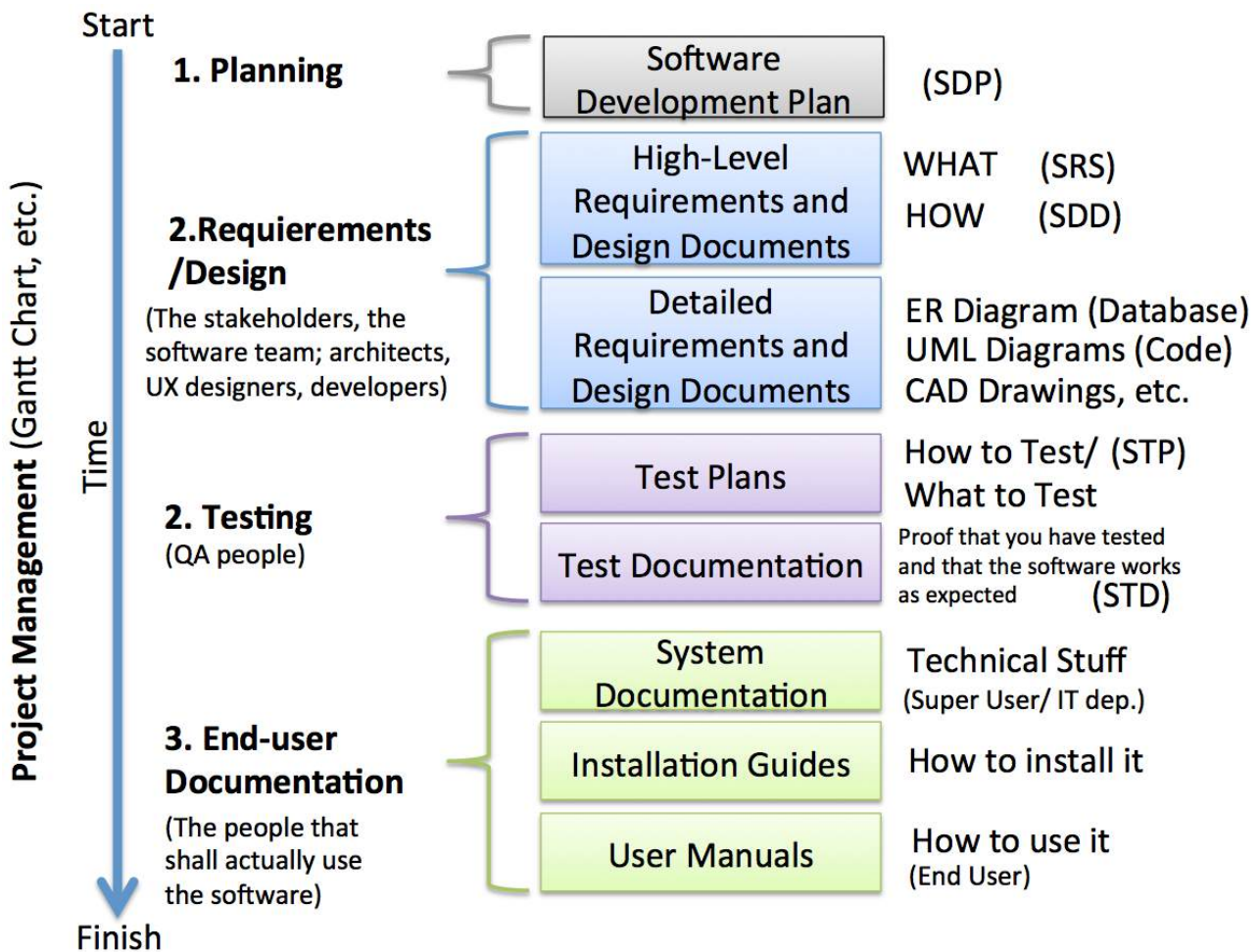
Figure 3-3: Typical Documentation involved in Software Development

QA – Quality Assurance. Quality Assurance (QA) refers to the engineering activities implemented in a quality system so that requirements for a product or service will be fulfilled.

3.4 Iterations and Releases

In Software Development, we typically have different iterations and releases, as shown in Figure 3-4.
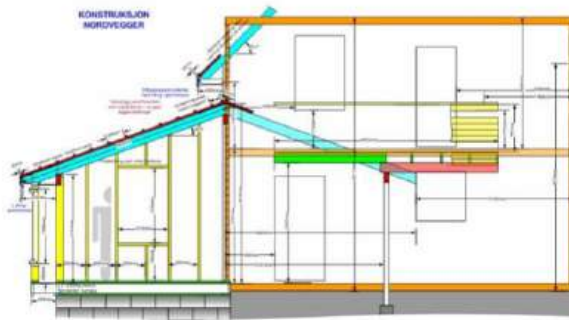
Figure 3-4: Software Iterations and Releases

Part 2 : Software Engineering

In this part, we introduce the different features and topics involved in software engineering, such as software teams, software development processes, software project management, etc.

4 Development Teams

A typical Software Team consists of the following roles:

- Project Manager
- System Architect
- UX Designer (Software Designer)
- Programmer
- Software Tester

In addition, we have Stakeholders or Customers that play an important role in the development.

In Figure 4-1 we see a typical Software Team.
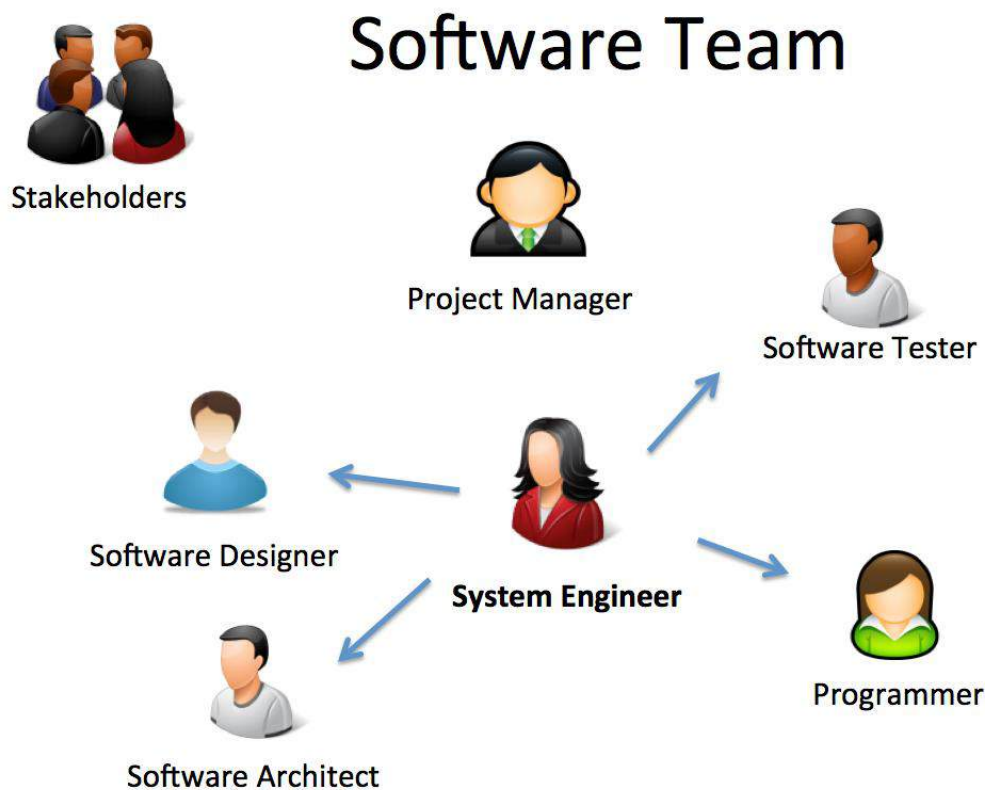
Figure 4-1: Software Team

A System Engineer is a general person that could be a Programmer, Architect, Designer, Tester in different phases in the project, or he could be a tester in one project and a programmer in another project – all in one person. That is usually the case in small companies, while in larger companies these roles (designer, tester, programmer) could be a full-time job.

4.1 Teams

To successfully create software, collaboration inside the team is essential.
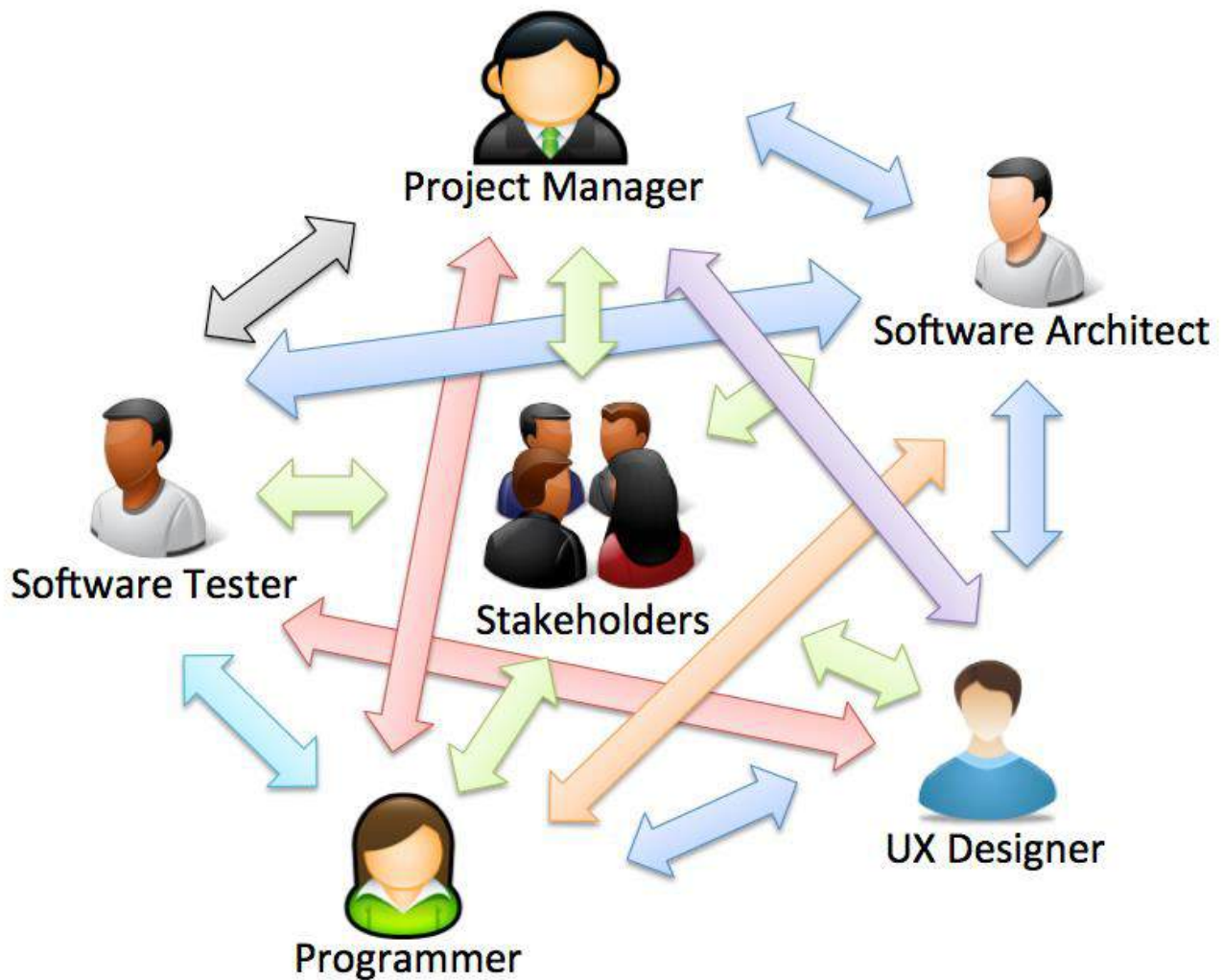
Figure 4-2: Team Collaboration

It is important that the team collaborate. Communication as well!

4.2 Roles

A typical Software Team consists of the following roles:

- Project Manager
- System Architect
- UX Designer (Software Designer)
- Programmer
- Software Tester

In addition, we have Stakeholders or Customers that play an important role in the development.

We will discuss these roles in more detail below.

4.2.1   Stakeholders

All the people that have an interest in the outcome of the software are called Stakeholders. In most cases the Stakeholders are referred to as "Customers" but others may also be referred to as stakeholders, such as management, shareholders, etc.

4.2.2   Project Manager

The Project Managers have responsibility of the planning, execution and closing of the project.

More about Project management in a later chapter.

4.2.3   System Architect

With "Technical Design" we mean the Platform and Architecture Design, i.e., how to build the software.

This is typically done by a so-called Software/System Architect.

4.2.4   UX Designer

UX Design is the Design of the User eXperience (UX) and the Graphical User Interface (GUI), sometimes also called Human Machine Interface (HMI). This is what the end user of the software sees.

This is typically done by a so-called UX Designer.

4.2.5   Programmer

The Programmer or the Developer is doing the actual implementation of the software, i.e., the coding.

4.2.6   Software Tester

Before the customer can start using the software it needs to be properly tested. The Developer/Programmer needs to test his software, but since software consists of several software modules and components created by different developers, we need dedicated software testers that can test the software on a higher level.

The Customers are/should also be involved in the testing as well.

5 Software Development Phases

In software development, we have the following phases:

- Requirements (e.g., from Customer)
- Analysis and Design
- Implementation, Coding
- Documentation
- Testing
- Deployment, Installation and Maintenance

This chapter introduces these phases. Figure 5-1 shows an overview of the different phases involved in Software Development:
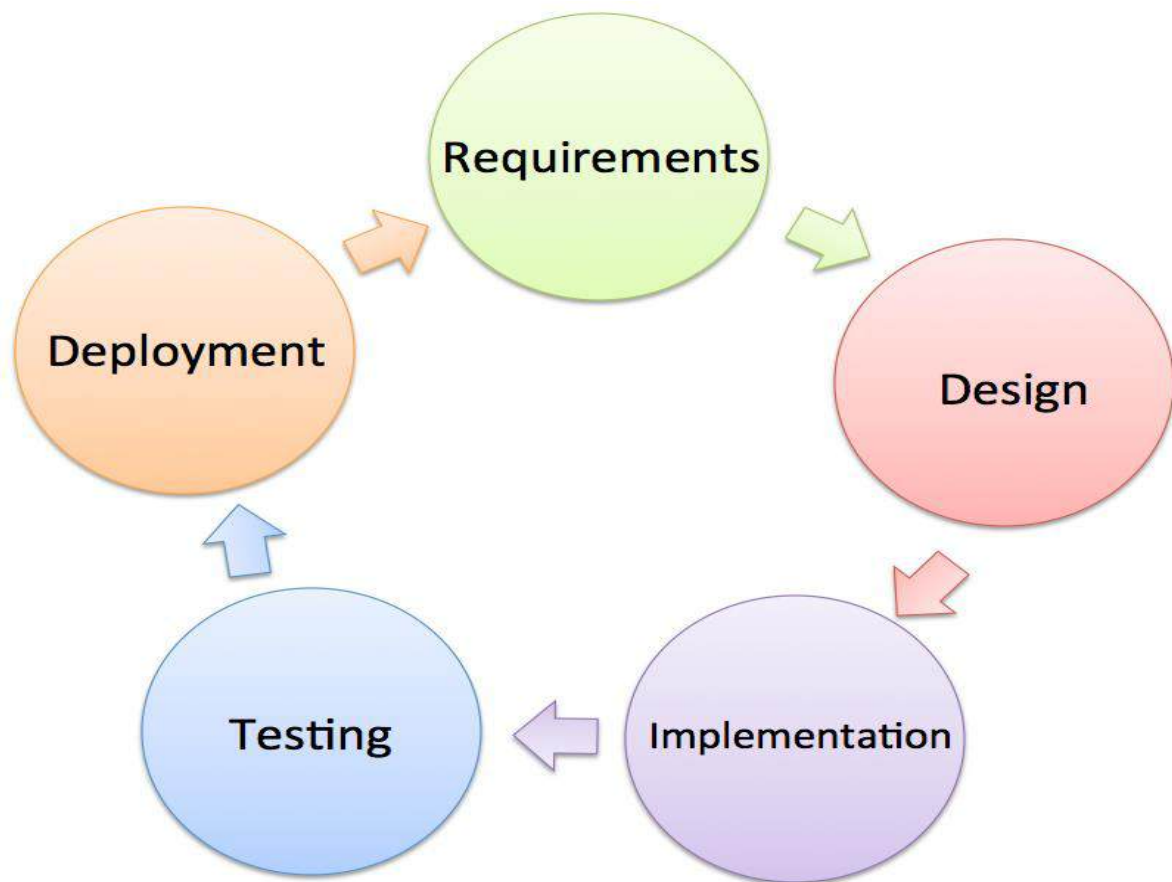
Figure 5-1: Phases in Software Development

5.1 Requirements

In the requirements, we describe what the system should do. The requirements include both functional requirements and non-functional requirements [1].

Functional Requirements: Statements of services the system should provide, how the system should react to inputs and how the system should behave in different situations. May state what the system should not do.

Non-Functional Requirements: Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc. Often apply to the system rather than individual features or services.

[https://dilbert.com]
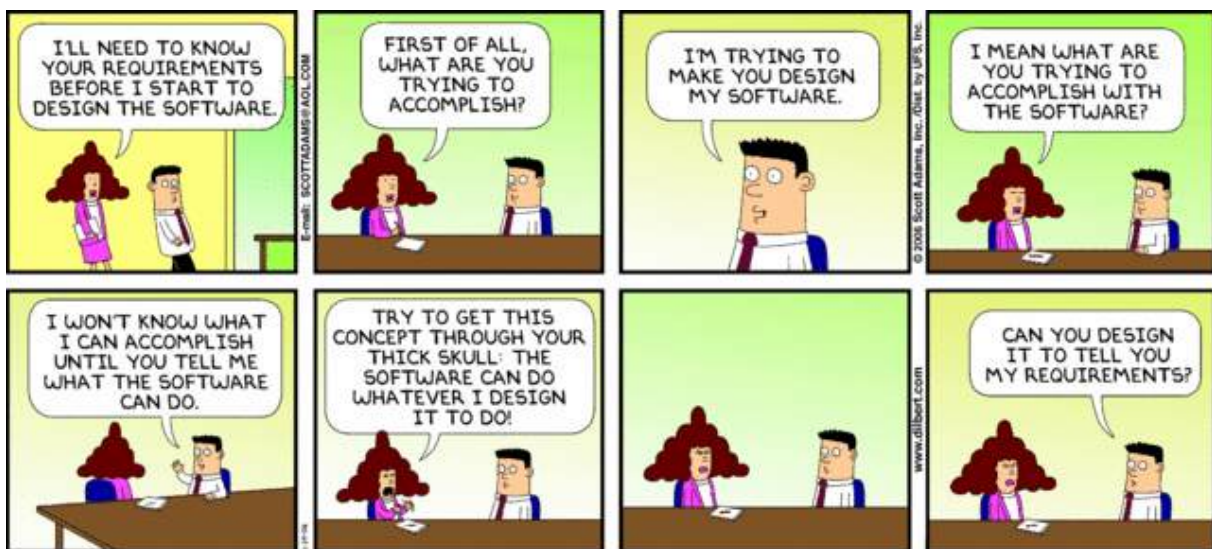
The requirements are often collected in a so-called "Software Requirements Specification (SRS)" document.

The SRS could contain stuff like [2]:

- Introduction
  - Purpose
  - Definitions
  - System overview
  - References
- Overall description
  - Product perspective
    - System Interfaces
    - User Interfaces
    - Hardware interfaces
    - Software interfaces

- Communication Interfaces
- Memory Constraints
- Operations
- Site Adaptation Requirements
  - o Product functions
  - o User characteristics
  - o Constraints, assumptions, and dependencies
- Specific requirements
  - o External interface requirements
  - o Functional requirements
  - o Performance requirements
  - o Design constraints
    - Standards Compliance
  - o Logical database requirement
  - o Software System attributes
    - Reliability
    - Availability
    - Security
    - Maintainability
    - Portability

- Other requirements

The Requirements is normally given by the Customer if we deal with customized products. The software requirements document is the official statement of what is required of the system. It should include both a definition of user requirements and a specification of the system requirements. It is NOT a design document. As far as possible, it should include a set of WHAT the system should do rather than HOW it should do it [1].

5.2 Design

In the design phase, we use the specification and transform it into descriptions of how we should do it.

In principle, requirements should state what the system should do, and the design should describe how it does this – but in practice this is not so easy! - In practice, requirements and design are inseparable.

We can divide design into 2 main groups:

- Technical Design – Platform and Architecture Design, i.e., how to build the software.
- UX Design – Design of User eXperience (UX) and the Graphical User Interface (GUI), sometimes also called Human Machine Interface (HMI). This is what the end user of the software sees.

5.2.1   Technical Design

Technical Design is Platform and Architecture Design, i.e., how to build the software. This is typically done by a so-called Software Architect.

5.2.2   UX Design

UX Design is the Design of the User eXperience (UX) and the Graphical User Interface (GUI), sometimes also called Human Machine Interface (HMI). This is what the end user of the software sees. This is typically done by a so-called UX Designer.

5.3 Implementation

Implementation = Coding.

Software is usually designed and created (coded/written/programmed) in integrated development environments (IDE) like Eclipse, Xcode or Microsoft Visual Studio that can simplify the process and compile the program to an executable unit. Software is usually created on top of existing software and the application programming interface (API) that the underlying software frameworks provide, e.g. Microsoft .NET, etc.

Most of the software has a Graphical User Interface (GUI). Normally you separate the GUI design and code into different layers or files.

[http://geek-and-poke.com]
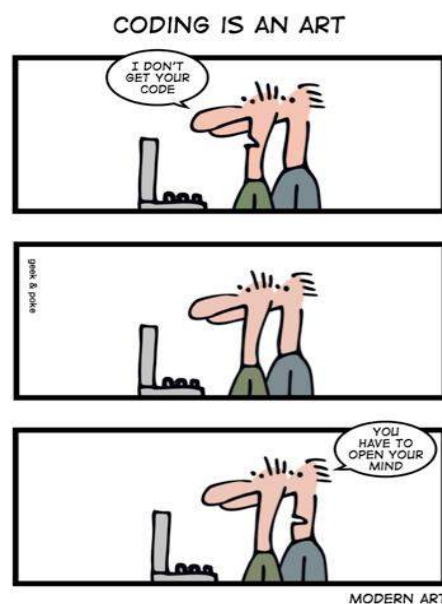
More about implementation later in this document.

5.4 Testing

Testing can be performed on different levels and by different people. Testing is a very important part of software development. About 50% of the software development is about testing your software. Creating User-friendly Software is Crucial! More about Testing later in this document.

5.5 Deployment

What is Deployment?

Software deployment is all the activities that make a software system available for use.

Examples:

- Get the software out to the customers

- Creating Installation Packages

- Documentation, e.g., Installation Guide, etc.

- Installation

- etc.

Deployment strategies may vary depending of what kind of software we create, etc.

More about Deployment later in this document.

When the software is deployed, or installed, you normally go into a Maintenance phase. The maintenance of software involves bug fixes of the software after the software is released, etc. At some time, you also need to start planning new releases of the software.

6 Software Development Process

There are lots of different software development processes or methods in use today [3], e.g.:

- Waterfall model
- V-model
- Spiral model
- Unified Process (UP)/ Rational Unified Process (RUP)
- Scrum
- eXtreme Programming (XP)
- Lean Software Development
- TDD (Test Driven Development)
- Lean Software Development
- Kanban
- etc.

These processes or models may be divided in 2 main categories: **Plan-driven models** and **Agile methods**. The Waterfall model, V-model and the Spiral model are so-called plan-driven models, while Scrum and eXtreme Programming are so-called Agile methods.

[http://geek-and-poke.com]
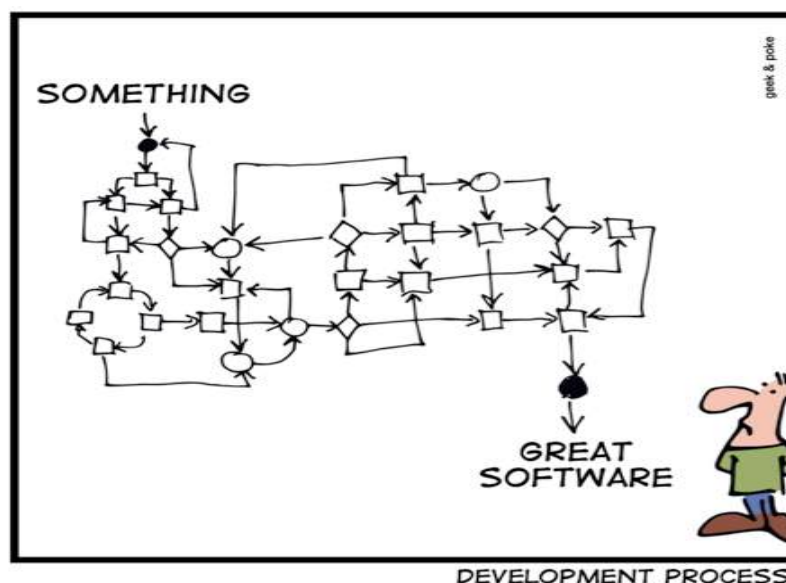
Traditionally plan-driven methods were used in software development, while today Agile methods such as Scrum have become very popular, especially in smaller development teams.

Plan-driven models (e.g. Waterfall) generally produce more documentation than Agile models.

In Figure 6-1 we see an overview of some of the most used methods.
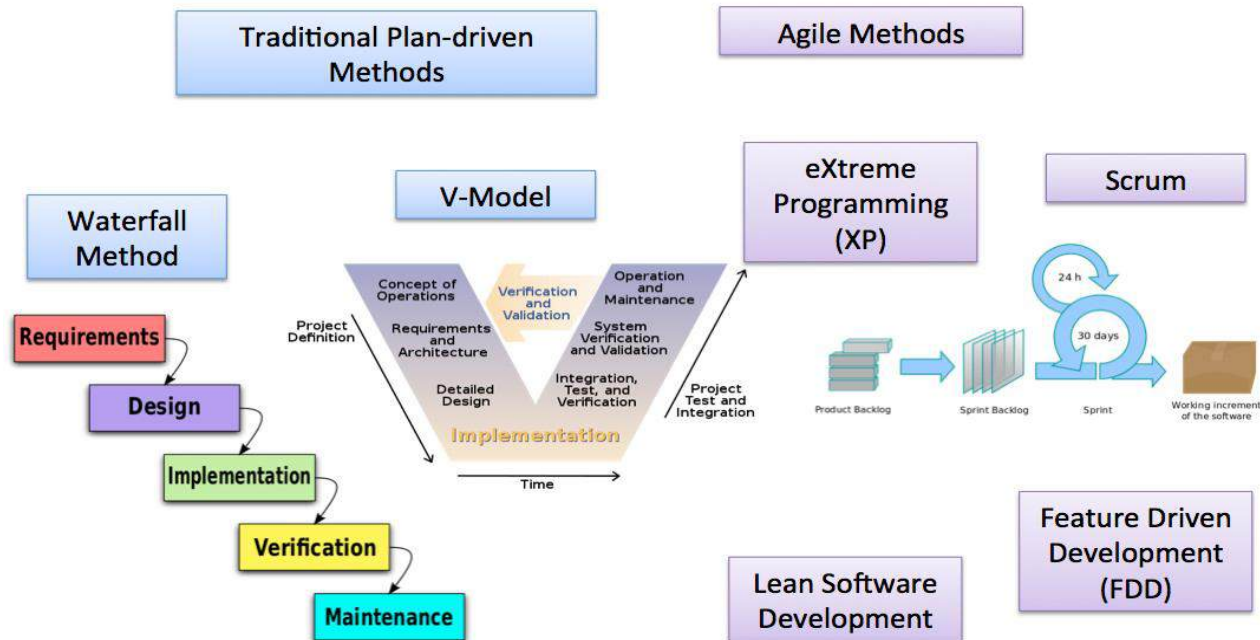
Figure 6-1: Software Development Methods

In Figure 6-2 we see the main difference between Agile development and ordinary plan-driven development.
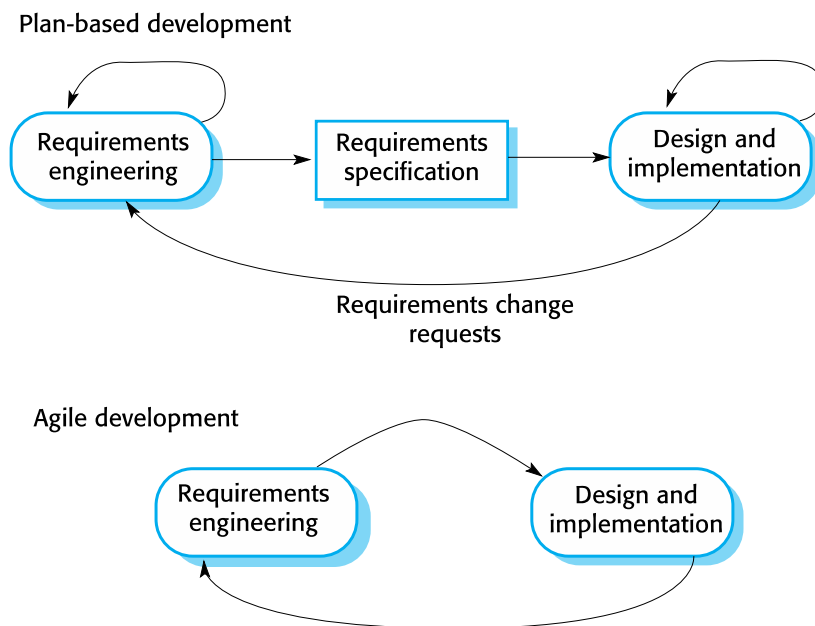
Figure 6-2: Plan-driven vs. Agile Development [1]

HOW DO YOU KNOW IT'S AGILE?

PART 1: NO CODE OWNERS

[http://geek-and-poke.com]
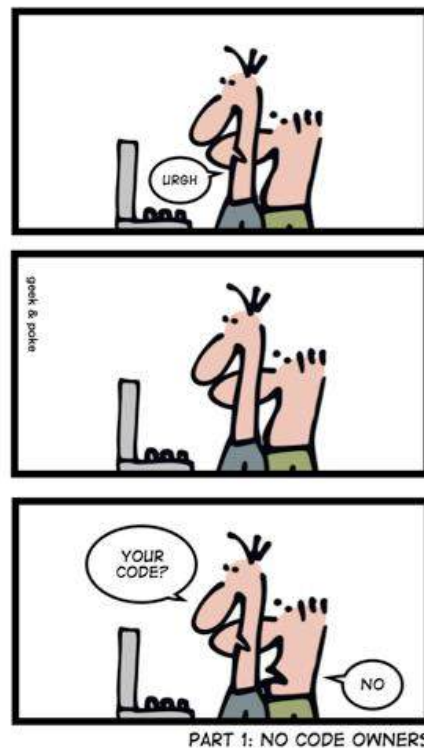
6.1 Plan-driven models

We have different plan-driven models such as the Waterfall model, V-model, and Spiral model which we will discuss in more details.

6.1.1   Waterfall model

The Waterfall model [4] consists of the following phases:

- Requirements specification (Requirements analysis)
- Software design
- Implementation and Integration
- Testing (or Validation)
- Deployment (or Installation)
- Maintenance

Traditionally with the Waterfall model, you can only start on the next phase when the previous phase is finished. Therefore, it is called the Waterfall method, see Figure 6-3.
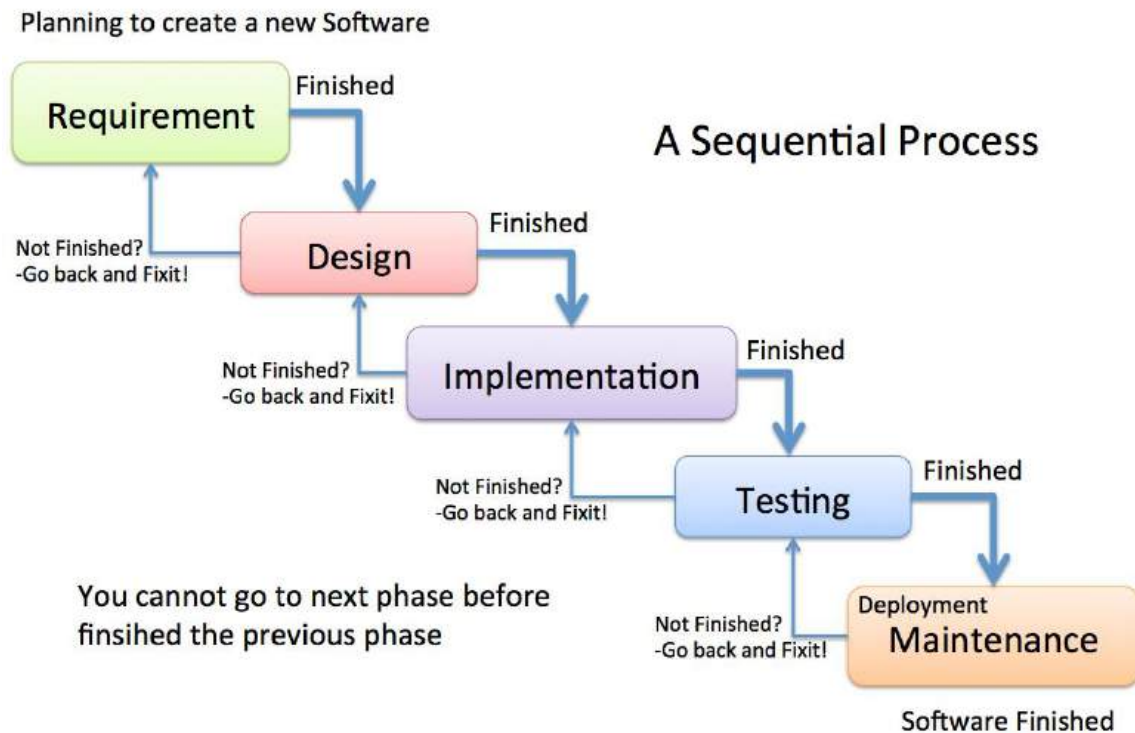
Figure 6-3: Waterfall model [4]

In practice, there is impossible to create perfect requirements and design before you start implementing the code, so it is common to go back and update these phases iteratively.

6.1.2   V-model

The V-model [5] is derived from the more traditional Waterfall model.

The V-model is an extension of the waterfall model, but it's using a more flexible approach.

"The V-Model reflects a project management view of software development and fits the needs of project managers, accountants and lawyers rather than software developers or users."
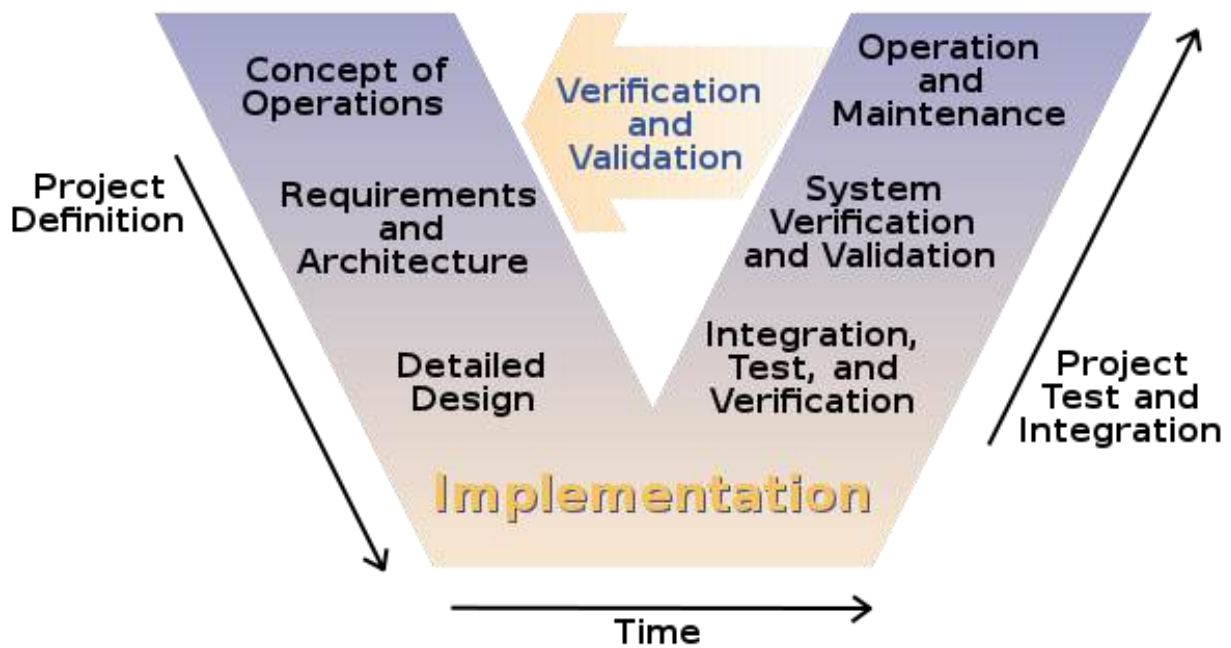
Figure 6-4: V-model [5]

As we see in Figure 6-4, the left side is about requirements and design, while the right-side of the model is about testing and validating.

6.2 Agile Software Development

Agile software development is a group of software development methods based on iterative and incremental development.

[https://dilbert.com]

So, what is Agile development? – Here is a short summary:

- A group of software development methods
- Iterative approach
- Incremental: Software available to Customers every 2-4 weeks

- Self-organizing and cross-functional Teams
- Refactoring

In Figure 6-5 we see some important Agile features and principles.
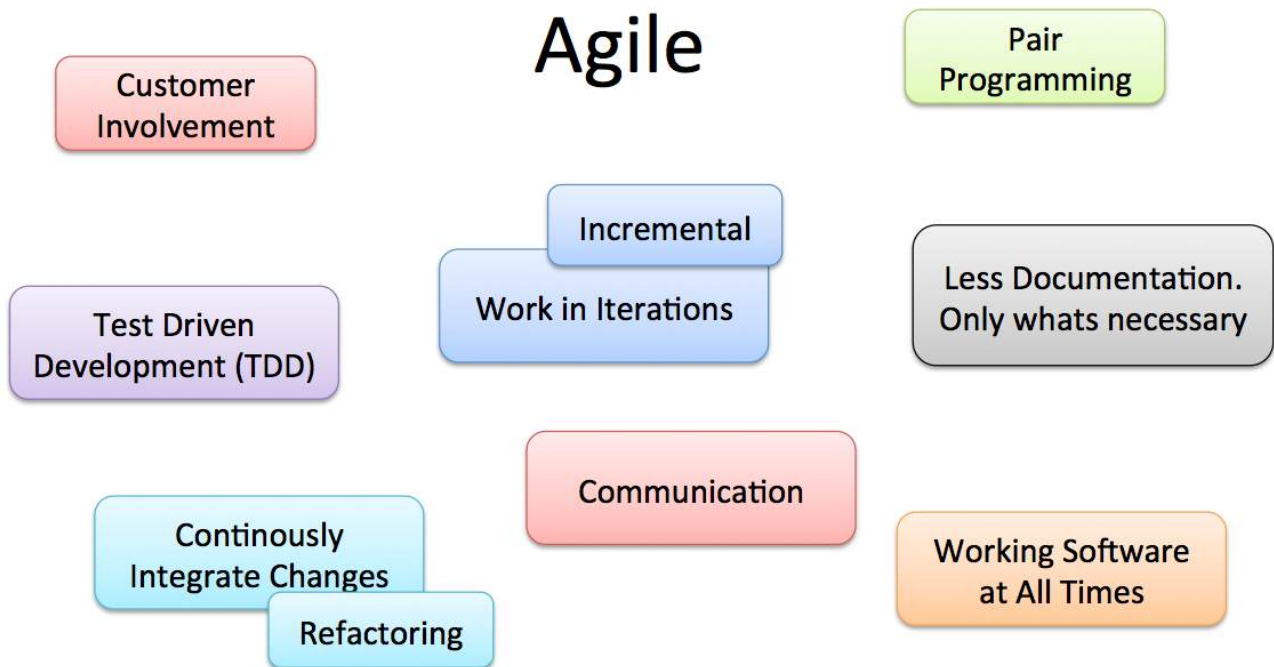
Figure 6-5: Agile Features and Principles

Examples of popular Agile methods:

- Scrum
- eXtreme Programming (XP)

In Figure 6-6 we see the key features with Agile Software Development.

Figure 6-6: Agile Software Development

Figure 6-7 shows some main differences between Agile development and more traditional development methods, such as, e.g., the Waterfall method.
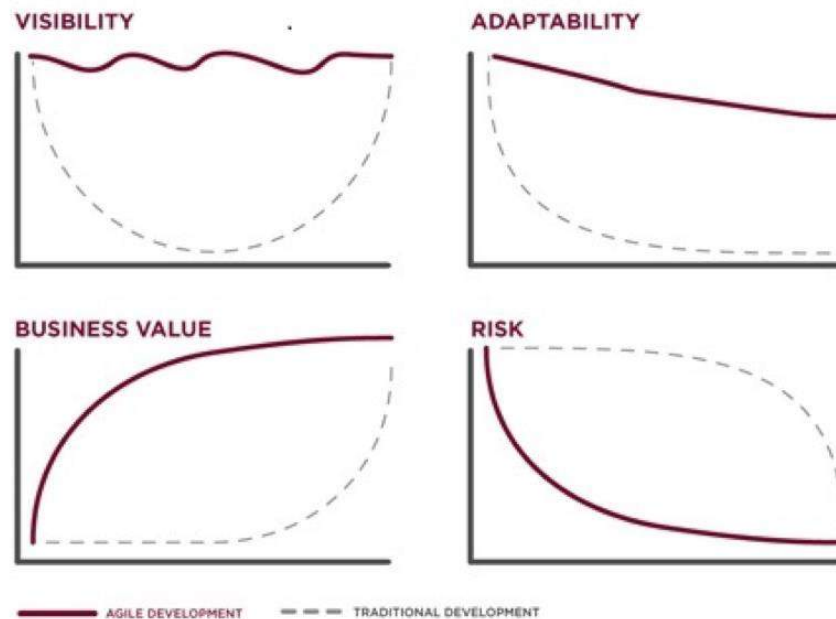
Figure 6-7: Agile vs. Traditional Development [6]

6.2.1   The Manifesto for Agile Software Development

In 2001, some software developers met to discuss development methods. They published the Manifesto for Agile Software Development to define the approach now known as agile software development.

The Manifesto for Agile Software Development is as follows [7]:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work, we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

[http://geek-and-poke.com]
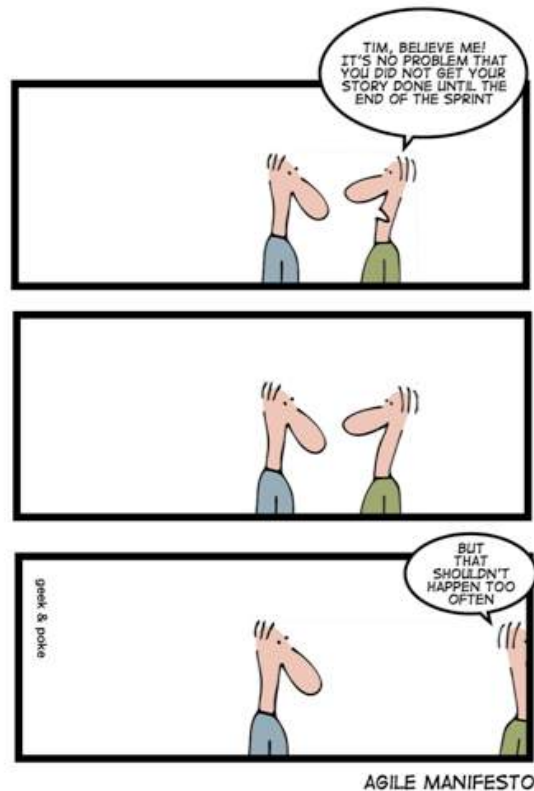
6.2.2   Burndown Chart

A burn down chart is a graphical representation of work left to do versus time. The outstanding work (or backlog) is often on the vertical axis, with time along the horizontal. That is, it is a run chart of outstanding work. It is useful for predicting when all the work will be completed.

It is often used in agile software development methodologies such as Scrum. However, burn down charts can be applied to any project containing measurable progress over time.

In Figure 6-8 we see a typical Burndown chart.
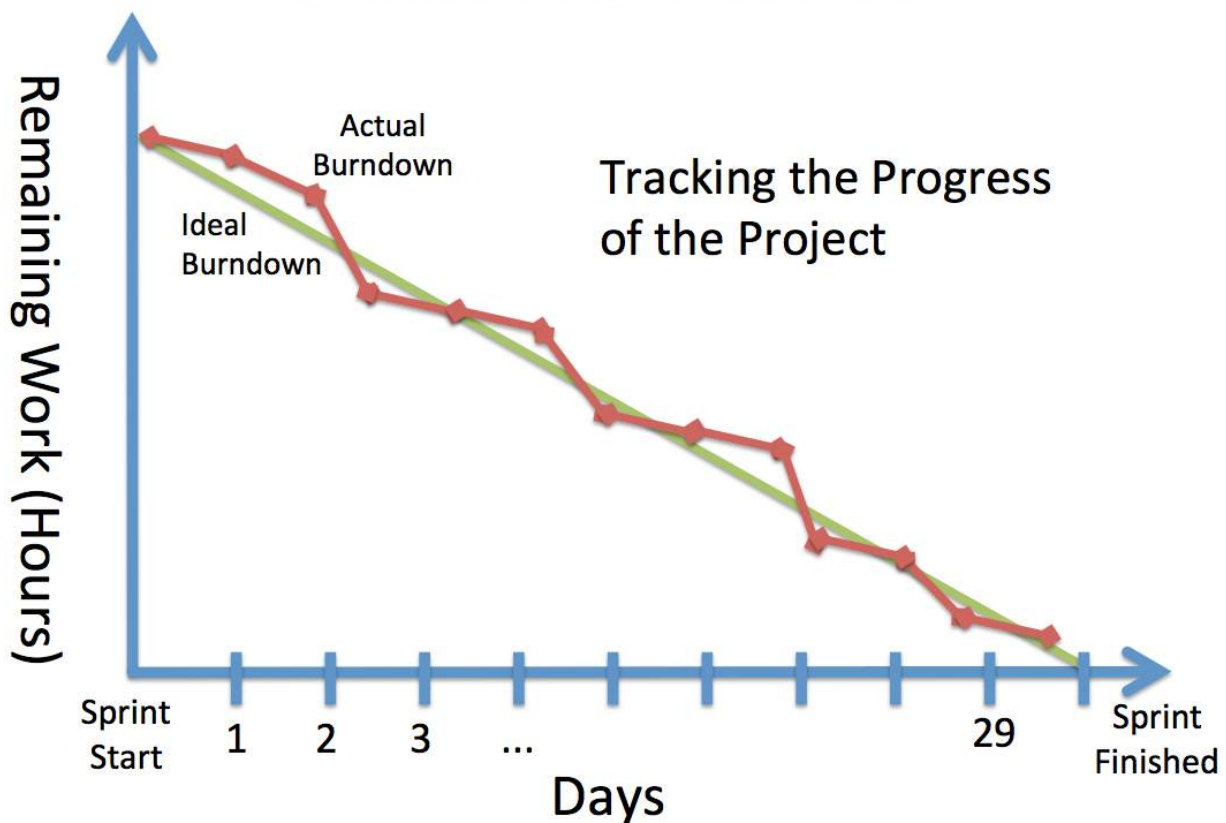
Figure 6-8: Burndown Chart

6.2.3   Waterfall vs. Agile

Agile is more flexible than traditional methods (like the waterfall).

Here are some key factors that separate the traditional waterfall method versus the more flexible Agile methods, such as Scrum:

- Agile and Scrum is based on Iterations while Waterfall is Sequential
- Agile and Scrum focus on less documentation
- Agile is good for small projects – not so good for larger projects?
- If the Customer don't know what he wants in detail – Scrum is a good approach

In Figure 6-9 we see some important differences between the traditional waterfall method and the Agile Development approach. We see that Agile delivers value in each iteration of the development.
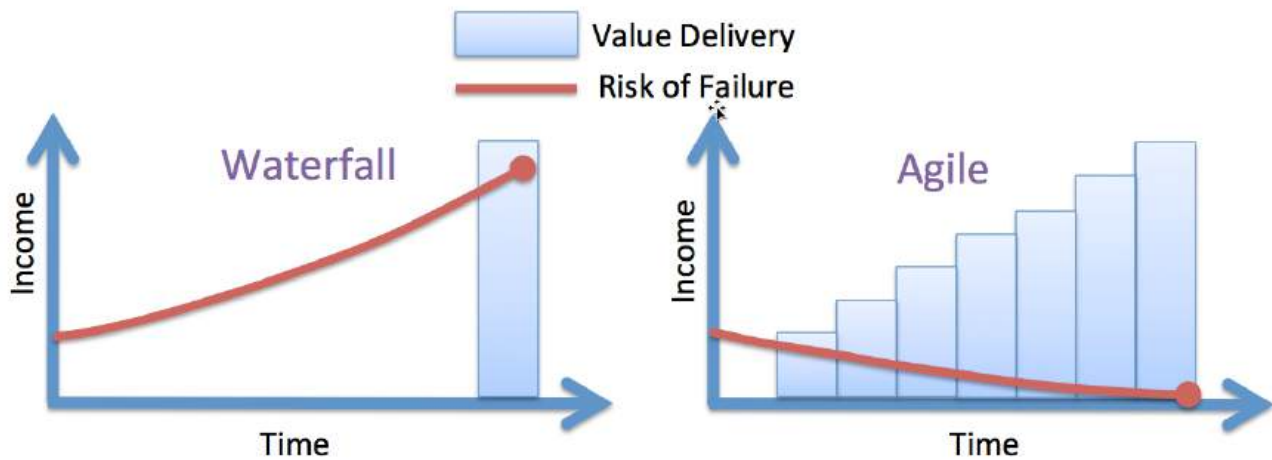
Figure 6-9: Waterfall vs. Agile Development

6.2.4   eXtreme Programming (XP)

eXtreme Programming or shorted XP is a popular Agile method. Typical features in XP are as follows:

- Pair Programming
- Code Reviews
- Refactoring
- Unit Testing - In XP you start by writing Unit Tests before you start coding
- Standup Meetings

[https://dilbert.com]

In Figure 6-10 we see how XP works in practice.
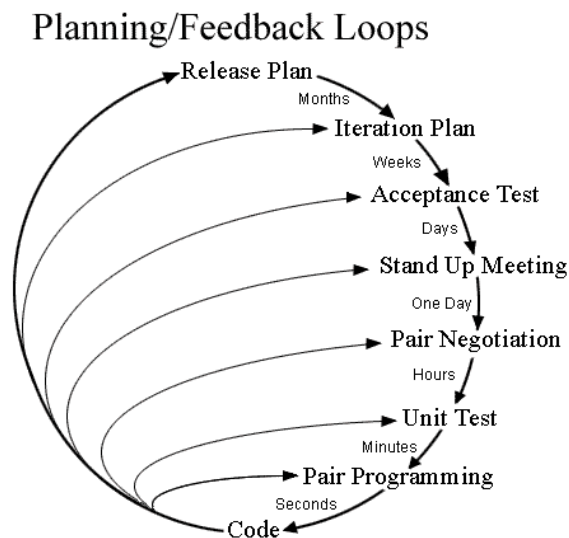
Figure 6-10: eXtreme Programming

In XP, they practice so-called "Pair Programming" (Figure 6-11), meaning 2 developers working together.

Figure 6-11: Pair Programming [8]

So, is Pair Programming Good or Bad? There exists various studies of the productivity of Pair Programming [1]:

- Study 1: Comparable with that of 2 developers who work independently
- Study 2: A significant loss in productivity compared with 2 developers working alone

A reasonable question is: Should the 2 developers have the same skills or not?

Newer less, there are benefits with XP:

- Collective Ownership for the code created and the results of the project.

- Continuous informal Review process because each code line is looked at by at least 2 people
- It supports Refactoring, which is a continuous process of software improvement
- Less time is spent on repairing bugs.
- Improved Code Quality
- It reduces the overall risk

[http://geek-and-poke.com]
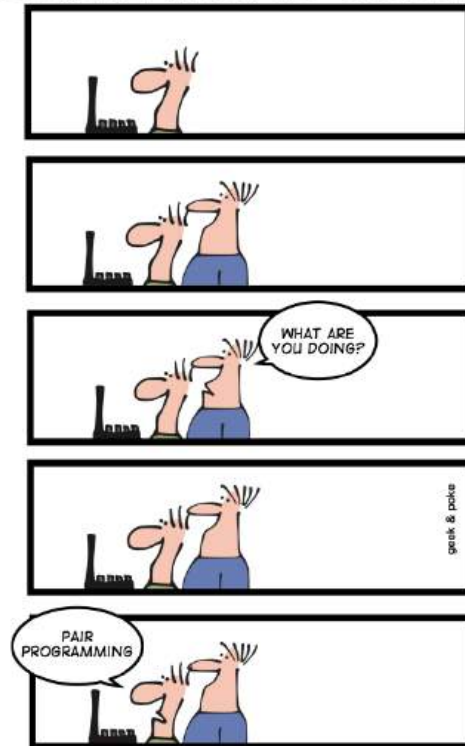
6.2.5   Scrum

Scrum [9] is a so-called Agile method, and it has become very popular today. In Figure 6-12 we see an overview of the Scrum method.

 Scrum with Examples: https://youtu.be/h3RVZv3NGWk

Scrum is simple and easy to understand. The method is more flexible and more informal than plan-driven methods.
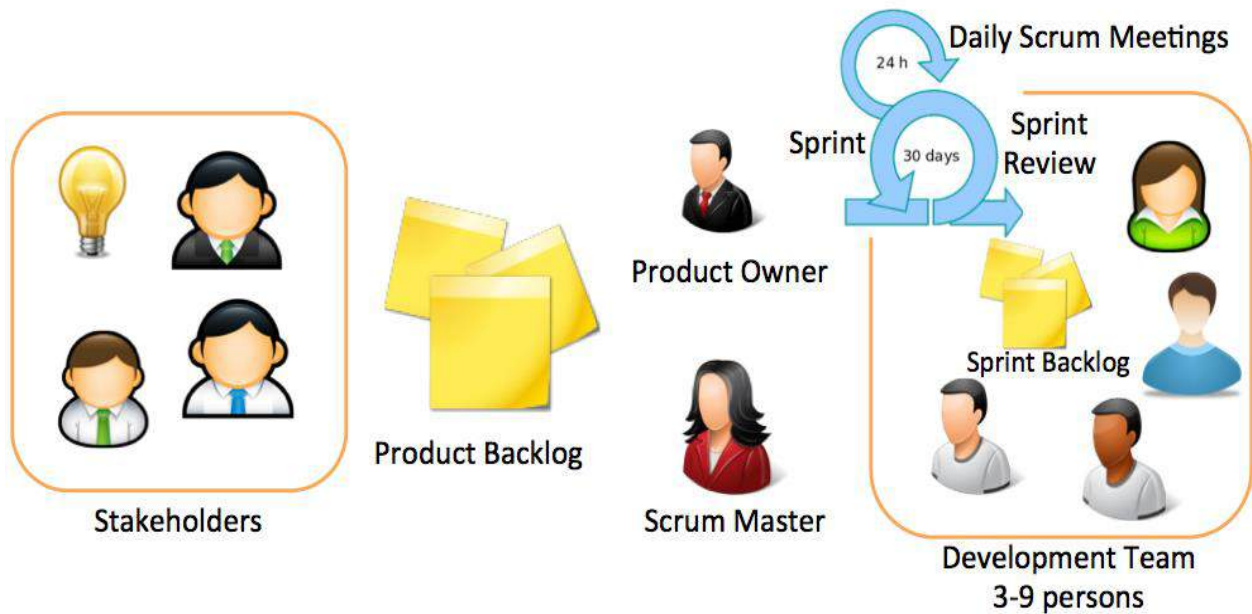
Figure 6-12: Scrum Overview

In short, Scrum is a Framework for Software Development.

- Agile Software Development method
- Simple to understand
- Flexible
- Extremely difficult to master!
- Self-organizing Teams (3-9 people)
- Scrum Team:
  - o Product Owner
  - o Scrum Master
  - o Development Team

Some important Scrum Events are:

- The Sprint (duration between 14-30 days)

- Sprint Planning Meeting (8 hours if 30 days' sprint)

- Daily Scrum Meeting (Max 15 min, every day at the same time) (also called Standup Meeting)

- Sprint Review (4 hours if 30 days' sprint)

An example of a Daily Scrum Meeting is shown in Figure 6-13. It is normal to hold this meeting as a "standup meeting", where participants standing during the meeting.

Figure 6-13: Example of a Daily Scrum Meeting (Standup Meeting) [9]

Daily Scrum Meeting:

Important features of the Daily Scrum Meeting are as follows:

- Max 15 min.

- The meeting is held at the same time and place every day

- "Stand Up" Meeting

- Purpose:

  – Synchronize activities and create a plan for the next 24 hours.

  – Track Progress

- Agenda – Each Team member explains:

  – What has been accomplished since the last meeting?

  – What will be done before the next meeting?

  – What obstacles are there in the way?

6.2.6   Kanban

Kanban is based on Lean and Toyota production principles and Just-in-Time principles.   Kanban has fewer "rules" than scrum.   Kanban is flow-based, while Scrum is Time box-based (Sprints). Kanban focuses on limit the WIP (Work in Progress). Kanban has focus on estimation.

In Kanban, they use a Kanban board (Figure 6-14) to track the progress. The Kanban board is very like the Task board used in Scrum.
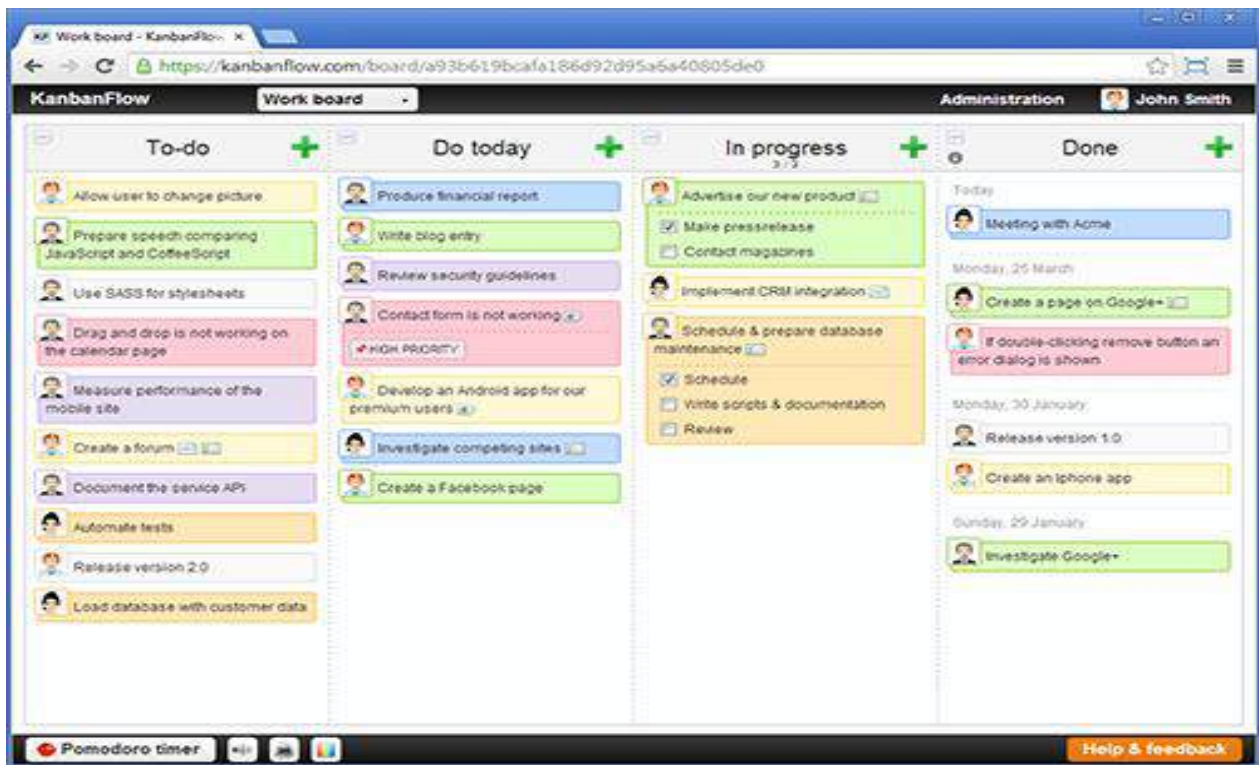
Figure 6-14: Example of a Kanban board

6.3 Hybrid Process Models

6.3.1   Unified Process (UP)/ Rational Unified Process (RUP)

The Unified Process (UP)/ Rational Unified Process (RUP) is a so-called hybrid process model [1]. It takes elements from many of the traditional plan drive methods as well iterative/incremental delivery, which is an important part of Agile methods.

The RUP has been designed to work together with UML (Unified Modelling Language).

In UP we have 4 different phases [10]:

- Inception
- Elaboration

- Construction
- Transition

Figure 6-15 shows these 4 phases.
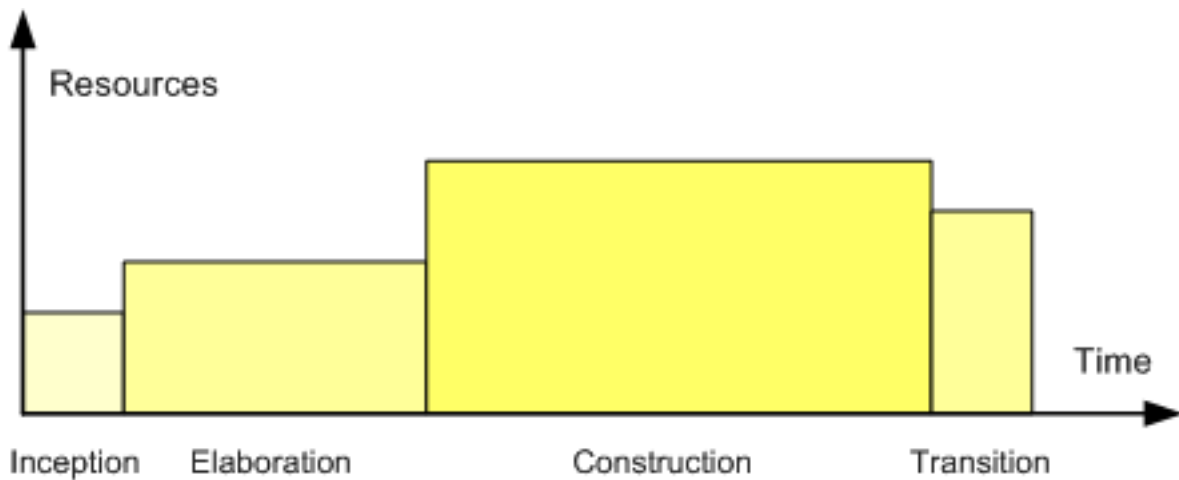
Figure 6-15: Phases in Unified Process (UP) [10]

6.4 Summary

Agile methods have become very popular today. Agile methods are good in some situations, while more traditional methods are better in other situations.

To create great software, we need to combine the best of all these approaches and adjust them to fit the needs of your company. There are lots of different kinds of software, and one method is not fit to solve all these different situations.

Agile methods have less focus on documentation

6.5 Exercises

Make sure to discuss and reflect over the following:

1. What is a Software Development Process? Why is it important to have a good Software Development Process?

2. Explain Plan-driven software development in general

3. Give some examples of such Plan-driven software development methods

4. Explain Agile software development in general

5. Give some examples of Agile software development methods

6. Explain the Waterfall method

7. Explain the differences between Agile and plan-driven development. Give some examples in each category

8. Explain some features used in eXtreme Programming (XP)?

9. What is Scrum?

10. Give examples of Advantages and Disadvantages with Scrum

11. What is a Daily Scrum Meeting?

12. What are the different phases involved in software engineering?

13. Suggest a Software Project where it may be beneficial to use the Waterfall model and another where Scrum is the best choice

7 Scrum

Scrum [9] is a so-called Agile method, and it has become very popular today. In Figure 7-1 we see an overview of the Scrum method.

Scrum is simple and easy to understand. The method is more flexible and more informal than plan-driven methods.

 Scrum with Examples: https://youtu.be/h3RVZv3NGWk
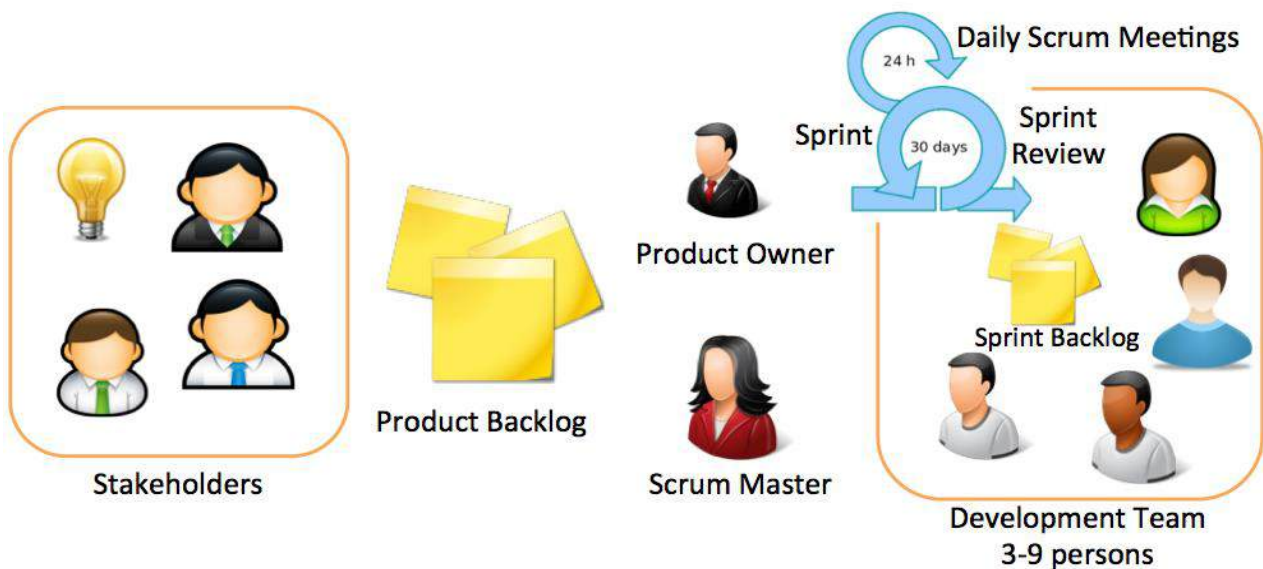
Figure 7-1: Scrum Overview

In short, Scrum is a Framework for Software Development.

- Agile Software Development method
- Simple to understand
- Flexible
- Extremely difficult to master!
- Self-organizing Teams (3-9 people)
- Scrum Team:
    - Product Owner
    - Scrum Master
    - Development Team

7.1 The Scrum Process

Figure 7-2 shows the Scrum Process.
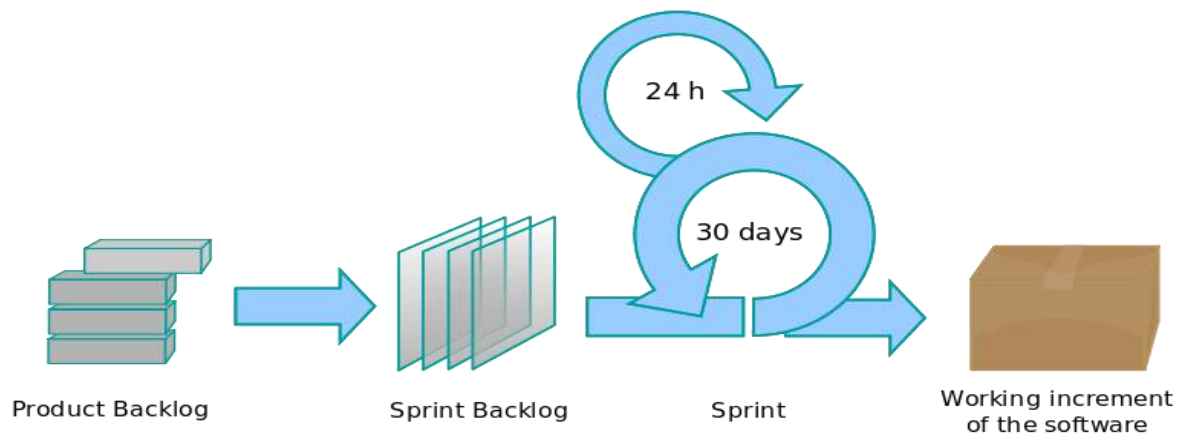
Figure 7-2: Scrum Process [9]

7.2 Scrum Events

- The Sprint (duration between 14-30 days)

- Sprint Planning Meeting (8 hours if 30 days' sprint)

- Daily Scrum Meeting (Max 15 min, every day at the same time) (also called Standup Meeting)

- Sprint Review (4 hours if 30 days' sprint)

An example of a Daily Scrum Meeting is shown in Figure 7-3. It is normal to hold this meeting as a "standup meeting", where participants standing during the meeting.

Figure 7-3: Daily Scrum Meeting (Standup Meeting) [9]

7.2.1   Daily Scrum Meeting

Important features of the Daily Scrum Meeting are as follows:

- Max 15 min.

- The meeting is held at the same time and place every day

- "Stand Up" Meeting

- Purpose:

    – Synchronize activities and create a plan for the next 24 hours.

    – Track Progress

- Agenda – Each Team member explains:

    – What has been accomplished since the last meeting?

    – What will be done before the next meeting?

    – What obstacles are there in the way?

[https://dilbert.com]

7.3 Scrum Artifacts

- Product Backlog

- Sprint Backlog

- Increment

Scrum's artifacts represent work or value in various ways that are useful in providing transparency and opportunities for inspection and adaptation. Artifacts defined by Scrum are specifically designed to maximize transparency of key information needed to ensure Scrum Teams are successful in delivering a "Done" Increment [11].

7.4 The Scrum Team

The Scrum Team has the following members:

- Product Owner

- Scrum Master

- Development Team

In addition, we have the Stakeholders, but they are not part of the Scrum team itself.

[http://geek-and-poke.com]
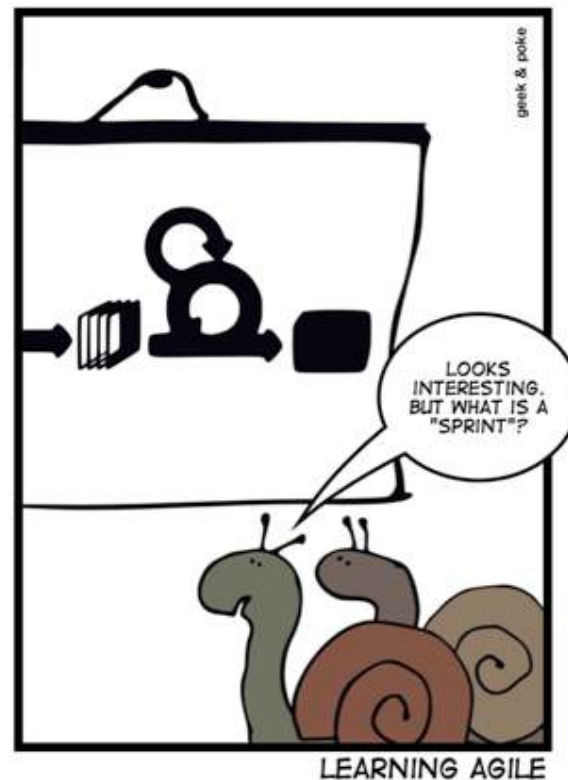
7.5 Scrum Meetings

So, if we summarize, we have 4 different meetings in Scrum:

- Sprint Planning Meeting
- Daily Scrum Meeting
- Sprint Review Meeting
- Retrospective Meeting

Figure 7-4 summarizes the different meetings (purpose, duration, and frequency).

Meeting	Purpose	Duration	Frequency
Sprint Planning Meeting	Determine what work to do in the coming sprint.	Two hours per week in the sprint, up to four hours	Once per sprint
Daily Scrum Meeting	Allow team members to commit, collaborate, and communicate risks.	Fifteen minutes	Daily
Sprint Review Meeting	Show the customer and other stakeholders the work that the team accomplished in the sprint, and receive feedback.	Two hours per week in the sprint, up to four hours	Once per sprint
Retrospective Meeting	Identify and implement ideas for process improvement.	Three hours	Once per sprint

Figure 7-4: Overview of Scrum Meetings

For Scrum meetings, we have the following guidelines:

- The meeting agenda should be clear.
- If team members start a discussion that does not address the purpose of the meeting, the members should take the discussion offline, to be completed later. The Scrum Master should identify and indicate when team members should take a discussion offline.
- All meetings should follow the basic structure that is described for that meeting.
- Meetings should start on time, even if some team members are late.
- Team members should be on time except in rare, unavoidable cases. If your schedule prevents you from being on time regularly, the conflict should be resolved as soon as possible. If necessary, the Scrum Master should adjust the meeting time to resolve the conflict if the change does not unfairly inconvenience another member of the team.
- Each team member should come to the meeting prepared.
- Meetings should finish on time. In most cases, the length of the meeting is determined by the length of the sprint. For example, take two hours for a sprint planning meeting if the sprint is one week, and four hours if the sprint is two weeks long.
- Scrum enforces this meeting structure to a level that might make people uncomfortable. This reaction comes from the pressure to be on time, the peer accountability that is associated with making and keeping commitments, and the transparency that is required to actively participate. Daily Scrum meetings are also usually a standup meeting.

7.6 Scrum Terms

Below we summarize the terms used in Scrum.

Scrum:

Scrum is a framework structured to support complex product development. Scrum consists of Scrum Teams and their associated roles, events, artifacts, and rules. Each component within the framework serves a specific purpose and is essential to Scrum's success and usage.

The Scrum Team:

The Scrum Team consists of a Product Owner, the Development Team, and a Scrum Master. Scrum Teams are self-organizing and cross-functional. Self-organizing teams choose best to accomplish their work, rather than being directed by others outside the team.

Development Team:

The Development Team are the professionals who do the work of delivering a potentially releasable Increment of "Done" product at the end of each Sprint. Development Teams are structured and empowered by the organization to organize and manage their own work.

Product Owner:

The Product Owner is the person responsible for maximizing the value of the product, the work of the Development Team, and management of the Product Backlog.

Scrum Master:

The Scrum Master is a servant-leader for the Scrum Team responsible for ensuring Scrum is understood and enacted. Scrum Masters do this by ensuring that the Scrum Team adheres to Scrum theory, practices, and rules.

Product Backlog:

The Product Backlog is an ordered list of everything that might be needed in the product and is the single source of requirements for any changes to be made to the product. The Product Owner is responsible for the Product Backlog, including its content, availability, and ordering.

Sprint Backlog:

The Sprint Backlog is the set of Product Backlog items selected for the Sprint plus a plan for delivering the product Increment and realizing the Sprint Goal. The Sprint Backlog is a forecast by the Development Team about what functionality will be in the next Increment and the work needed to deliver that functionality.

Increment:

The Increment is the sum of all the Product Backlog items completed during a Sprint and all previous Sprints.

Sprint:

The heart of Scrum is a Sprint, a time-box of one month or less during which a "Done", useable, and potentially releasable product Increment is created. Sprints have consistent durations throughout a development effort. A new Sprint starts immediately after the conclusion of the previous Sprint.

Sprint Planning Meeting:

The work to be performed in the Sprint is planned at the Sprint Planning Meeting. This plan is created by the collaborative work of the entire Scrum Team.

Daily Scrum:

The Daily Scrum is a 15-minute time-boxed event for the Development Team to synchronize activities and create a plan for the next 24 hours. This is done by inspecting the work since the last Daily Scrum and forecasting the work that could be done before the next one.

Sprint Review:

A Sprint Review is held at the end of the Sprint to inspect the Increment and adapt the Product Backlog if needed. During the Sprint Review, the Scrum Team and stakeholders collaborate about what was done in the Sprint. Based on that and any changes to the Product Backlog during the Sprint, attendees collaborate on the next things that could be done. This is an informal meeting, and the presentation of the Increment is intended to elicit feedback and foster collaboration.

Sprint Retrospective:

The Sprint Retrospective is an opportunity for the Scrum Team to inspect itself and create a plan for improvements to be enacted during the next Sprint. The Sprint Retrospective occurs after the Sprint Review and prior to the next Sprint Planning Meeting. This is a three-hour time-boxed meeting for one-month Sprints. Proportionately less time is allocated for shorter Sprints.

7.7 Tips and Tricks

Here are some Tips and Tricks when performing Agile/Scrum:

- Bring the Customer to the Daily Scrum Meetings

- Check out Pair Programming

- Use a Task Board (Whiteboard with Sticky Notes)

- Write Tests before you Write Code

- Continuously Integrate Changes and have Code Reviews and do continuous Code Refactoring

- Prioritize the Product Backlog

- Have Demonstrations for the Customer during the Project

- Be sure to have a common understanding of Goals, Problems and Solutions

7.8 Scrum Tools

There exists lots of tools that's support the Scrum methodology, Azure DevOps is one of them. More about Azure DevOps later in this document.

Azure DevOps with Scrum: https://youtu.be/-QmfMhtrxp0

8 Project Management

Project management is the key factor in any software development projects. Project management is the discipline of planning, organizing, motivating, and controlling resources to achieve specific goals.

Making Gantt diagram with Microsoft Excel: https://youtu.be/L31m3Jf87PY

Planner App in Microsoft Teams: https://youtu.be/LrZK3oUgkL4

In Figure 8-1 we see the well-known project triangle.

Figure 8-1: Project Triangle

Here are some Key factors for successful project management:

- Proper Planning
- Kick-off and Brainstorming
- Planning and Estimation
- Project Tracking

- Communication and Collaboration
- Meetings
- Using proper Tools, such as e.g., Azure DevOps

Project Work consists of working with Project Management, Development and Documentation in parallel.

If you remove one of these, the project will fail! Assume you have a table with 3 legs, see Figure 8-2. If you remove one of the legs from the table, the table will fall apart.

Figure 8-2: Project Management, Development and Documentation

8.1 Project Planning

Software development involves lots of activities that need to be planned and synchronized. To do that we need good tools for these activities. The Gantt chart is probably the most used tool. In addition, we need to have different meetings to plan and coordinate the different activities.

Agile Development also needs Project Management and Planning, which we will discuss in more detail later in this chapter.

[http://geek-and-poke.com]
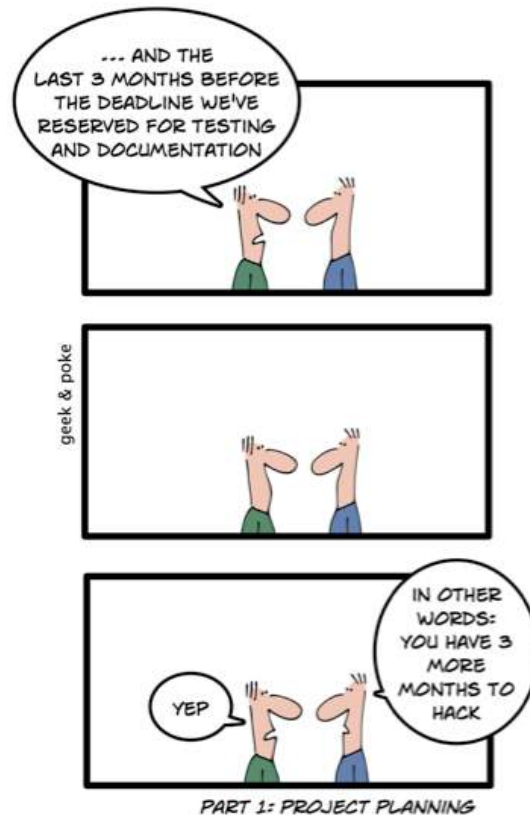
In project work in general it is important that you complete the plans of this week so that you don't fall behind and must do last week's work in addition to this week's work. Then there will be more and more to do every week. See Figure 8-3.
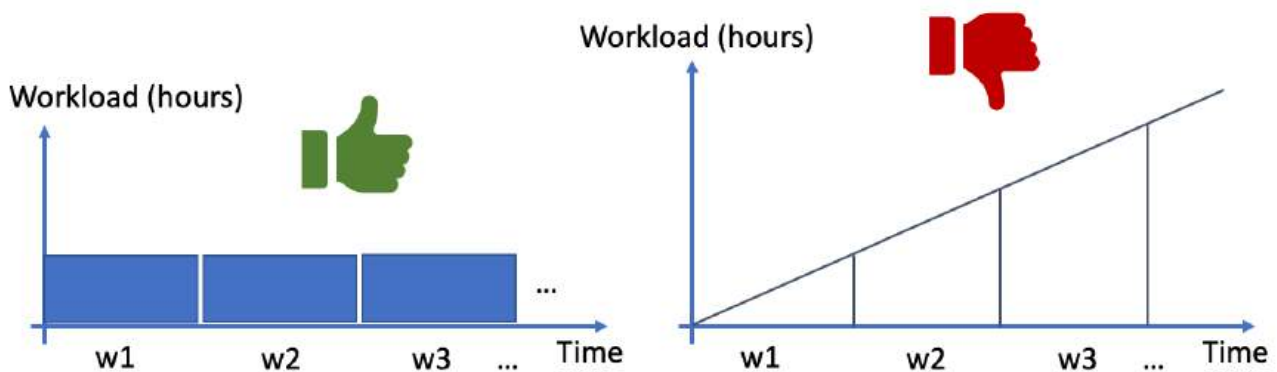
Figure 8-3: How to Work in a Project

8.2 Kick-off/Brainstorming

A Project should always start with a Kick-off meeting where a brainstorming session is important of that meeting.

During the brainstorming, you should:

- Involve all in the group

- Discuss what you are going to do in the project

- How are you going to solve the project?

- etc.

In addition to getting good ideas for solving the project, you should learn from previous projects.

Examples: Who are going to solve the different parts, what kind of Frameworks are you going to use, what kind of development tools you use, etc.

[http://geek-and-poke.com]

8.3 Software Development Plan (SDP)

Communication is the key to success! Below we list some examples how to avoid Communication Problems [12]:

- Listen to all with concentration
- Don't pre-judge
- Give all team members a turn
- See the value in every idea
- Don't make assumptions
- Ask questions to clarify
- When in doubt, communicate

A good idea is to create a Software Development Plan. The Software Development Plan gives an overview of all the communication within the project or within the team, i.e., what kind of communication, how the communication should be done, etc.

Examples of Communication:

- Meetings: The Team will meet every Monday from …
- Standards: Which Word processor, Templates, etc.
- E-mail… or other communication platforms, …
- Collaboration: How will you communicate? Work together on Tuesdays, …
- Other Tools: Microsoft Project, …
- etc.

The Software Development Plan typically includes the following sections:

1. **Introduction**: This briefly describes the objectives of the project and set out the constraints (e.g., budget, time, etc.) that affects the management of the project

2. **Project Organization**: This section describes how the development team is organized, the people involved and their roles in the team.

3. **Risk Analysis**

4. **Hardware and Software Resource Requirements**

5. **Work Breakdown** (WBS, Work Breakdown Structure): Break down the project into activities and identifies milestones

6. **Project Schedule**: Shows dependencies between activities, the estimated time required to reach each milestone, allocation of people to activities. (5) and (6) is typically done in a Gantt Chart (created in e.g., Microsoft Project)

7. **Monitoring and Reporting Mechanisms**: Definition of the Management Report that should be produced, when these should be produced, etc.

Other words for the Software Development Plan may be "Communication Plan" or "Project Plan".

A Software Development Plan (SDP) is all about the Internal Communication within the Development Team and how it Communicates with rest of the Organization, the Customers, etc.

8.3.1   Gantt Chart

One of the most used tools for project planning is the Gantt chart. The Gantt chart gives an overview of tasks, subtasks, milestones, resources, etc. in a project.

In Figure 8-4 we see a Gant Chart example created with Microsoft Project.
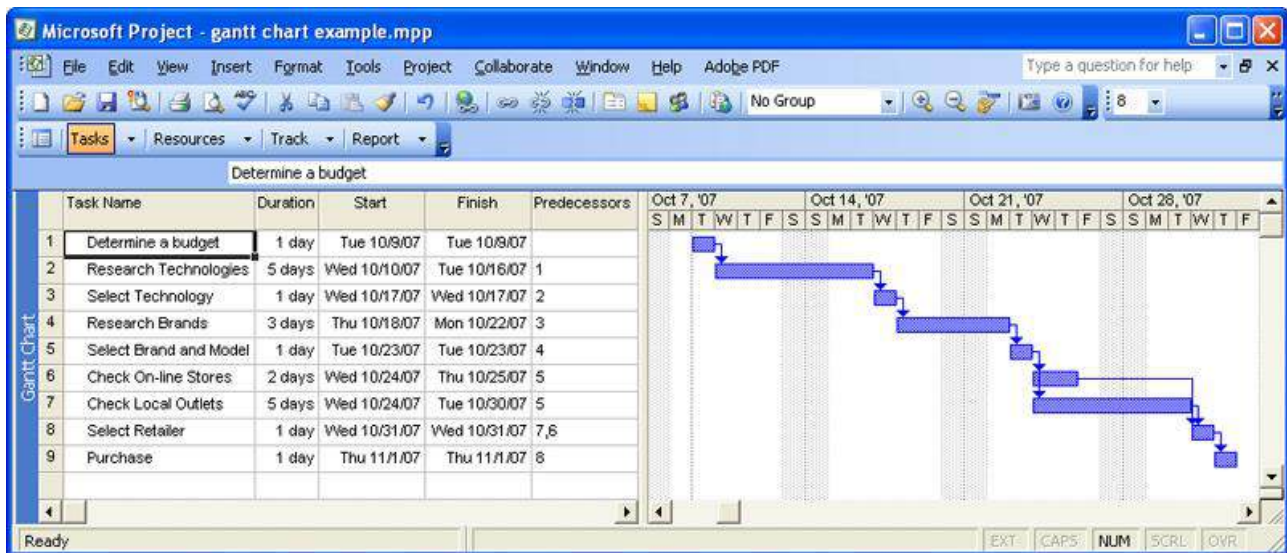
Figure 8-4: Gant Chart Example created with Microsoft Project

It is important that Project Management is an active part of your software project. The Gantt Chart should be used through the whole project; it is not something you create in the beginning of the project and put in a drawer.

In Figure 8-5 we see the recommended way of working with the different project activities.

How to work in the project period

Important: Work with these activities in parallel!!!
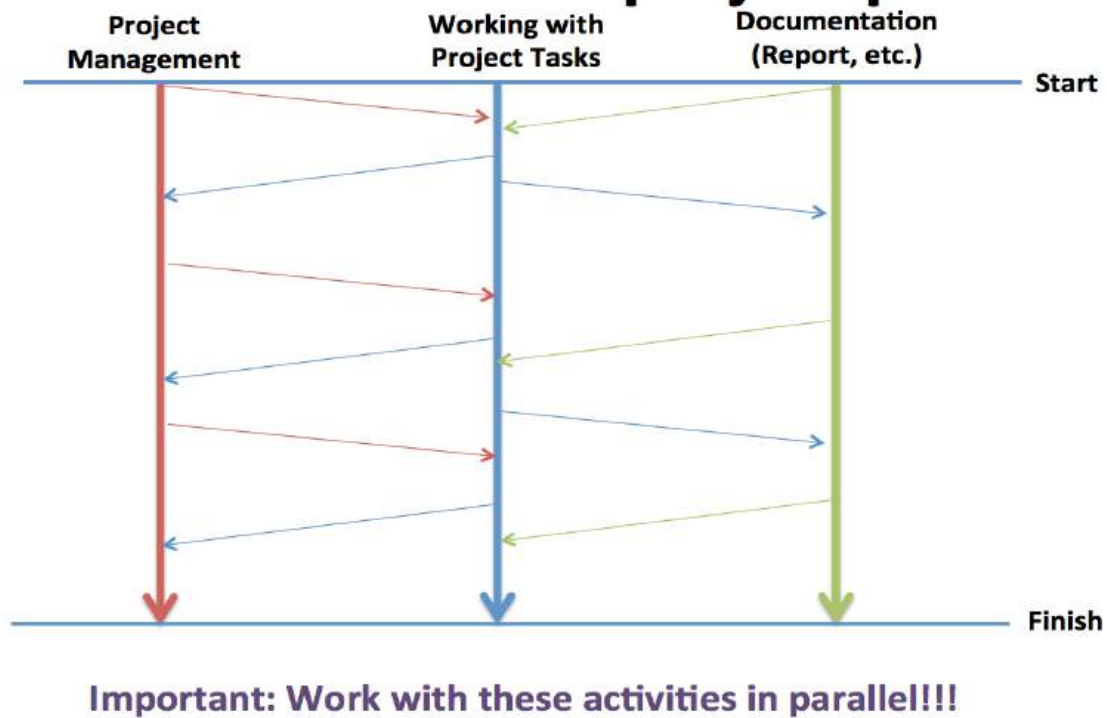
Figure 8-5: Project Work

Always create a Project Plan!

8.4 Meetings

It is necessary to have meetings when planning and creating software, but these meetings should not be misused.

Below we list some typical meetings needed during the software development project:

- Kickoff and Planning Meetings
- Project Meetings
- Daily Scrum Meetings
- Review Meetings
- Meetings for Planning the next Sprint/Iteration

For meetings in general we have the following guidelines:

- The meeting agenda should be clear.
- All meetings should follow the basic structure that is described for that meeting.
- Meetings should start on time, even if some team members are late.
- Meetings should finish on time.
- Each team member should come to the meeting prepared.

Always be prepared before the meetings (otherwise you don't need to be there)!

8.4.1   Meeting Agenda

A typical meeting agenda could be as follows:

- Project Plan, Gantt Chart (Project Manager)

- Work Items, Overview and Status (Test Manager)

- Demonstration of Applications/Coding (Individual)

- Short Status for each member (Individual)

    o What have you done so far?

    o What shall the focus be the next weeks?

    o Any Technical Challenges/Problems/Issues? (It is very important to get an overview of the challenges in the project, or else the whole project will be at risk if you don't talk about them!)

    o Other matters

- The meeting should last no longer than 60 minutes.

When you are finished with the meeting, write a short Minutes of Meeting as soon as possible.

8.4.2   Minutes of Meeting

Write a "Minutes of Meeting" (send on e-mail to team members and supervisor the same day!). The purpose of this is twofold:

- Important decisions or agreements are recorded, so they are not forgotten!
- The second purpose is to record unsolved issues that require follow up action, so-called action items. Each action item is assigned to one (preferred) or more team members with a specific deadline for completion.

Always create Minutes of Meeting!

The Minutes of Meetings should include a table like this:

Task	Responsible	Deadline
…	…	…
…	…	…

In this way, we can easily get an overview of the tasks agreed in the meeting, which is responsible for the tasks, and a specific deadline for each task. This task list should be followed up in the next meeting.

8.5 Agile Project Planning and Tracking

Successful projects often have the following characteristics:

- The needs of the customers drive the project.
- The team creates a high-level plan for delivering the project.
- The team develops the product over several iterations and refines the high-level plan over time.
- The team has effective tools for adapting to changes that occur.

Figure 8-6 shows the steps involved in Agile Project Planning and Tracking.
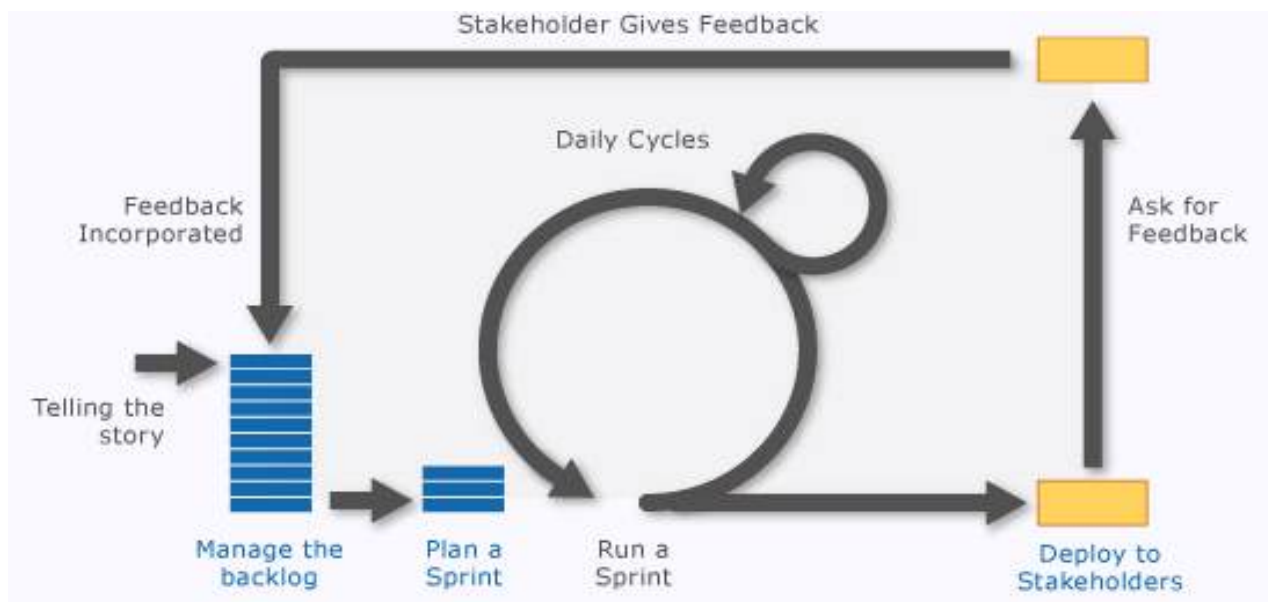
Figure 8-6: Agile Project Planning and Tracking

In Agile Project Planning and Tracking, everything is broken down to so-called iterations, as shown in Figure 8-7.
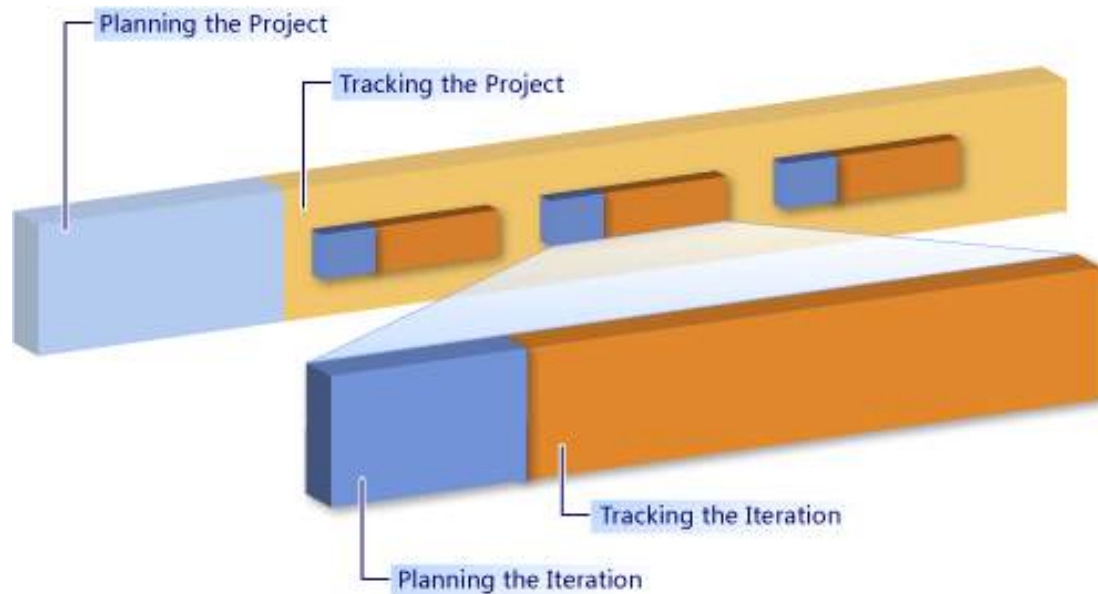
Figure 8-7: Using Iterations in Agile Project Planning and Tracking

The Task board is a key tool in Agile Project Planning and Tracking, see Figure 8-8
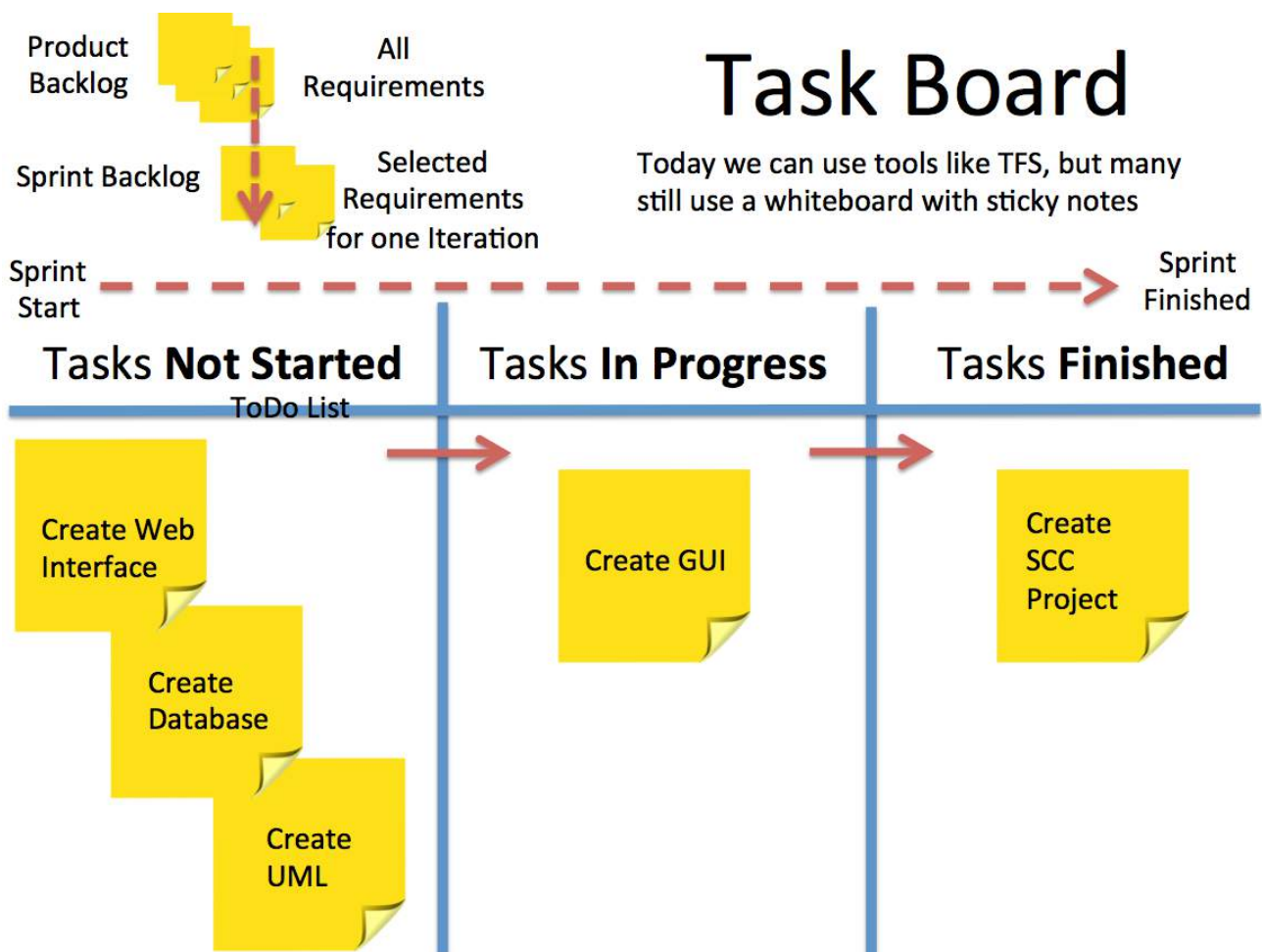
Figure 8-8: Task board used in Agile Software development

The Task board is used together with the Burndown chart, as shown in Figure 8-9.
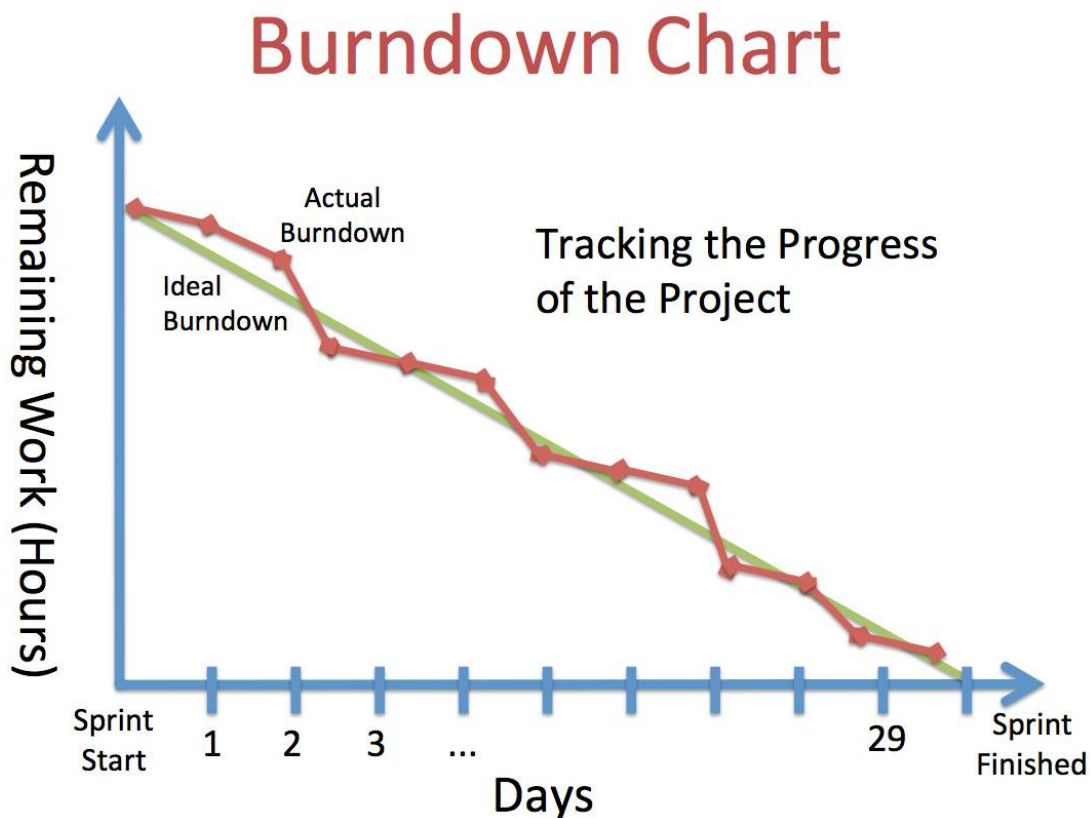
Burndown Chart

Figure 8-9: Burndown Chart

A burn down chart is a graphical representation of work left to do versus time. The outstanding work (or backlog) is often on the vertical axis, with time along the horizontal. That is, it is a run chart of outstanding work. It is useful for predicting when all the work will be completed.

It is often used in agile software development methodologies such as Scrum. However, burn down charts can be applied to any project containing measurable progress over time.

Azure DevOps have all these features (Task board, Burndown chart, etc.) built in.

Azure DevOps is explained in detail in a later chapter.

8.6 Microsoft Teams

Microsoft Teams is a great tool for teamwork, collaboration and project work. Here you can chat, share documents, have online meetings, etc. Microsoft Teams is also a good tool for Project Planning and Management by using, e.g., Microsoft Planner.

Microsoft Planner App in Microsoft Teams: https://youtu.be/LrZK3oUgkL4

8.7 Summary

Here are some important keywords for successful project management:

- Software Project Management is important to keep the project on track

- Agile Project Management = less documentation

- Useful tools are in Project Management and Tracking are Gantt Chart, Task board, Burndown Chart

- You should always create a Communication Plan

- You should always start the project with a Brainstorming session.

9 Requirements Engineering

Before you start to implement a software system, you need to understand what the system is intended to do. This intended functionality is the "Requirements". The process of creating these requirements is called Requirement Analysis or Requirement Engineering. It is the process of understanding what you want and what you need in your software.

Requirements Engineering (RE) refers to the process of formulating, documenting, and maintaining software requirements.

The results of the Requirement Analysis or Requirement Engineering process is normally one or more documents, called the Software Requirement Specification (shorted "SRS").

The requirements are in some cases created by the customer, at least the overall requirements (it defines "What" the customers want), while more details are normally created by architects and developers in the software company that is going to develop the actual software. Here we can have both "What" and "How" the software shall be designed or implemented.

The main challenges in Requirements Engineering are that the customers most often don't know what they want or are not qualified to know what they need.

In general, we can summarize the following:

- Stakeholders don't know what they want.
- Stakeholder's express requirements in their own terms.
- Different stakeholders may have conflicting requirements.
- Organizational and political factors may influence the system requirements.
- The requirements change during the analysis process. New stakeholders may emerge, and the business environment may change.

In Figure 9-1 we see different types of requirements.
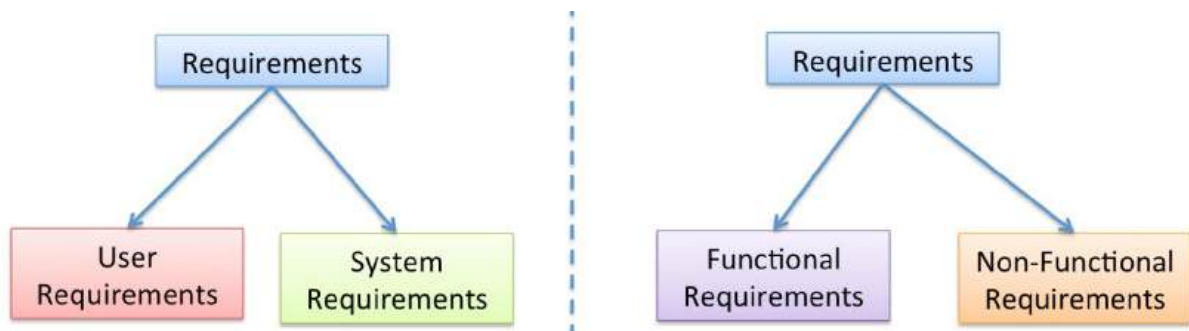
Figure 9-1: Software Requirements

In Figure 9-2 and below we explain the different software requirements categories in more detail.
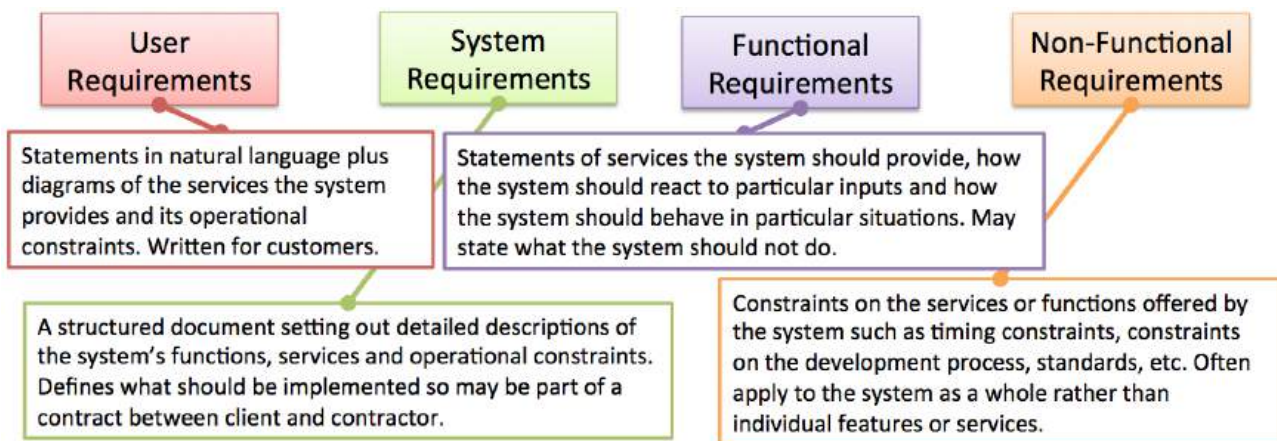
Figure 9-2: Software Requirements Categories Overview

User Requirements

Statements in natural language plus diagrams of the services the system provides and their operational constraints. Written for customers.

System Requirements

A structured document setting out detailed descriptions of the system's functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.
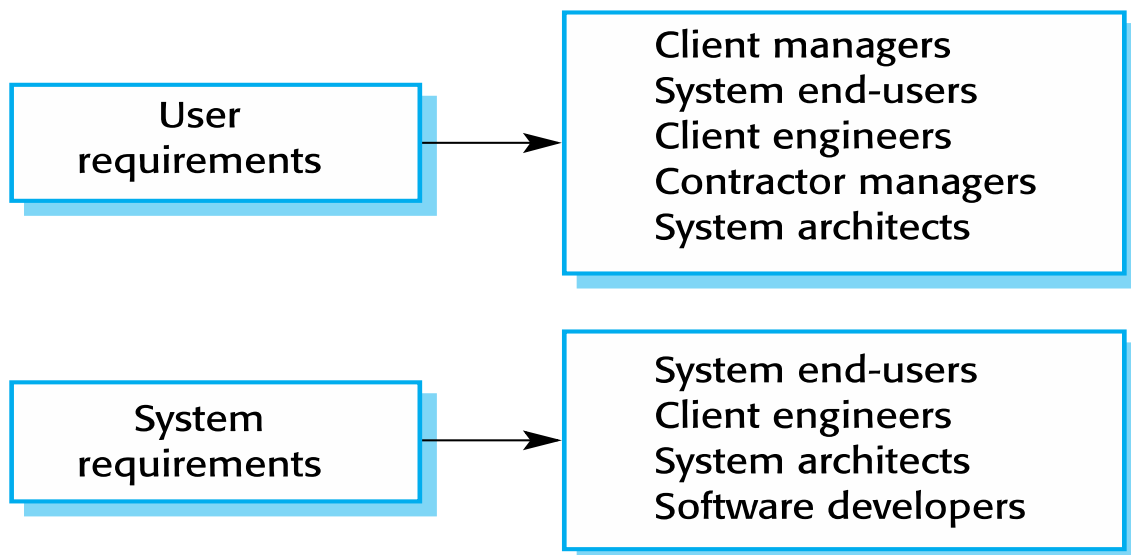
Figure 9-3: Requirements Users [1]

Functional Requirements

Statements of services the system should provide, how the system should react to inputs and how the system should behave in particular situations.

May state what the system should not do.

Non-Functional Requirements

Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.

They often apply to the system rather than individual features or services.

Since there are different people involved creating and reading the requirements documents, the requirements are normally split into "High-Level Requirements" and "Detailed Requirements" (Figure 9-4).
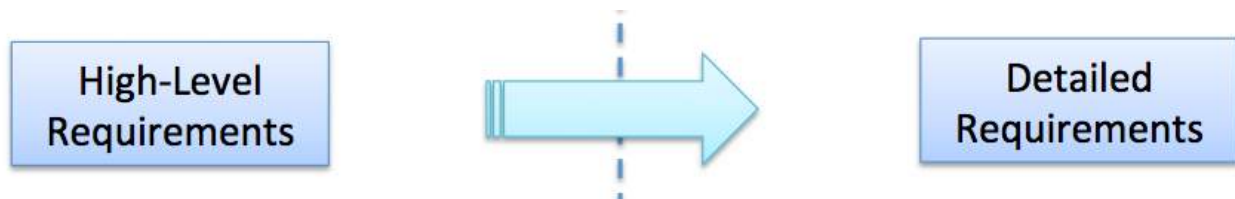
Figure 9-4: High-Level Requirements vs. Detailed Requirements

High-level requirements are for "business" people, while detailed requirements are for developers, etc.

9.1 User Requirements

User requirements are statements in natural language plus diagrams of the services the system provides and its operational constraints. User requirements are written for customers.

9.2 System Requirements

System requirements is setting out detailed descriptions of the system's functions, services and operational constraints. They define what should be implemented may be part of a contract between client and contractor.

9.3 Functional Requirements

Functional Requirements are:

- Describe functionality or system services.

- Depending on the type of software, expected users and the type of system where the software is used.

- Functional user requirements may be high-level statements of what the system should do.

- Functional system requirements should describe the system services in detail.

9.4 Non-Functional Requirements

Non-Functional Requirements are:

- These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.

- Process requirements may also be specified using an IDE, programming language or development method.

- Non-functional requirements may be more critical than functional requirements. If these are not met, the system may be useless.

9.5 SRS

Software Requirements Specifications (SRS) are:

- The software requirements document is the official statement of what is required of the system developers.

- It should include both a definition of user requirements and a specification of the system requirements.

- It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it.

In practice, requirements and design are inseparable. Many don't separate SRS and SDD (Software Design Document) documents but include everything in a Requirements & Design Document. Such a document could be called "Software Requirements and Design Document" (SRD).

In Figure 9-5 we see some typical contents of such a SRS/SDD document.
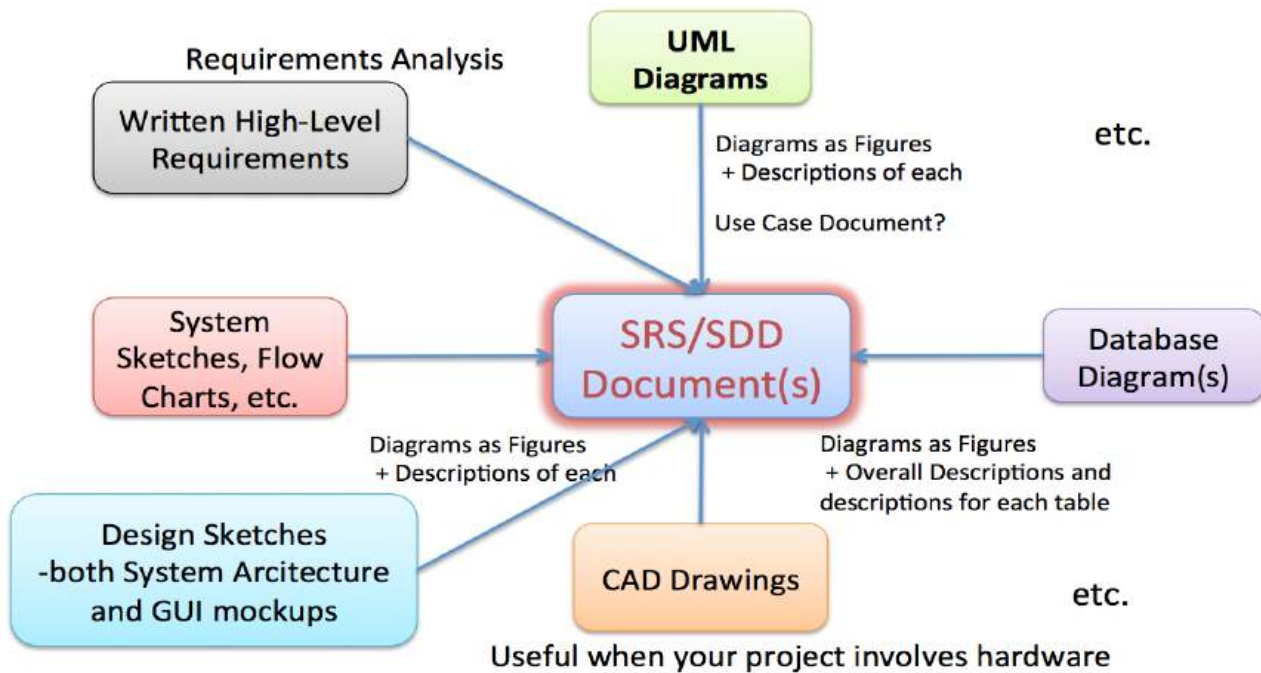
Figure 9-5: Typical SRS/SDD Contents

In Figure 9-6 we see the different users involved in the SRS document.
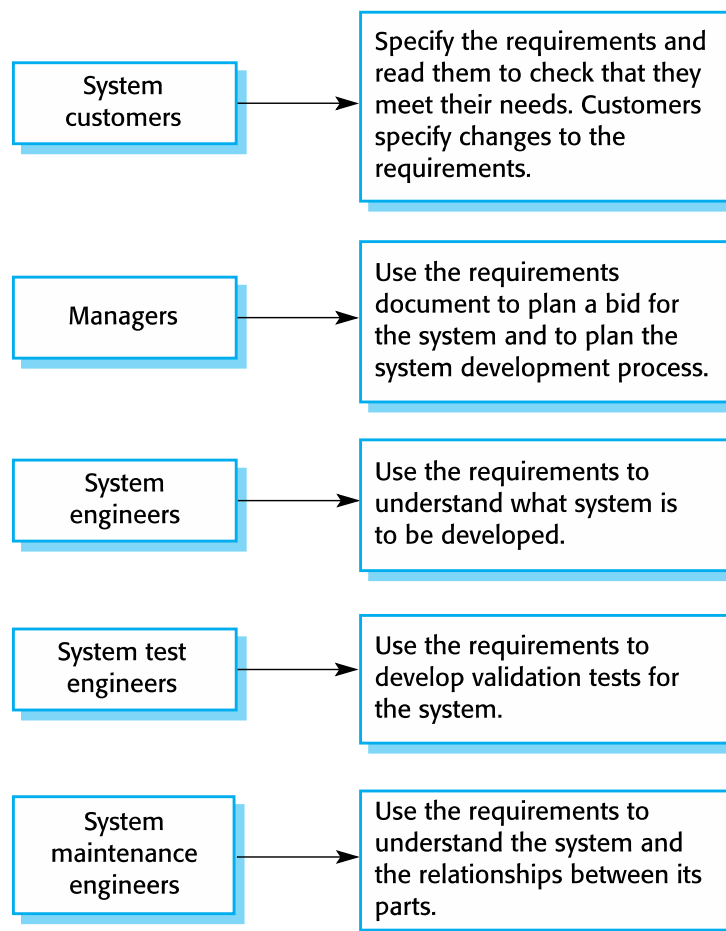
Figure 9-6: Users of SRS [1]

In Table 9-1 we see an example of what chapters that we can include in an SRS document.

Table 9-1: Example of Table of Contents for the SRS document [1]

Chapter	Description
Preface	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This should describe the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.
Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
User Requirements Definition	Here, you describe the services provided for the user. The nonfunctional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.
System Architecture	This chapter should present a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.
System Requirements Specification	This should describe the functional and nonfunctional requirements in more detail. If necessary, further details may also be added to the nonfunctional requirements. Interfaces to other systems may be defined.
System Models	This might include graphical system models showing the relationships between the system components and the system and its environment. Examples of possible models are object models, data-flow models, or semantic data models.

System Evolution	This should describe the fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs, and so on. This section is useful for system designers as it may help them avoid design decisions that would constrain likely future changes to the system.
Appendices	These should provide detailed, specific information that is related to the application being developed, for example, hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organization of the data used by the system and the relationships between data.
Index	Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, and so on.

9.6 Project Estimation

Getting an overview of the total cost in a software project is important. The features and requirements need to be broken down into manageable tasks by the team. Each Tasks then needs to be estimated (Hours).

9.7 Exercises

Make sure to discuss and reflect over the following:

1. What are Software Requirements?

2. Requirements vs. Design – What is the main difference?

3. Different types of Requirements

4. What are User Requirements?

5. What are System Requirements?

6. What are Functional Requirements?

7. What are Non-Functional Requirements?

8. Give some examples of Non-Functional Requirements

9. What is SRS?

10. Mention some Requirements Analysis Problems/Challenges?

10 User eXperience (UX)

Designing and creating the graphical user interface is a very important part of software development. We have different names for it; User eXperience (UX), Graphical User Interface (GUI) or Human Machine Interface (HMI).

GUI design has been in constant change since the first computers and software were created. In Figure 10-1 we see the difference between Windows 1 and Windows 8.
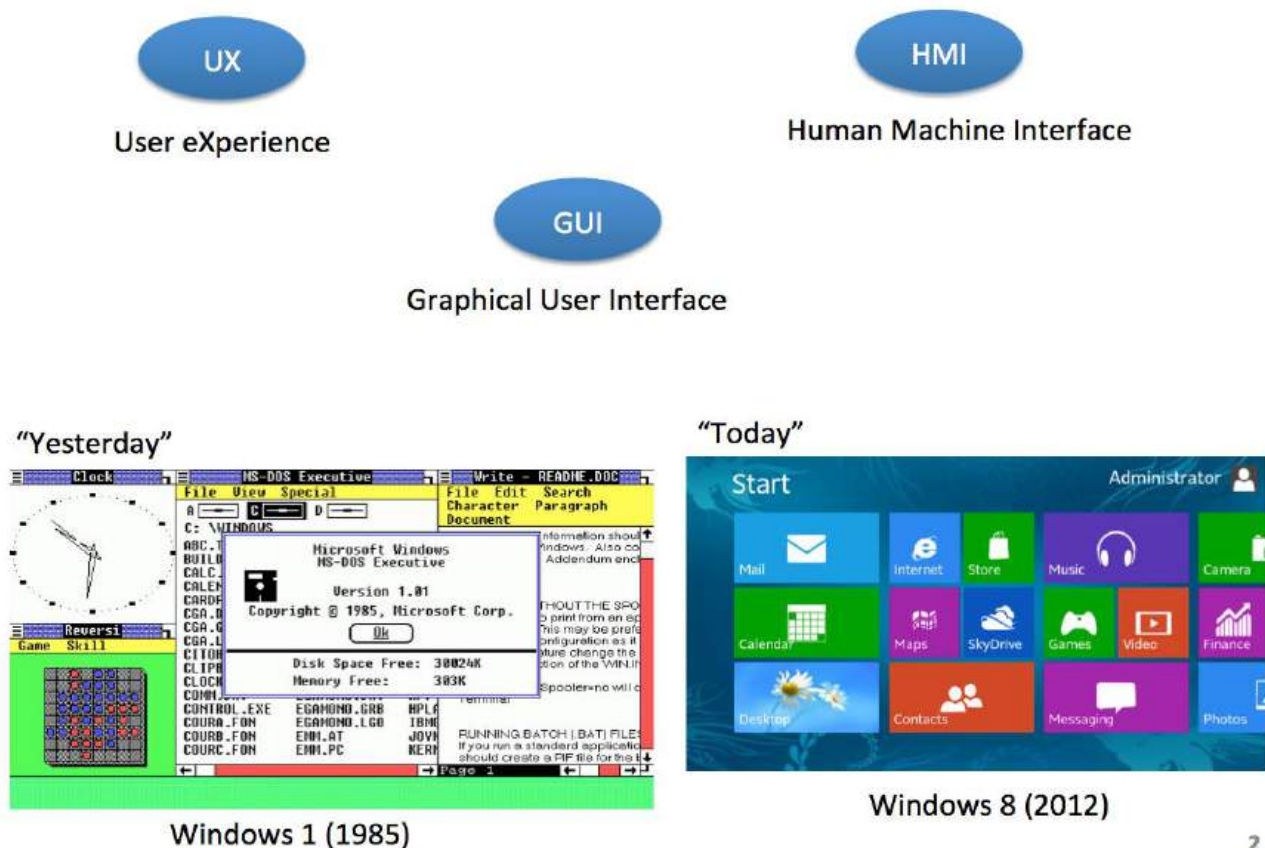
Figure 10-1: Examples of User eXperience – UX

It is important that Documents, GUI, Code, etc. have the same "Look and Feel" – Use Common Templates, APIs, etc.

In software design, look and feel is a term used in respect of a graphical user interface and comprises aspects of its design, including elements such as colors, shapes, layout, and typefaces (the "look"), as well as the behavior of dynamic elements such as buttons, boxes, and menus (the "feel"). The term can also refer to aspects of an API, mostly to parts of an API that are not related to its functional properties.

Look and feel in operating system user interfaces serve two general purposes. First, it provides branding, helping to identify a set of products from one company. Second, it increases ease of use, since users will become familiar with how one product functions (looks, reads, etc.) and can translate their experience to other products with the same look and feel.

It is the "UX Designer" that design the GUI, while the Programmer make sure to implement it in the proper programming language (Figure 10-2).
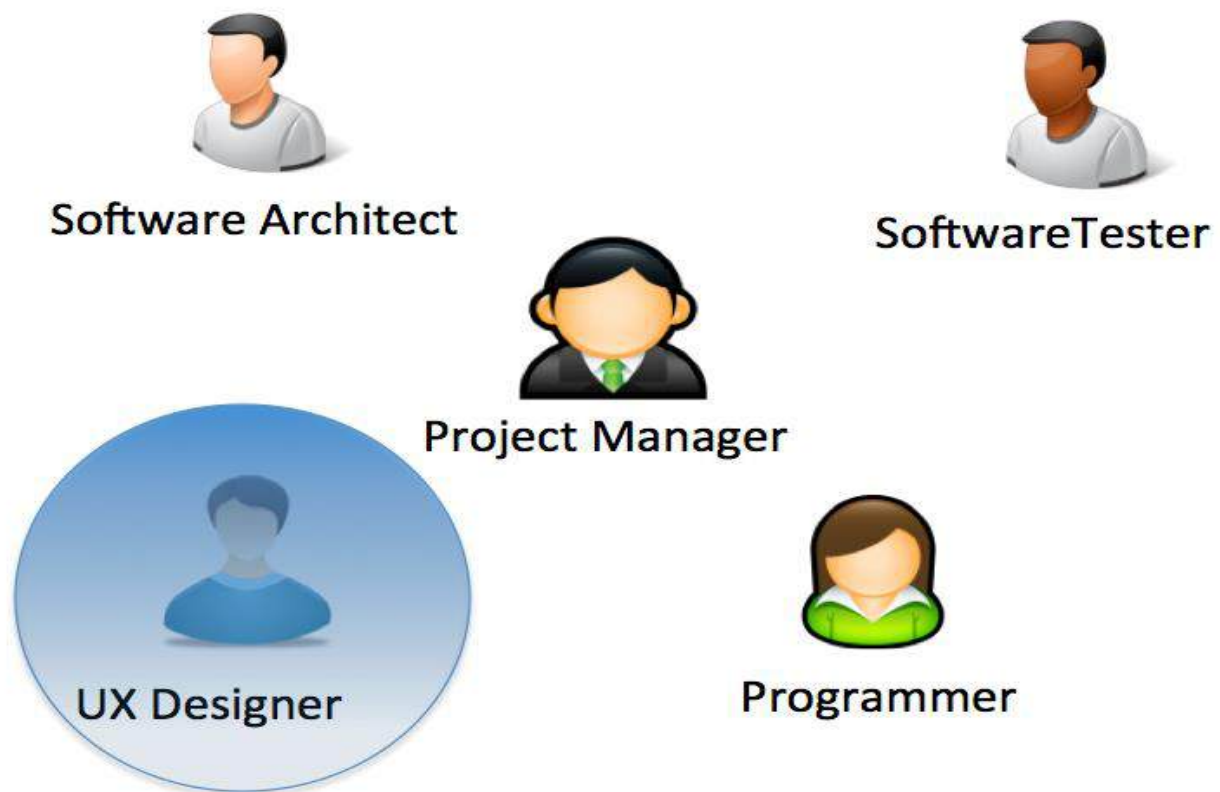
Figure 10-2: UX Designer

10.1   UX Guidelines

Different platforms have different UX and UX guidelines, so it is important to follow these general guidelines for the different platforms. The GUI is totally different on, e.g., Windows and Mac OS X.

The different platform vendors create their own guidelines that the developers should follow.

For Windows 8 UX Guidelines, see [13].

Figure 10-3 is an example from the Mac OS X UX Guidelines [14].
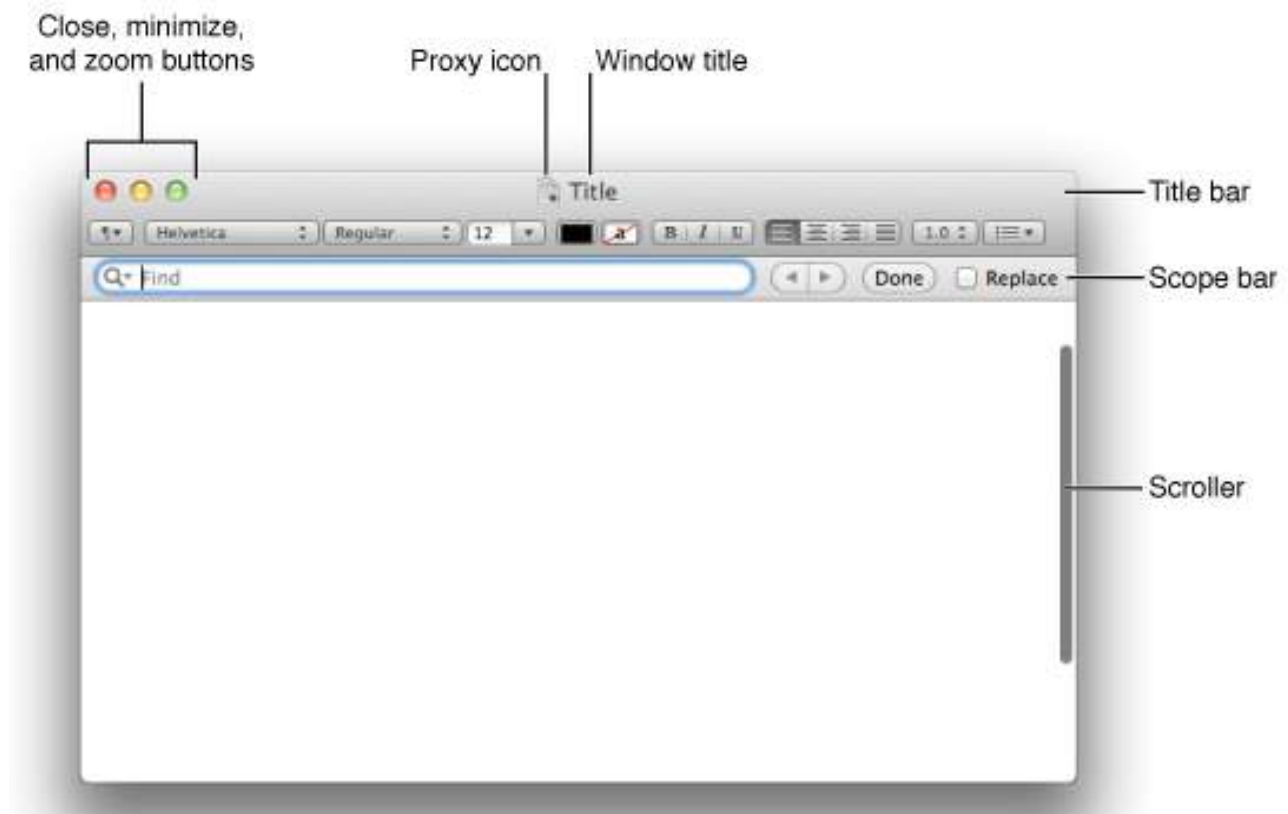
Figure 10-3: Mac OS X UX Guidelines [14]

10.2   GUI Mockup

Creating so-called GUI mockups is an important part of the process of creating user-friendly GUI.

Mockups and prototypes are not so cleanly distinguished in software and systems engineering, where mockups are a way of designing user interfaces on paper or in computer images. A software mockup will thus look like the real thing but will not do useful work beyond what the user sees. A software prototype, on the other hand, will look and work just like the real thing. In many cases it is best to design or prototype the user interface before source code is written or hardware is built, to avoid having to go back and make expensive changes [15].

In Figure 10-4 we see an example of a GUI Mockup.

Figure 10-4: GUI Mockup Example

10.3   Creativity

Be creative - Think outside the box!

"Thinking outside the box" is a metaphor that means to think differently, unconventionally, or from a new perspective.

11 UML

11.1 Introduction

UML is a modeling language used in software engineering. It is very popular among OOA, OOD, OOP. UML was developed in the 1990s and adapted as an ISO standard in 2000. UML 2.2 has 14 different types of diagrams. See Figure 11-1.
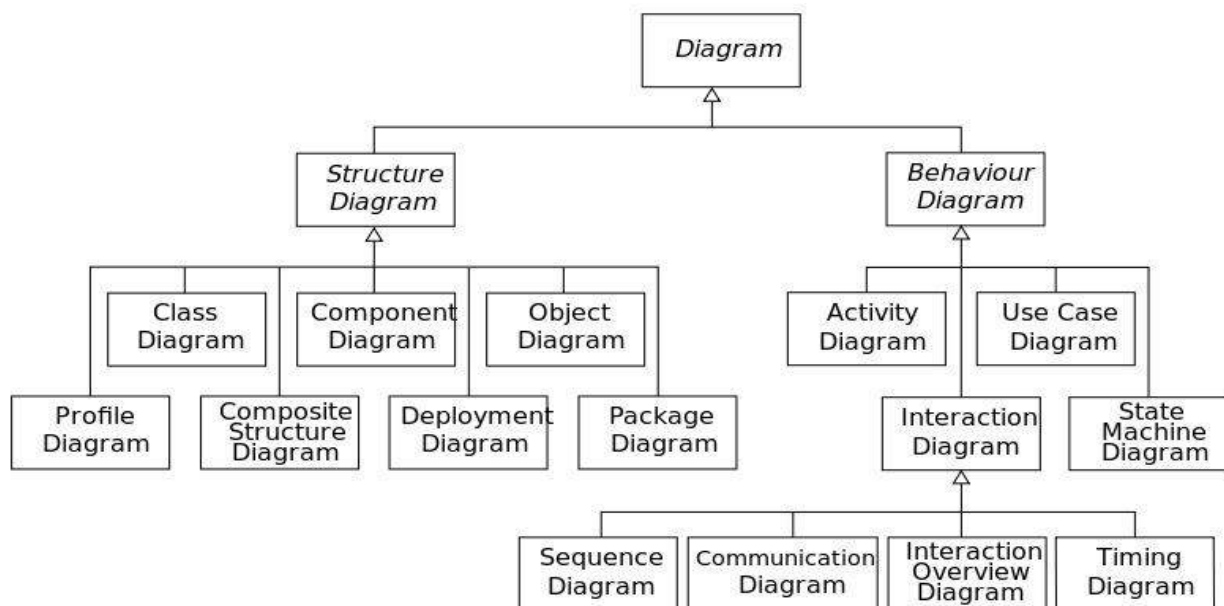
Figure 11-1: UML Diagrams

We have 2 main categories of diagrams:

- Structure Diagrams
- Behavior Diagrams
    - Interaction Diagrams (subcategory of Behavior Diagrams)

The diagrams available in UML are:

- **Class Diagram**
- Component Diagram
- Deployment Diagram
- Object Diagram
- Package Diagram
- Activity Diagram

- **Sequence Diagram**
- Communication Diagram
- **Use Case Diagram**
- **State Machine Diagram**
- Composite Structure Diagram
- Interaction Overview Diagram
- Timing Diagram

Why use UML?

- Design:
  - Forward Design: doing UML before coding. Makes it easier to create the code in a structured manner
  - Backward Design: doing UML after coding as documentation
- Code
  - Some tools can auto-generate Code from UML diagrams

11.2   UML Software

There exist hundreds of different software for creating UML diagrams, here I mention just a few:

- Enterprise Architect
- StarUML
- Diagram tools like Lucidchart, Miro, Draw.io, etc.
- ++ hundreds of other alternatives

Some of these software tools are free to use while others cost money. More about UML software later in this document.

11.3   Use Case

One of the most used UML diagrams is the Use Case Diagram.

In Figure 11-2 we see a Use Case example.
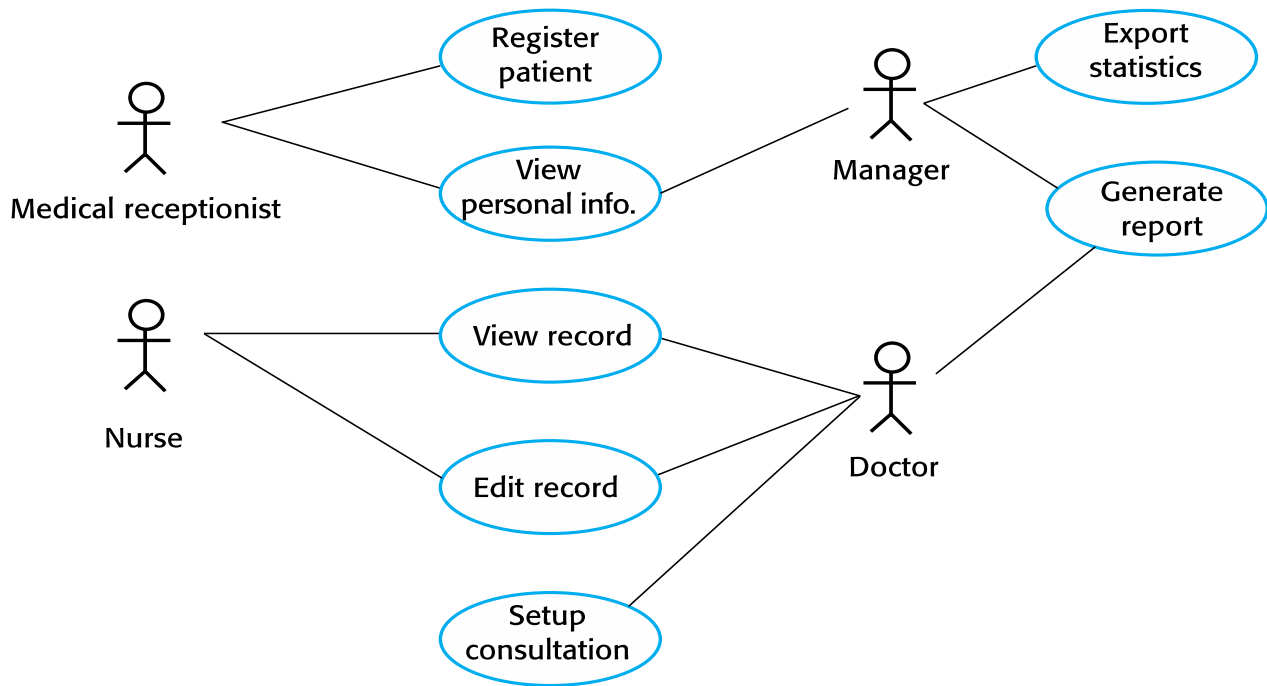
Figure 11-2: Use Case Example

11.4   Sequence Diagram

In Figure 11-3 we see an example of a Sequence Diagram.
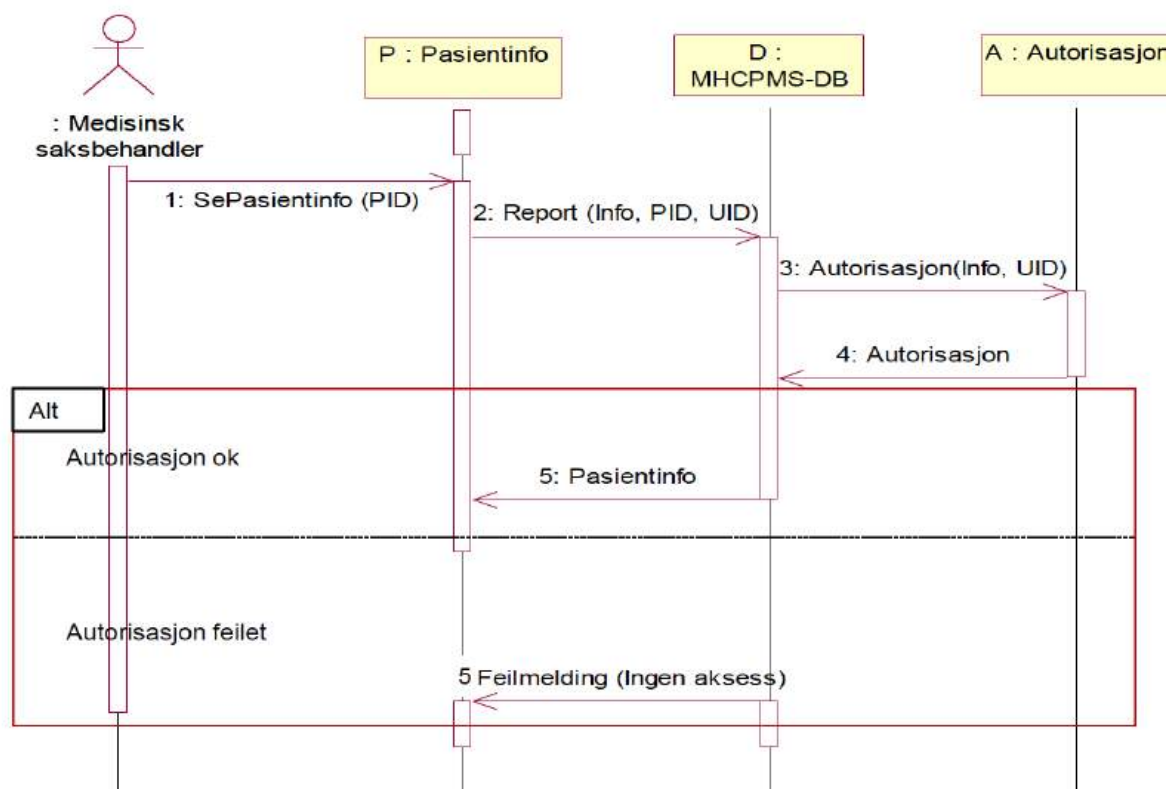
Figure 11-3 Sequence Diagram Example

11.5   Class Diagram

Figure 11-4 shows a Class Diagram Example.
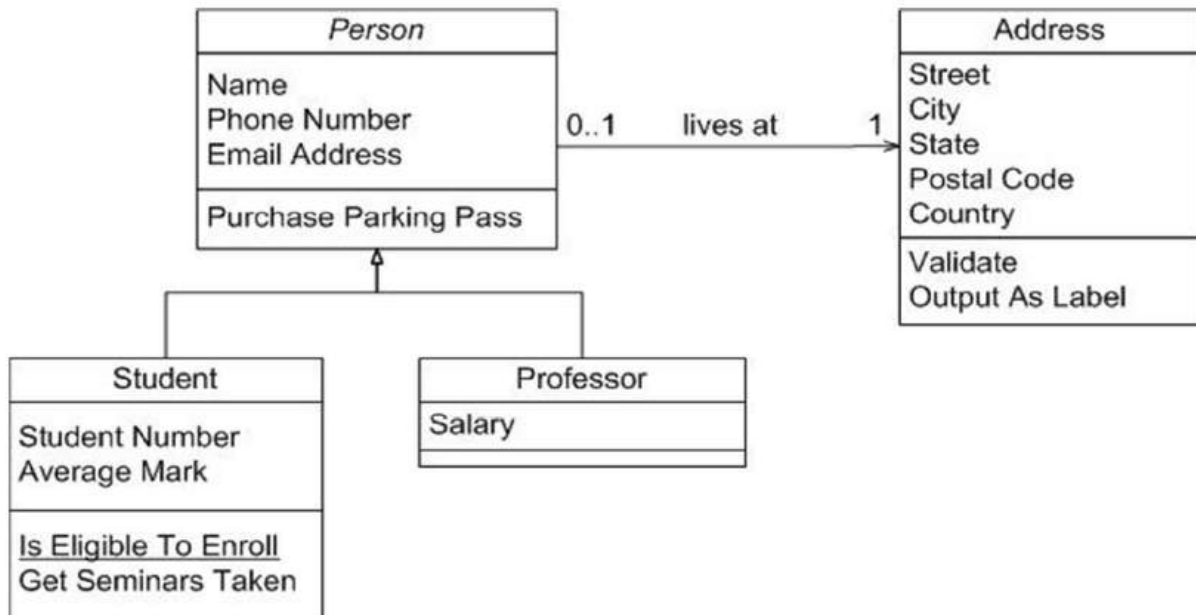
Figure 11-4: Class Diagram Example

11.6   Creating UML Diagrams

There are many types of UML diagrams, so you need to focus in some of the diagram types which are relevant for your project.

I will focus on the UML diagrams mentioned above, namely Use Case Diagrams, Sequence Diagrams and Class Diagrams.

Requirements Analysis Phase (WHAT):

- **Use Case Diagrams**

Design Phase (HOW):

- **Sequence Diagrams** (Typically one Sequence diagram for each Use Case)

- **Class Diagrams** (just one Class diagram in total)

See Figure 11-5 for the recommended approach when writing UML diagrams.

Creating UML - A practical Approach

①

Written Requirements

Create Use Cases from your written Requirements

②

A graphical/visual representation of the Requirements

Use Case Diagrams

Interactions between a system and its environment

WHAT

WHAT the system shall do

DESIGN

④

Class Diagram

You get the Class Names from the different Sequence Diagrams

HOW

Create one Class Diagram that gives an overview of all your classes and the relationship between them

③

Sequence Diagrams

HOW

Typically create one Sequence Diagram for each Use Case

Figure 11-5: How to create UML diagrams

Finally, include your UML diagrams and descriptions of them in the SRS/SDD document(s), see Figure 11-6.

Requirements Analysis

Written High-Level Requirements

UML Diagrams

Diagrams as Figures + Descriptions of each

etc.

Use Case Document?

System Sketches, Flow Charts, etc.

SRS/SDD Document(s)

Database Diagram(s)

Diagrams as Figures + Descriptions of each

Diagrams as Figures + Overall Descriptions and descriptions for each table

Design Sketches -both System Arcitecture and GUI mockups

CAD Drawings

etc.

Useful when your project involves hardware

Figure 11-6: UML documentation

11.7   UML in Agile/Scrum?

UML is not a part of the Agile/Scrum methodology, because they use another philosophy with less focus on documentation.

Use Case and Scrum (Agile):

- The Team works closely together with the Product Owner

- Less need for detailed descriptions and requirements

- Agile/Scrum uses **User Stories** instead (which could be considered as a light version of Use Case)

- The User Stories are the base for the Product Backlog and the Sprint Backlog

11.8   Summary

You should create Design and Specifications (including UML) before you start Coding. UML diagrams is a general method/standard to do just that. But UML can also be used to document your code afterwards (so-called Reverse Engineering).

UML makes it easier to create structured code and an effective way to document your code properly. UML should also be part of the code refactoring process and UML should be as part of the continuous code improvements process.   Note! If you update the code, make sure to update the UML and vice versa!

Make sure that you code reflects the UML design regarding classes, etc. If you update your code, you need to update the UML diagrams and vice versa.

It is important that we always have working software (so it can be reviewed, tested, etc.)!

11.9   Exercises

Make sure to discuss and reflect over the following:

1. What is UML?

2. Give example of some types of UML diagrams (in total we have 14 different types)?

3. Give examples of software used to create UML diagrams

4. List the 2 different categories of UML diagrams we have

5. Create a Class Diagram for a typical School including Classes Teacher, Student, Course, Grade, etc.

12   Software Implementation

The goal of most software engineering projects is to produce a working program.

The act of transforming the detailed design into a valid program in some programming language, together with all its supporting activities, is referred to as implementation.

Most of the text in this chapter is taken from [16].

The implementation phase involves more than just writing code. Code also needs to be tested and debugged as well as compiled and built into a complete executable product (Figure 12-1).

We usually need to use a Source Code Control (SCC) tool to keep track of different versions of the code.
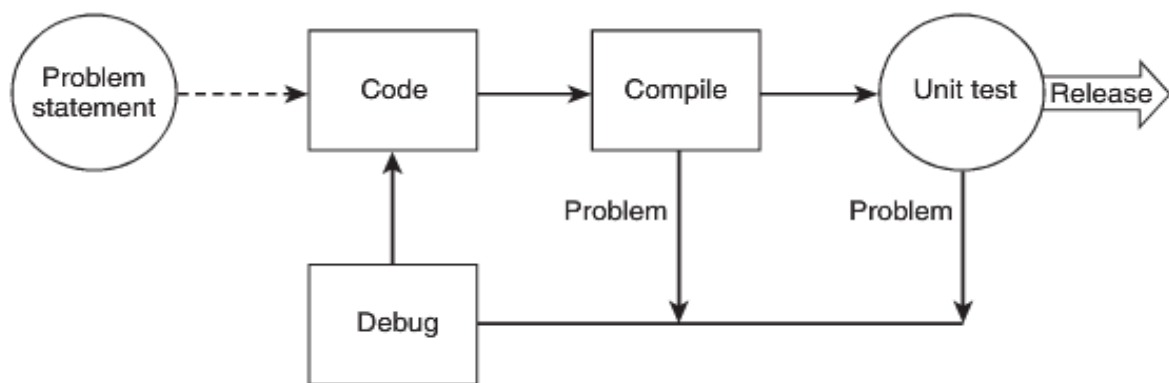
Figure 12-1: Software Implementation [16]

In many cases the detailed design is not done explicitly (in the Design Phase) but is left as part of the implementation. Doing the detailed design as part of the implementation is usually faster, but it may result in a less cohesive and less organized design, because the detailed design of each module will usually be done by a different person.

In small projects, the detailed design is usually left as part of the implementation. In larger projects, or when the programmers are inexperienced, the detailed design will be done by a separate person.

Here are some keywords for good implementation:

- **Readability**: The code can be easily read and understood by other programmers.
- **Maintainability**: The code can be easily modified and maintained. Note that this is related to readability, but it is not the same; for example, this involves the use of e.g., Hungarian notation, in which variable names include abbreviations for the type of variable.

- **Performance**: All other things being equal, the implementation should produce code that performs as fast as possible.
- **Traceability**: All code elements should correspond to a design element. Code can be traced back to design (and design to requirements).
- **Correctness**: The implementation should do what it is intended to do (as defined in the requirements and detailed design).
- **Completeness**: All the system requirements are met.

In this chapter, we will go through the following topics regarding implementation:

In this chapter, we will discuss the following:

- Programming Style and Coding Guidelines

- Comments

- Debugging

- Code Review

- Refactorization

12.1 Programming Style & Coding Guidelines

Almost all software development organizations have some sort of coding guidelines. These guidelines usually specify issues such as naming, indentation, and commenting styles, etc.

It is strongly recommended that you be consistent in your notation to avoid confusion when others are debugging or maintaining your code later. Especially in large software projects there are usually some programming conventions. These conventions may seem to be of little value at first, but they may become extremely helpful during the maintenance of the code.

Here are some recommendations:

- **Naming**: This refers to choosing names for classes, methods, variables, and other programming entities.
- **Separating words and capitalization**: Many times, a name will be composed of more than one word. In human languages, we use spaces to separate words, but most programming languages will not allow us to do so. ("do_something", "doSomething", "DoSomething")
- **Indentation and spacing**: Indentation refers to adding horizontal spaces before some lines to better reflect the structure of the code. Spacing refers to both spaces and blank lines inserted in the code.
- **Function/method size**: Many studies have shown that large functions or methods are statistically more error-prone than smaller ones.

- **File-naming issues**: Having a standard for specifying how to name the files, which files to generate for each module, and how to locate a given file from a module is very advantageous.
- **Specific programming constructs**: Different programming languages support different features; although they usually have good reasons to include certain features, there are many that can be misused and need special precautions.

12.1.1   Naming Convention

We have different naming convention/notation such as:

- Camel notation
- Pascal notation
- Hungarian notation

Camel Notation

For variables and parameters/arguments

Examples: "myCar", "backColor"

Pascal Notation

For classes, methods, and properties

Examples: "ShowCarColor"

Hungarian Notation

For controls on your user interface, we either use "Pascal notation" or "Hungarian notation", but stick to one of them!

Examples: "txtName", "lblName"

Acronyms

Examples: "DBRate", "ioChannel", "XmlWriter", "htmlReader"

12.2   Comments

Comments are very important and can significantly aid or hurt readability and maintainability.

There are two main problems with comments:

- they may distract from the actual code and make the program more difficult to read and

- they may be wrong.

[http://geek-and-poke.com]
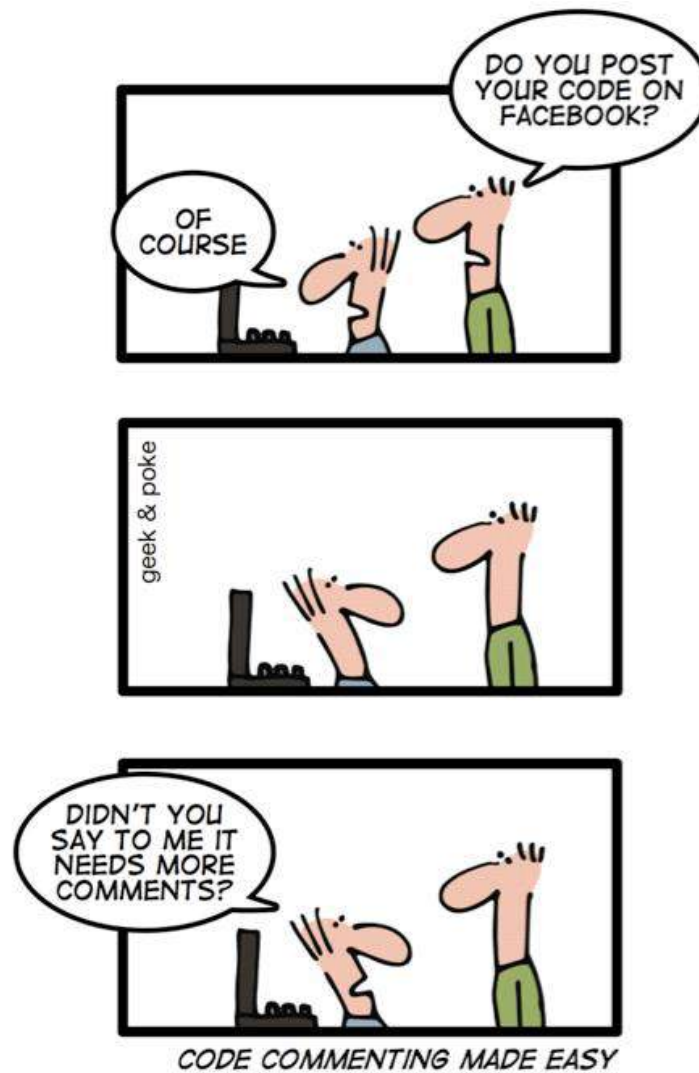
We may classify comments into 6 different types:

1. Repeat of the code

2. Explanation of the code

3. Marker in the code

4. Summary of the Code

5. Description of the code intent

6. External references

These are explained below:

- Repeat of the code

- o These kinds of comments tend to be made by novice programmers and should be avoided.
  Bad Example:
  // increment i by one
  i++;

- Explanation of the code

    - o Sometimes, when the code is complex, programmers are tempted to explain what the code does in human language.

    - o In almost every case, if the code is so complex that it requires an explanation, then it should be rewritten.

1. Marker in the code

    – It is common practice to put markers in the code to indicate incomplete items, opportunities for improvement, and other similar information.

    – We recommend using a consistent notation for these markers and eliminating all of them before the code is in production.

    – Sometimes programmers put markers in the code to keep track of changes and who made them. We believe that information is better tracked with version management software and recommend doing so.

3. Summary of the code

    – Comments that summarize what the code does, rather than just repeating it, are very helpful in understanding the code, but they need to be kept up to date.

    – It is important to ensure that these comments are summarizing the code, not just repeating, or explaining it.

    – In many cases, the code that is being summarized can be abstracted into its own function, which, if named correctly, will eliminate the need for the comment.

4. Description of the code intent

    – These are the most useful kinds of comments; they describe what the code is supposed to do rather than what it does.

    – These are the only kinds of comments that override the code. If the code does not fulfill its intent, then the code is wrong.

5. External references

- These are comments that link the code to external entities, usually books or other programs.

- Many times, these can be viewed as a kind of intent statement, as in, "This function implements the XYZ algorithm, as explained in . . .," but we believe such comments require special attention.

- There may also be external prerequisites for the code, such as the existence of initializing data in the database tables.

The trade-off that comments imply should be recognized. Comments can help clarify code and relate it to other sources, but they also represent some level of duplication of the code.

6. Effort is invested in their creation and, above all, in their maintenance.

7. A comment that does not correspond to the actual code that it accompanies can cause errors that are very hard to find and correct.

8. Another danger comments present is that they can be used to justify bad coding practices. Many times, programmers will be tempted to produce code that is too complicated or too hard to maintain, and add comments to it, rather than rewrite it to good standards.

9. In fact, many experts recommend avoiding comments completely and produce what is called "self-documented code" - that is, code that is so well written that it does not need any documentation.

10. Comments have their place, especially in the form of describing the programmer's intent.

12.3   Debugging

Debugging is about different techniques for finding and fixing bugs (errors that make your code not work as expected) in your code.

1. It is difficult to write code without errors (bugs), but e.g., Visual Studio and other tools have powerful Debugging functionality (breakpoints, etc.)

2. The Compiler will also find syntax errors, etc.

3. For more "advanced" bugs other methods are required (Unit Testing, Integration Testing, Regression Testing, Acceptance Testing, etc.).

4. The focus here will be on these methods, while Debugging is something you learned in Programming courses.

In debugging we have 4 phases:

- Stabilization/Reproduction

    - The purpose of this phase is to be able to reproduce the error on a configuration, and to find out the conditions that led to the error by constructing a minimal test case

- Localization

    - The process of localization involves finding the sections of the code that led to the error. This is usually the hardest part, although, if the stabilization phase produces a very simple test case, it may make the problem obvious.

- Correction

    - The process of correction involves changing the code to fix the errors. Hopefully, if you understand what caused the error, you have a good chance of fixing the problem.

- Verification

    - The process of verification involves making sure the error is fixed, and no other errors were introduced with the changes in the code. Many times, a change in the code will not fix the error or may introduce new errors.

12.4   Code Reviews

We are all human beings. You may make some mistakes irrespective of your experience in a technology or module. If you just review your code with a second eye, those mistakes might have caught at that time only. This way you can reduce the no. of bugs reported by the testers or end users (Figure 12-2).

If you are working in a geographically distributed team, your coding conventions may differ and if you have some strict coding guidelines, this code review process will make it possible to recheck the standards in the code that you have written.

1. There are some possibilities of repetitive code block which can be caught during a code review process. Refactoring can be done based on that.

2. Unused code blocks, performance metrics etc. are some additional check points of doing a review.

3. If you are new to development, this code review process will help you to find out your mistakes and help you to improve them. This is a perfect knowledge sharing mechanism.

4. Find out the defects and correct them at the beginning before it is committed to the source control system.
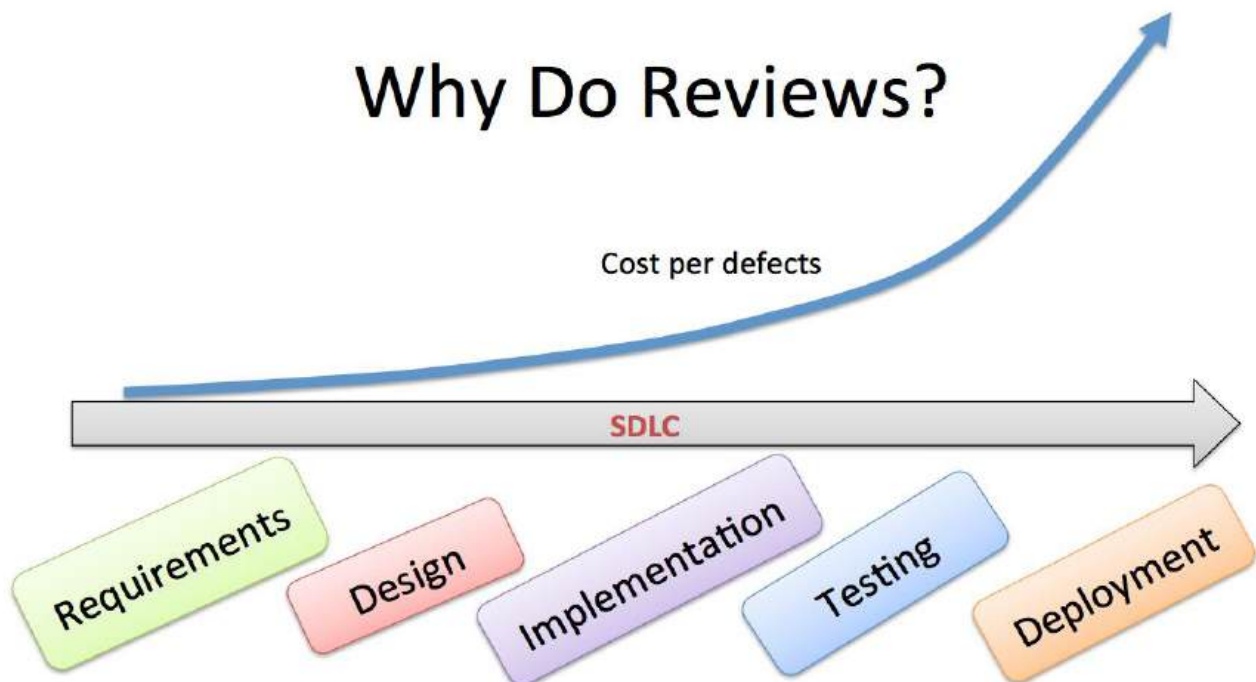
Figure 12-2: Why you should do reviews

[http://geek-and-poke.com]
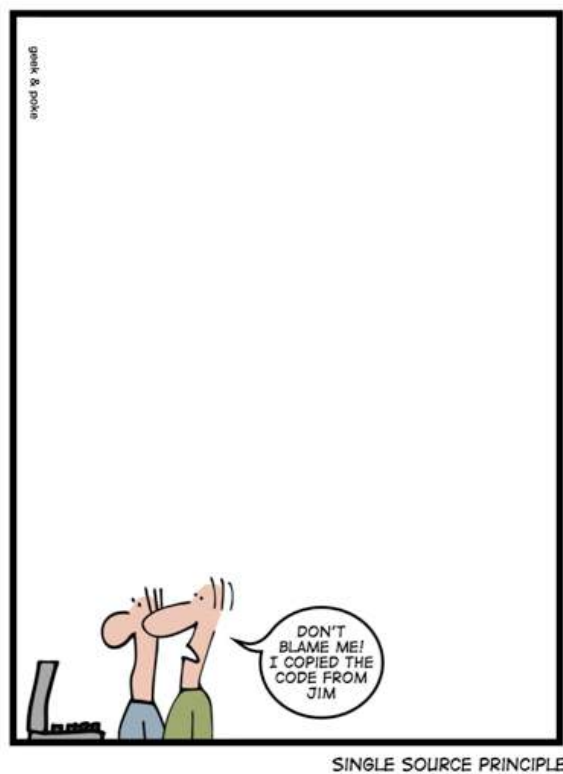
Better code always starts with review process!

Here are some topics that should be checked during the Code Review process [12]:

- **Readability**: The code can be easily read and understood by other programmers.

- **Maintainability**: The code can be easily modified and maintained. Note that this is related to readability, but it is not the same; for example, this involves the use of e.g., Hungarian notation, in which variable names include abbreviations for the type of variable.

- **Performance**: All other things being equal, the implementation should produce code that performs as fast as possible.

- **Traceability**: All code elements should correspond to a design element. Code can be traced back to design (and design to requirements).

- **Correctness**: The implementation should do what it is intended to do (as defined in the requirements and detailed design).

- **Completenes**s: All the system requirements are met.

12.5   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 defined as

- 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

Do you need to refactor your code? – here are some symptoms:

- Coding Style and Name Conventions not followed

- Proper Commenting not followed

- Duplicated code (clearly a waste)

- Long methods (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.

You should Refactoring your continuously and especially after Code Reviews.

13 Testing

13.1 Introduction

Different people have come up with various definitions for Software Testing, but generally, the goal of testing is:

- To ensure that the software meets the agreed requirements and design
- The application works as expected
- The application doesn't contain serious bugs
- Meets its intended use as per user expectations

Testing can be performed on different levels and by different people. Testing is a very important part of software development. About 50% of the software development is about testing your software.

🎬 Software Testing: https://youtu.be/MVdb1vqPH1U

Since modern software has become very complex, testing has become a very important part of software development (see Figure 13-1).
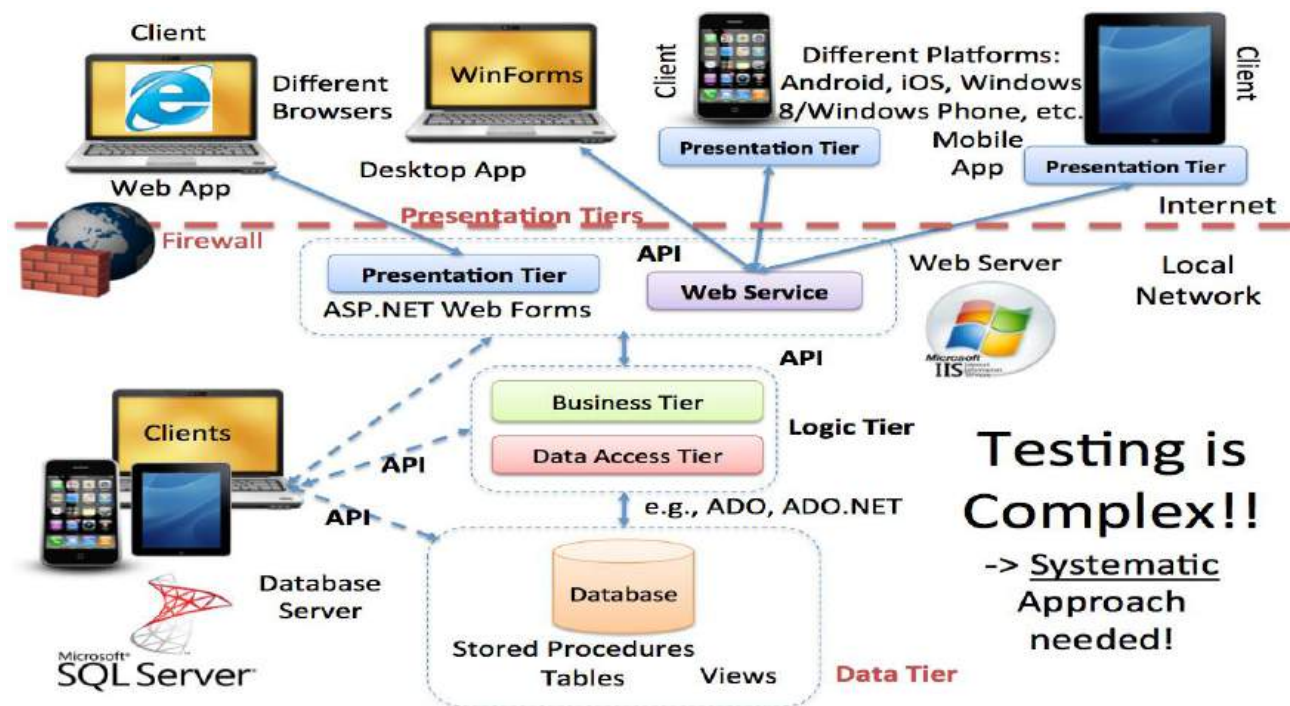
Figure 13-1: Modern Software Testing has become very complex

117

Since testing of advanced software systems is quite complex, we need a systematic approach to testing that involves different levels of testing (see Figure 13-2).
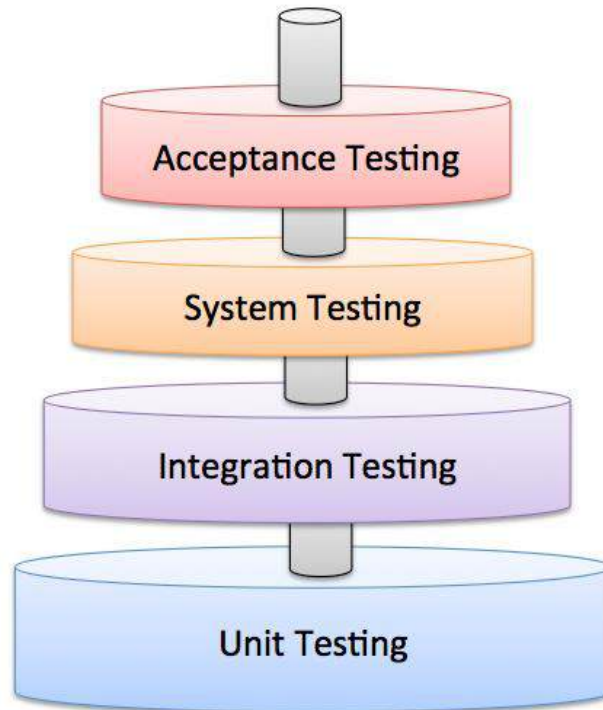
Figure 13-2: Systematic Testing

Since Software Development today involves different platforms, different devices, networks, servers, and clients, etc., it has become very complex to test it. Today we have not only ordinary Desktop Apps, but we also have Web Apps, Mobile Apps, Apps for TVs, etc.

The software we create is a layer between the user of the software and the hardware and the operating system (Figure 13-3).
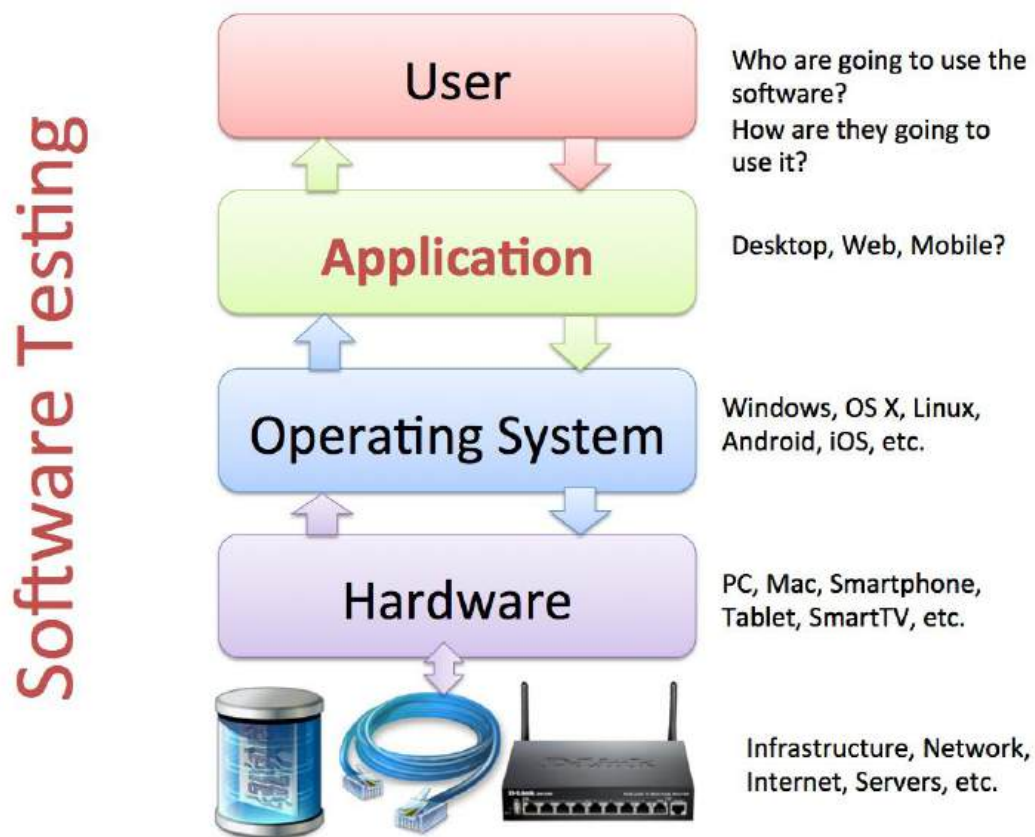
Figure 13-3: Components involved in Software Development & Testing

If we find bugs at the earlier stage, the cost to fix this will be less and thus it will reduce the overall cost of the application (Figure 13-4).
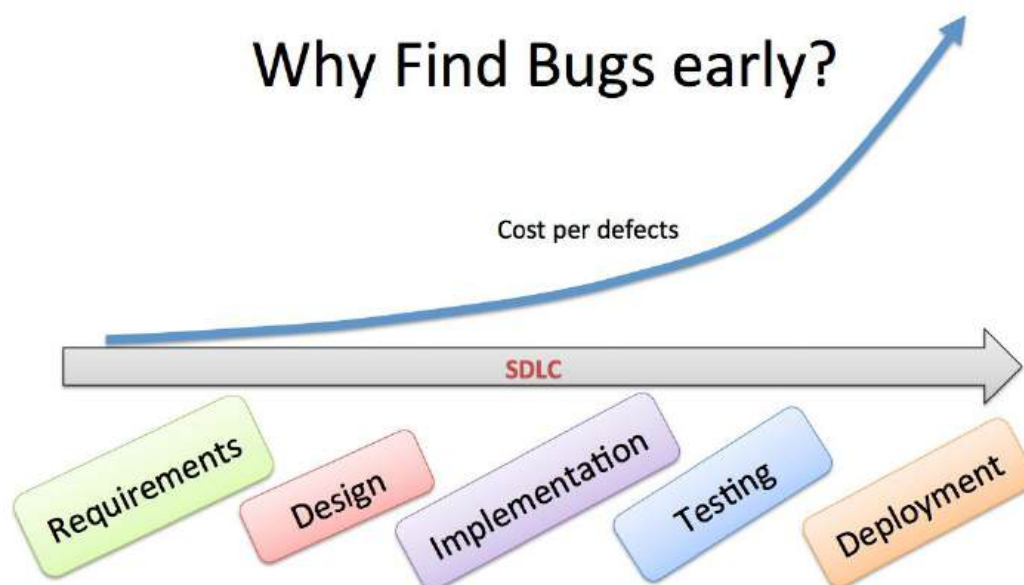
Figure 13-4: Find Bugs at an early stage

Figure 13-5 illustrates the necessary steps involved in testing.
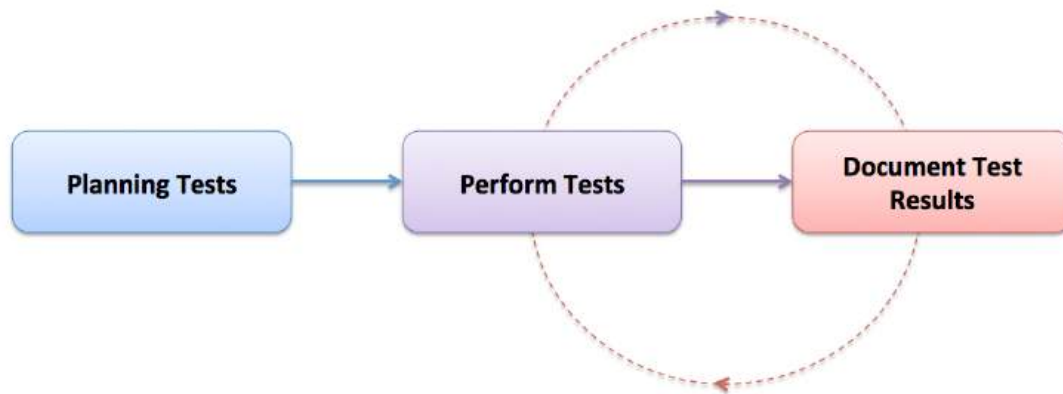
Figure 13-5: Software Testing

Testing is intended to show that a program does what it is intended to do and to discover program defects before it is put into use. When you test software, you execute a program using artificial data. You check for the presence of errors NOT their absence.

Testing is part of a more general verification and validation process, which also includes static validation techniques.

What is the purpose of Testing?

The main purpose of testing is as follows:

- To demonstrate to the developer and the customer that the software meets its requirements.
    - For custom software, this means that there should be at least one test for every requirement in the requirements document.
    - For generic software products, it means that there should be tests for all the system features, plus combinations of these features, that will be incorporated in the product release.
- To discover situations in which the behavior of the software is incorrect, undesirable or does not conform to its specification.
    - This means undesirable system behavior such as system crashes, unwanted interactions with other systems, incorrect computations, and data corruption.

The primary purpose of testing is to detect software failures so that defects may be discovered and corrected.

If we summarize why we do Testing:

- Finding Bugs in the Software before it is released to the Customer
- Finding unwanted system behaviors
- Verify/Validate that the Software works as expected (according to the Specifications)
- Find bugs as soon as possible!

It is commonly believed that the earlier a defect is found the cheaper it is to fix it.

There are different steps involved in the software testing process.

The steps are as follows:

- Design Test Cases
- Prepare Test Data
- Run the Software with the necessary Test Data
- Compare the results with the Test Cases

The final output of this process is a Test Report.

Basically, we do the following: Planning the Test, then we execute the Tests, finally we document the Test results.

Documents used in testing and created in the test process:

- **SRS** – Software Requirements Specifications: A document stating what at application must accomplish. The documents are the basis for the test plan, etc.
- **SDD** – Software Design Document: A document describing the design of a software application. The documents are the basis for the test plan, etc.
- **STP** - Software Test Plan: Documentation stating what parts of an application will be tested, and the schedule of when the testing is to be performed
- **STD** - Software Test Documentation: Introduction, Test Plan, Test Design, Test Cases, Test procedures, Test Log, …, Summary

We have the following stages in testing:

1. **Development testing**, where the system is tested during development to discover bugs and defects. Development testing includes all testing activities that are carried out by the team developing the system.
2. **Release testing**, where a separate testing team test a complete version of the system before it is released to users.
3. **User testing**, where users or potential users of a system test the system in their own environment.

Development testing: Development testing is the responsibility of the software development team. A separate team should be responsible for testing a system before it is released to customers.

Release testing: Release testing is the process of testing a release of a system that is intended for use outside of the development team.

The primary goal of the release testing process is to convince the supplier of the system that it is good enough for use. Release testing is usually a black box testing process where tests are only derived from the system specification.

User testing: We have different types of user testing:

- **Alpha** testing
    - Users of the software work with the development team to test the software at the developer's site.
- **Beta** testing
    - A release of the software is made available to users to allow them to experiment and to raise problems that they discover with the system developers.
- **Acceptance** testing
    - Customers test a system to decide whether it is ready to be accepted by the system developers and deployed in the customer environment. Primarily for custom systems.

13.1.1   Test Levels

In Figure 13-6 we see different test levels.
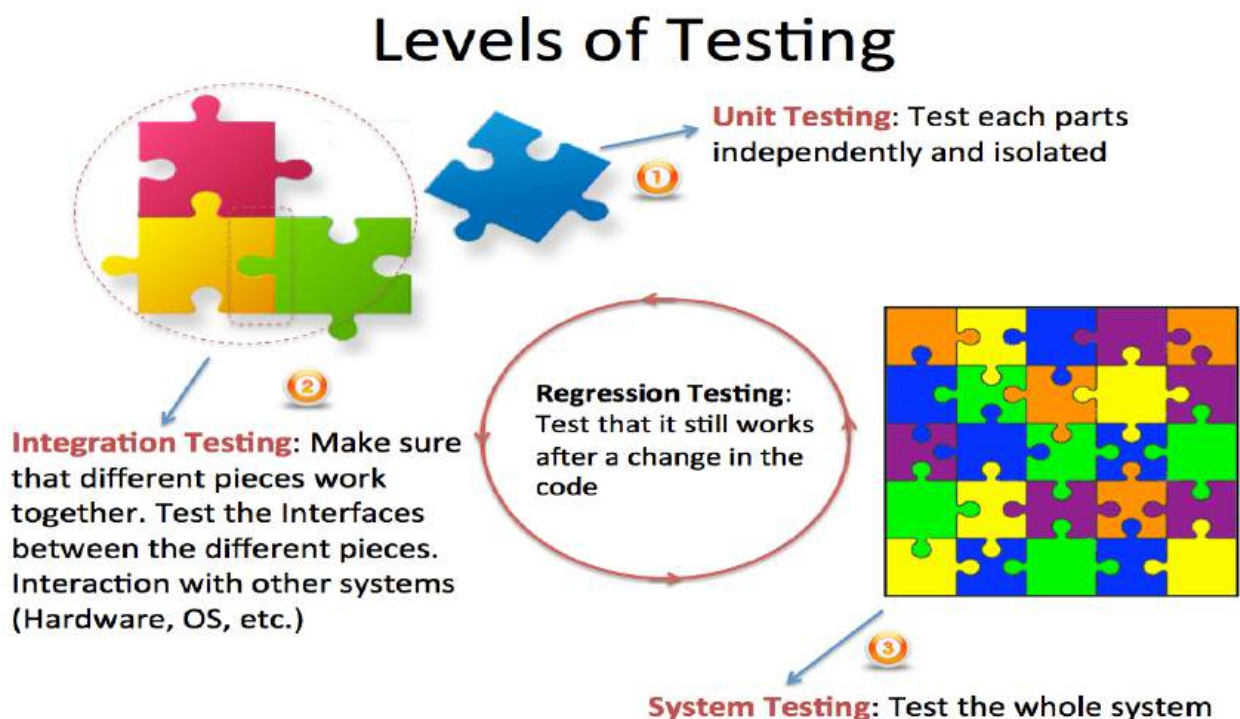
Figure 13-6: Levels of Testing

Short overview of the different Test levels in Figure 13-6 (more details later):

- 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 or validation testing is Black-box Tests that validate the entire system against its requirements, i.e., checking that a software system meets the specifications
- Acceptance Testing: The Customer needs to test and approve the software before he can take it into use. We have 2 types: FAT (Factory Acceptance Testing) and SAT (Site Acceptance Testing).

13.1.2   Bug Tracking

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 (a moth) inside a computer in 1947 that made the program not behave as expected. This was the "first" real bug.

13.1.3   Software versioning

Software versioning is used to separate different versions of the same software, both before it has been released and for subsequent releases. See example in Figure 13-7.

Before the software is released:

- Alpha Release(s)
- Beta Release(s)
- RC - Release Candidate(s)
- RTM – Release To Manufacturing

Maintenance (after the software is released):

- Patches (small fixes)
- SP - Service Packs
  (lots of small fixes and patches bundled together)
- …
- Start Planning next release
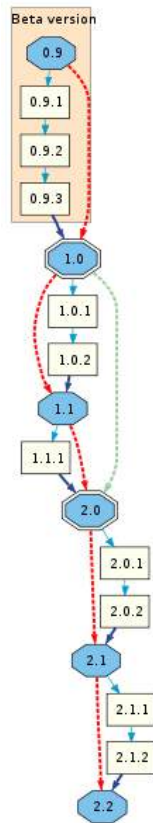
Figure 13-7: Software versioning

Software testing should be performed during the whole Software Development Life Cycle (SDLC) as shown in Figure 13-8.

Testing
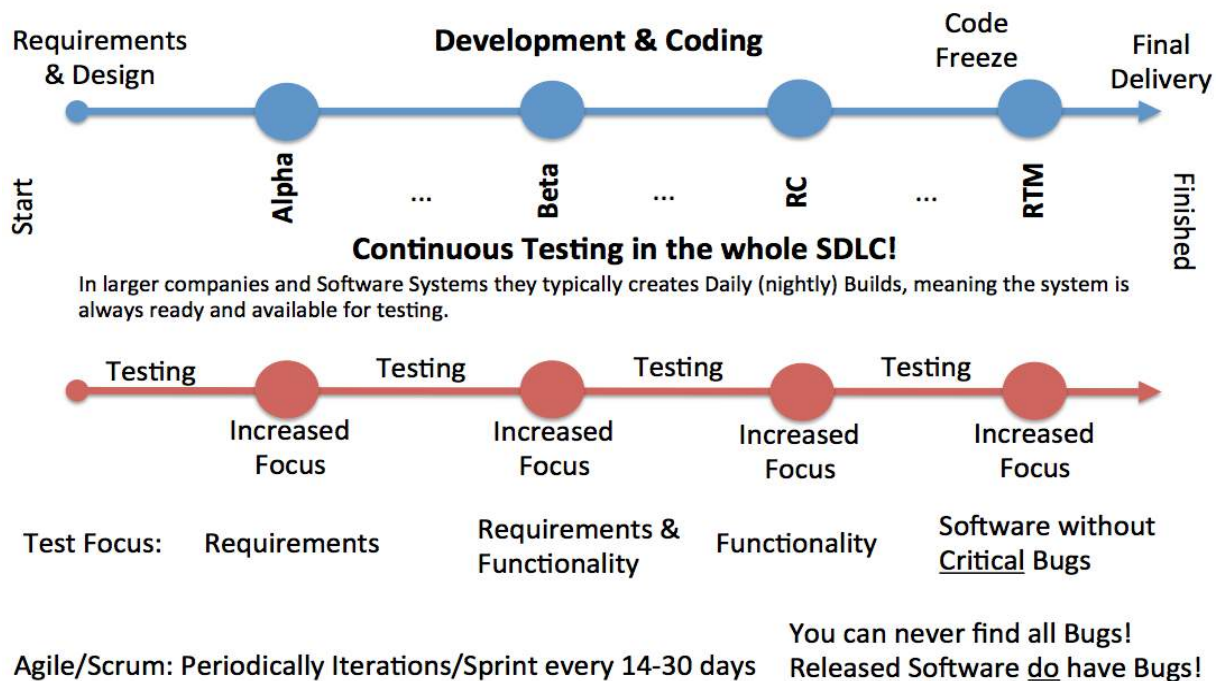
Figure 13-8: Testing during the Software Development Life Cycle (SDLC)

Sooner or later, you have to say enough is enough and release version 1.0 (see Figure 13-9).

The last 10% takes a lot of time!!!

Sooner or later you have to say enough is enough and release version 1.0.

One must define within the development company, development team or in dialogue with the customer what is defined as "good enough". Software will never be 100% complete or error-free!
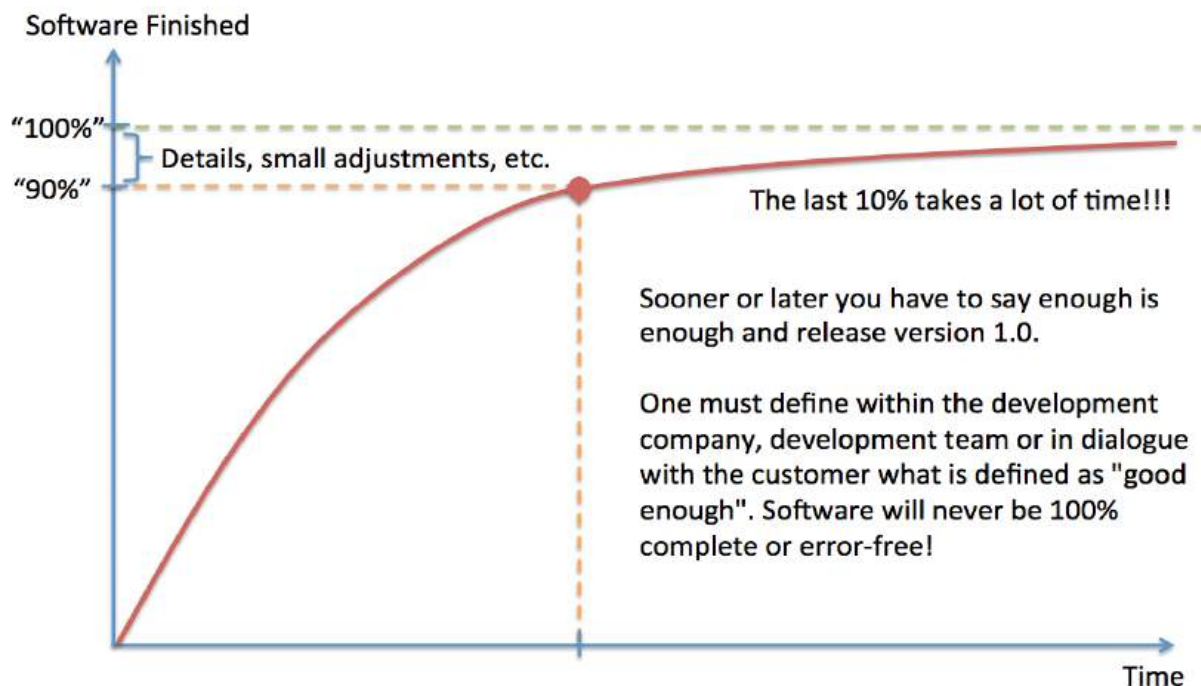
Figure 13-9: When is the Software Finished?

One must define within the development company, development team or in dialogue with the customer what is defined as "good enough".

Software will never be 100% complete or error-free (see Figure 13-10)!
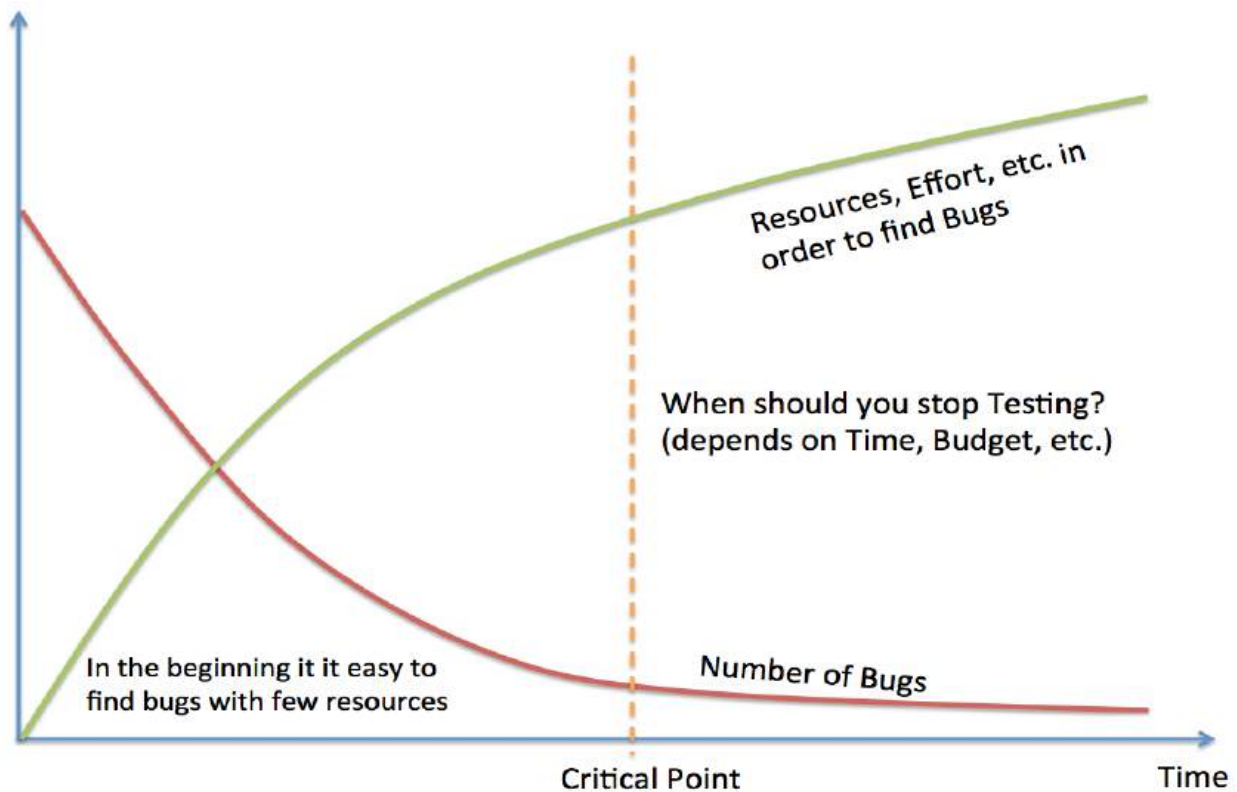
Figure 13-10: When are you Finished with Testing?

13.2   Test Categories

We can divide testing into 2 different categories, which is:

- Black-box Testing
- White-box Testing

13.2.1   Black-box Testing

Black-box testing is a method of software testing that examines the functionality of an application (what the software does) without going inside the internal structure (White-box Testing).

You need no knowledge of how the system is created. Black-box testing can be done by a person who only knows what the software is supposed to do. You may compare to driving a car – you don't need to know how it is built to test it.

13.2.2   White-box Testing

In White-box Testing you need to have knowledge of how (Design and Implementation) the system is built. White-box Testing is also called "Glass-box testing".

In Figure 13-11 we see how White-box testing works.
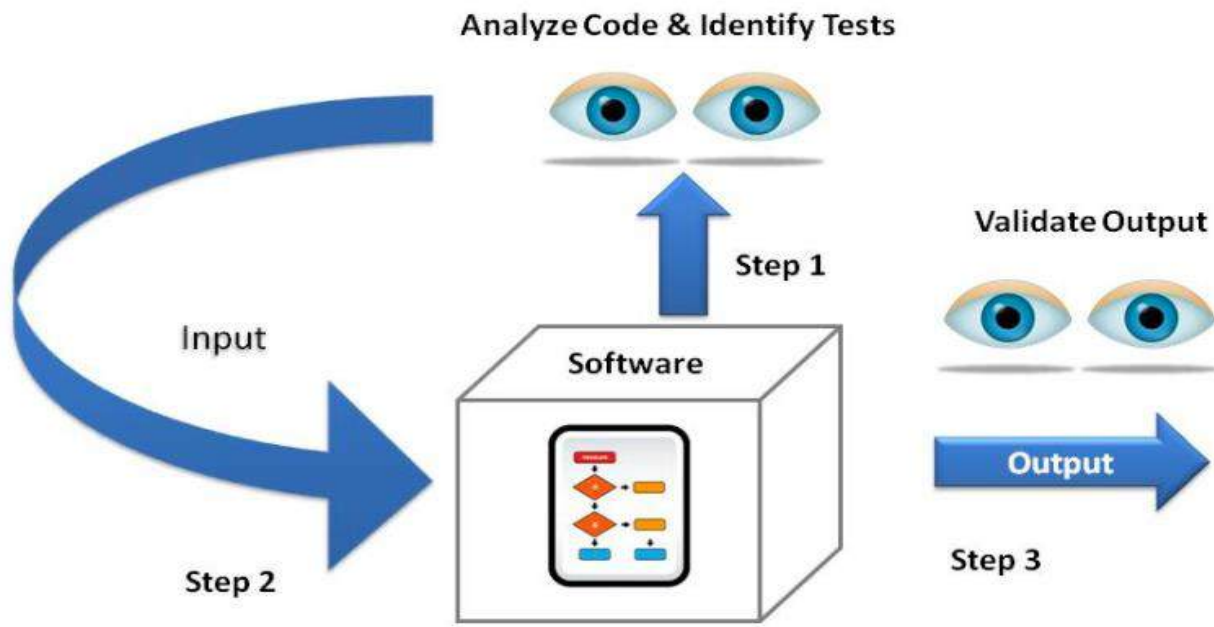
Figure 13-11: White-box Testing

13.3   Test Levels

As mentioned earlier, we have different Levels of Testing (see Figure 13-12).

- Unit Testing
- Regression Testing
- Integration Testing
- System Testing
- Acceptance Testing

These are explained in more detail below.
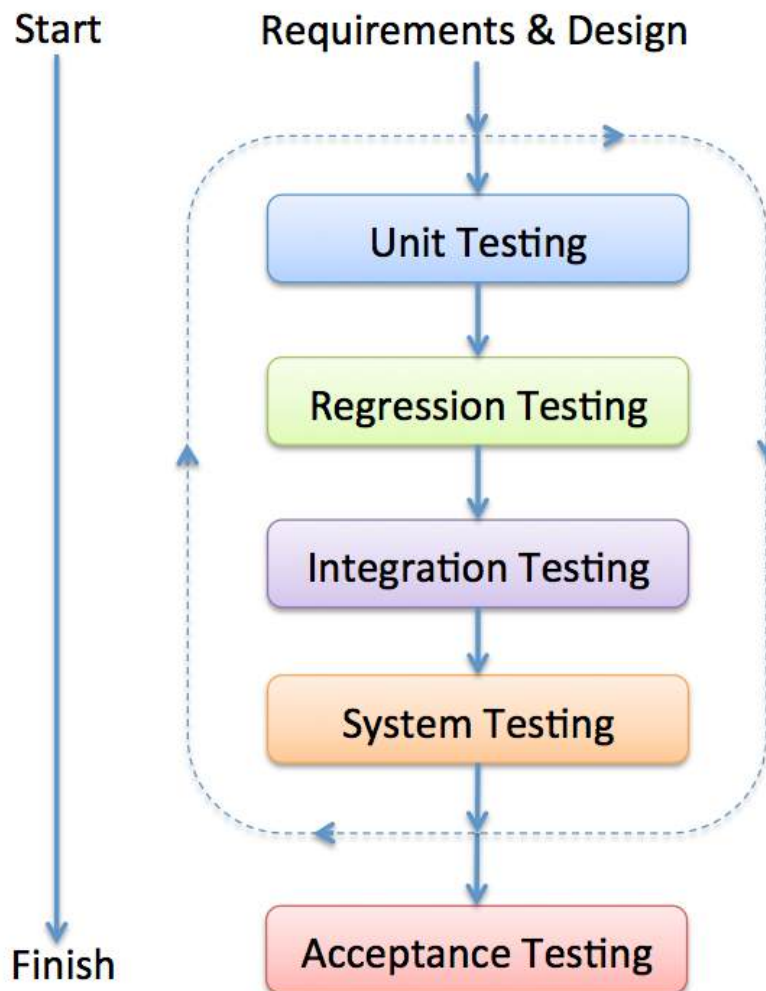
Figure 13-12: Software Test Levels

Short explanations of these Test Levels:

- 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

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

13.3.1   Unit Testing

Unit Testing (or *component testing*) refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class and methods level.

Unit Testing in Visual Studio: https://youtu.be/QIfNViZkqEc

ASP.NET Core - Unit Testing: https://youtu.be/EzeDCEQ2qMs

Unit Tests are written by the developers as part of the programming. They are automatically executed by the system, e.g., Visual Studio and Azure DevOps have built-in functionality for Unit Testing.

Sometimes the Unit Tests are written before you start programming, so-called Test-Driven Development (TDD).

[http://geek-and-poke.com]
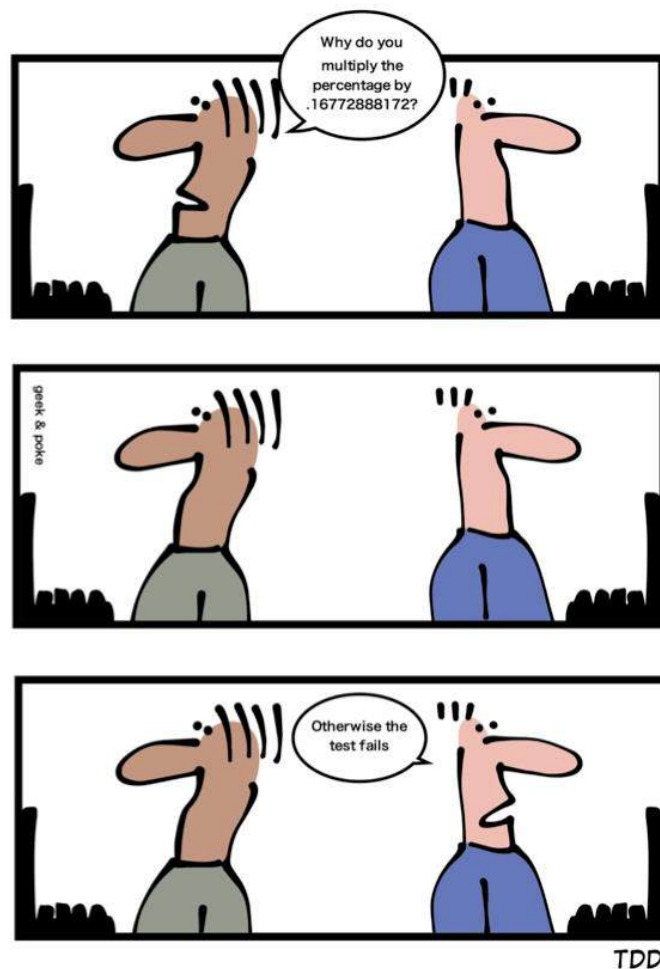
Since Unit testing are part of the development process, so-called Unit Tests Framework are usually integrated with the IDE.

Unit Tests Frameworks:

- **Visual Studio Unit Test Framework**. Unit Tests are built into Visual Studio (no additional installation needed)

- **JUnit** (Java)

  - JUnit is a unit testing framework for the Java programming language.

- **NUnit** (.NET)

  - NUnit is an open-source unit testing framework for Microsoft .NET. It serves the same purpose as JUnit does in the Java world

- **PHPUnit** (PHP)

- LabVIEW Unit Test Framework Toolkit

- etc.

Unit Testing in Visual Studio:

Visual Studio have built-in features for Unit Testing. In the Solution Explorer you just add a "Test Project" as part of your code (see Figure 13-13).
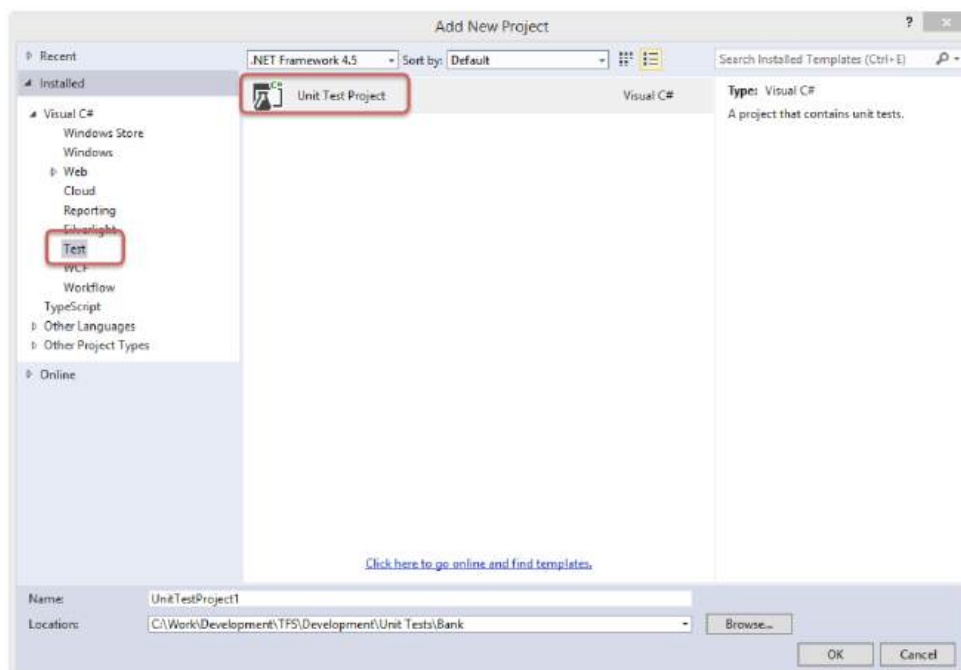
Figure 13-13: Unit Test Project in Visual Studio

In Figure 13-14 we see an example of how you create Unit Tests in Visual Studio and C#.

For Test classes, you need to use [TestClass] and for Test Methods you need to use [TestMethod]. You also need to add a reference to the code under test (select "Add Reference" in the Solution Explorer and include "using <namespace>") in your code.
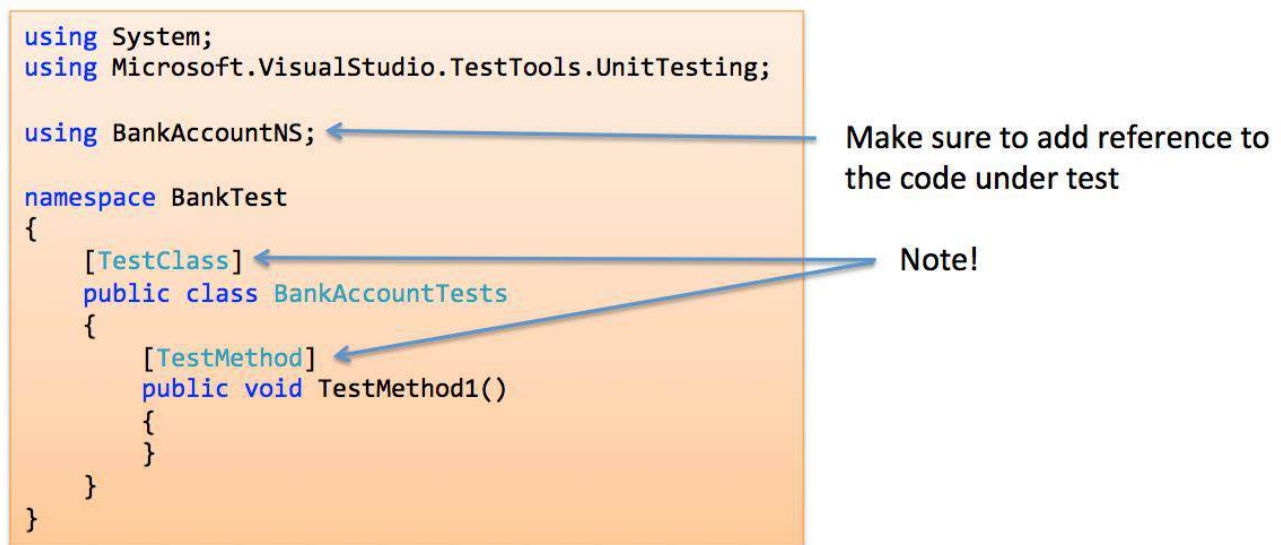
Figure 13-14: Unit Test Principle in Visual Studio and C#

The basic concept in Unit Testing is to compare the results when running the Methods with some Input Data ("Actual") with some Known Results ("Expected").

Example:

```
Assert.AreEqual(expected, actual, 0.001,"Test failed because...");
```

Unit Tests – Best Practice:

Here you see some basic best practice rules regarding unit tests:

- A Unit Test must only do one thing

- Unit Test must run independently

- Unit Tests must not be depending on the environment

- Test Functionality rather than implementation

- Test public behavior: private behavior relates to implementation details

- Avoid testing UI components

- Unit Tests must be easy to read and understand

- Create rules that make sure you need to run Unit Tests (and they need to pass) before you can Check-in your code in the Source Code Control System

13.3.2   Regression Testing

Regression testing focuses on finding defects after a major code change has occurred. Specifically, it seeks to uncover software regressions, or old bugs that have come back.

1. Regression testing is testing the system to check that changes have not "broken" previously working code.
2. In a manual testing process, regression testing is expensive but, with automated testing, it is simple and straightforward. All tests are rerun every time a change is made to the program.
3. Tests must run "successfully" before the change is committed.

13.3.3   Integration Testing

Integration testing verifies the interfaces between components against a software design.

13.3.4   System Testing/Validation Testing

System Testing follows Integration Testing.   It consists of Black-box Tests that validate the entire system against its requirements.   System Testing is about checking that a software system meets specifications and that it fulfills its intended purpose. System Testing is often executed by an independent group (QA group). QA – Quality Assurance.

Since system tests make sure the requirements are fulfilled, they must systematically validate each requirement in the SRS (Software Requirements Specification).

13.3.5   Acceptance Testing

Customers test a system to decide whether it is ready to be accepted from the system developers and deployed in the customer environment. It is primarily for custom systems.

In Figure 13-15 we see a typical acceptance test process.
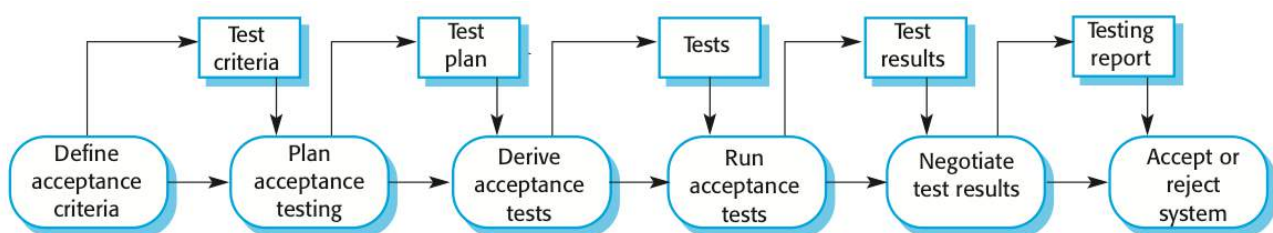
Figure 13-15: Acceptance Testing [1]

The steps are:

- Define acceptance criteria
- Plan acceptance testing
- Derive acceptance tests
- Run acceptance tests
- Negotiate test results
- Reject/accept system

We have 2 main types of Acceptance Testing:

- FAT – Factory Acceptance Testing
- SAT – Site Acceptance Testing

FAT – Factory Acceptance Testing is usually performed in the Test Environment at the software company.

SAT – Site Acceptance Testing is performed at the Customer in the actual Production Environment. This is the final step to determine if the requirements of a specification or contract are met.

If the test is accepted, the software is officially handed over to the customer.

Note! Other terms and definitions are used as well in different literature.

13.4   Test Documentation

In Figure 13-16 we see the steps involved in the software testing process.
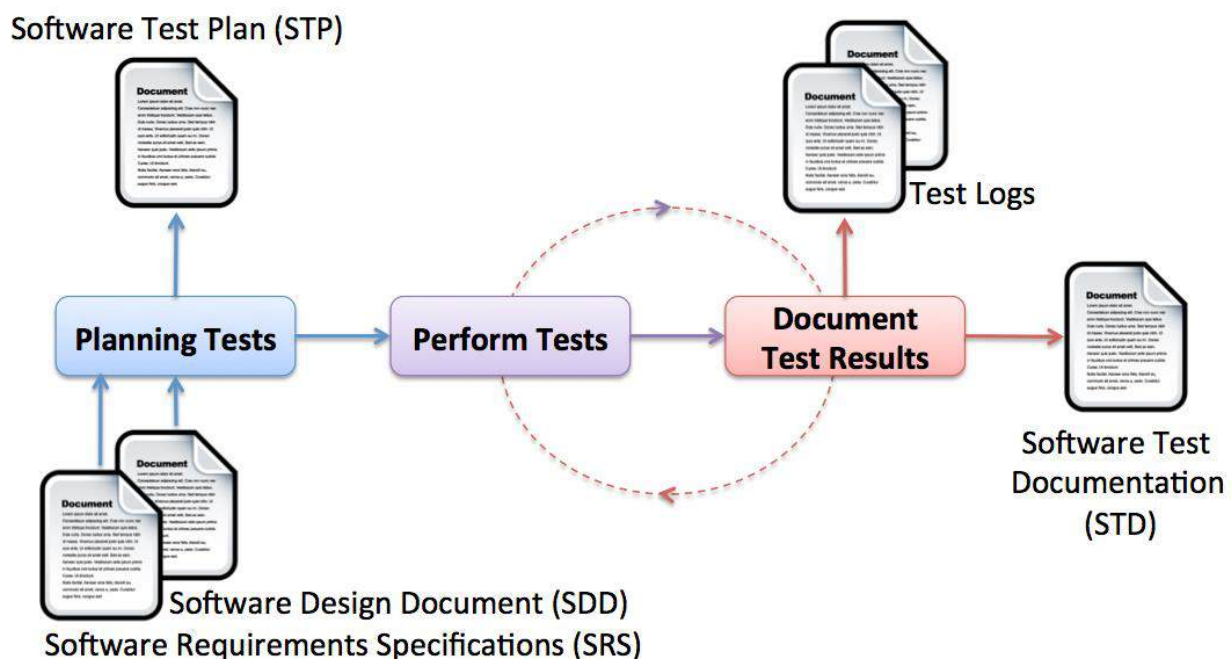
Figure 13-16: The Software Testing Process

Documents involved:

- **SRS** – Software Requirements Specifications: A document stating what at application must accomplish
- **SDD** – Software Design Document: A document describing the design of a software application
- **STP** - Software Test Plan: Documentation stating what parts of an application will be tested, and the schedule of when the testing is to be performed
- **STD** - Software Test Documentation: Introduction, Test Plan, Test Design, Test Cases, Test procedures, Test Log, …, Summary

In addition to writing different documents in your test phase, you should have a Bug Tracking System. With a Bug Tracking System, you can easily store all your bugs in a database system, set priorities, use search to find bugs, use different statistics, etc. More about Bug Tracking Systems below.

13.4.1   Test Planning

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 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.

A **Software Test Plan (STP)** document typically 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 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.

13.5   Bug Tracking Systems

All the results from the testing need to be documented, stored, and tracked.

For this purpose, we use a so-called Bug Tracking System.

Bug Reporting and Tracking with Azure DevOps: https://youtu.be/0tlWdqWdFeQ

Here are some popular Bug Tracking Systems in use today:

- Azure DevOps
- Jira
- Bugzilla
- ClearQuest

More about Bug Tracking Systems in Chapter 23 - Bug Tracking Systems.

We will focus on Azure DevOps in this document. The bug tracking features in Azure DevOps will be discussed in another chapter.

In Azure DevOps we can add requirements, user stories, tasks, new features, bugs, etc. as so-called "Work Items" (Figure 13-17).

Figure 13-17: Azure DevOps – Work Items

13.6   Test Environment

A testing environment is a setup of software and hardware on which the testing team is going to perform the testing of the newly built software product.

VirtualBox - Installation of Windows 11 in a Virtual Machine:
https://youtu.be/OA9urLN4DWo

Install WinForm Desktop App in Virtual Test Environment using VirtualBox:
https://youtu.be/g7CPEVFT8AA

Install ASP.NET Core Web App in Virtual Test Environment using VirtualBox:
https://youtu.be/7XrRd7voasI

Here are also some other resources using VMware Workstation:

ASP.NET Core - Deploy to Virtual Test Environment: https://youtu.be/WzJIoP4WpmU

ASP.NET Core - Web Server IIS Deployment: https://youtu.be/MoI9SSLV4B4

Why do we need a Test Environment? Here are some reasons:

- "It works on my PC" says the Developer

- We need a Clean Environment when testing

- 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.

This setup consists of a physical setup which includes hardware, and logical setup that includes Server Operating system, client operating system, database server, front end running environment, browser (if web application), IIS (version on server side) or any other software components required to run this software product.

This testing setup is to be built on both the ends – i.e., the server and client.

To set up such environments, virtualization is the answer.

More about virtualization below.

Developers   Developers & Testers   Customers

Development → Testing → Production

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)

Development Environment

Test Environment

Production Environment

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.

Figure 13-18: Development, Test and Production Environment

JUST IN CASE YOU'RE STILL NOT
SURE WHETHER YOU'RE IN A
SOFTWARE PROJECT

WAIT UNTIL YOU HEAR THIS:

geek & poke

ON MY
MACHINE IT
WORKS

[http://geek-and-poke.com]
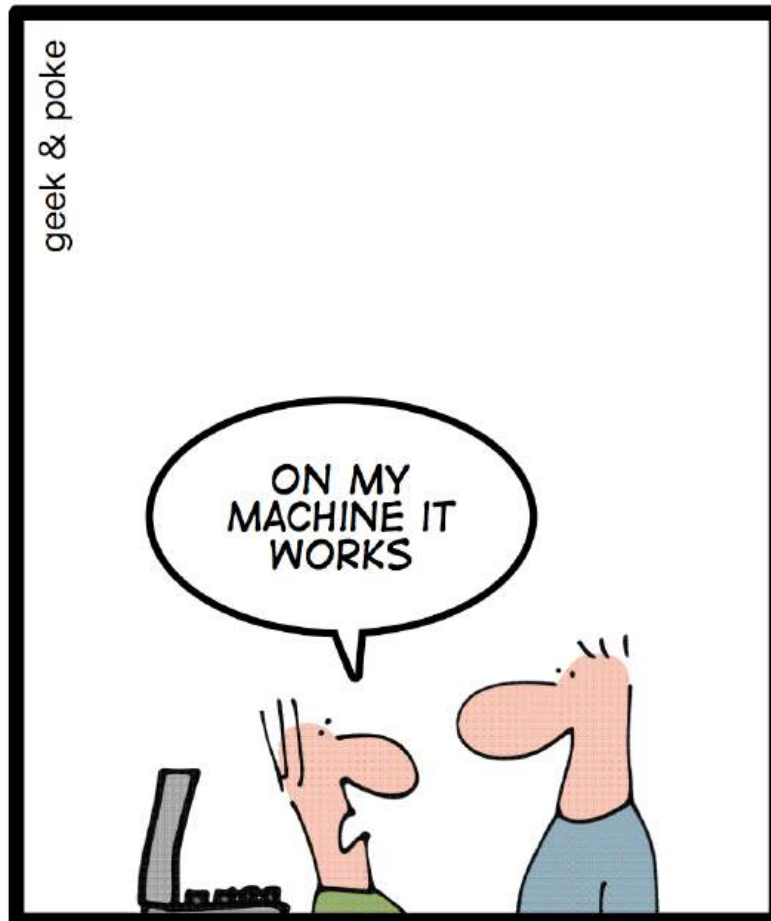
13.6.1   Virtualization

To create test environments easily, virtualization is the answer. There exists lots of different virtualization solutions on the market today.

Here are some examples:

- VirtualBox
- VMware Workstation
- VMware vSphere
- Microsoft Hyper-V
- VMware Fusion (Mac)

- Parallels Desktop (Mac)
- etc.

Introduction to Virtualization: https://youtu.be/gn8pSgShnBQ

VirtualBox - Installation of Windows 11 in a Virtual Machine:
https://youtu.be/OA9urLN4DWo

Install WinForm Desktop App in Virtual Test Environment using VirtualBox:
https://youtu.be/g7CPEVFT8AA

Install ASP.NET Core Web App in Virtual Test Environment using VirtualBox:
https://youtu.be/7XrRd7voasI

VirtualBox is a free virtualization tool. VMware Workstation is another virtualization tool.

With such tools you can create so-called Virtual Machines (VM) where you can install and run all kinds of software.

In this way, you can easily test your software without destroying your own computer and you can easily test it in different operating systems, etc. See Figure 13-19.
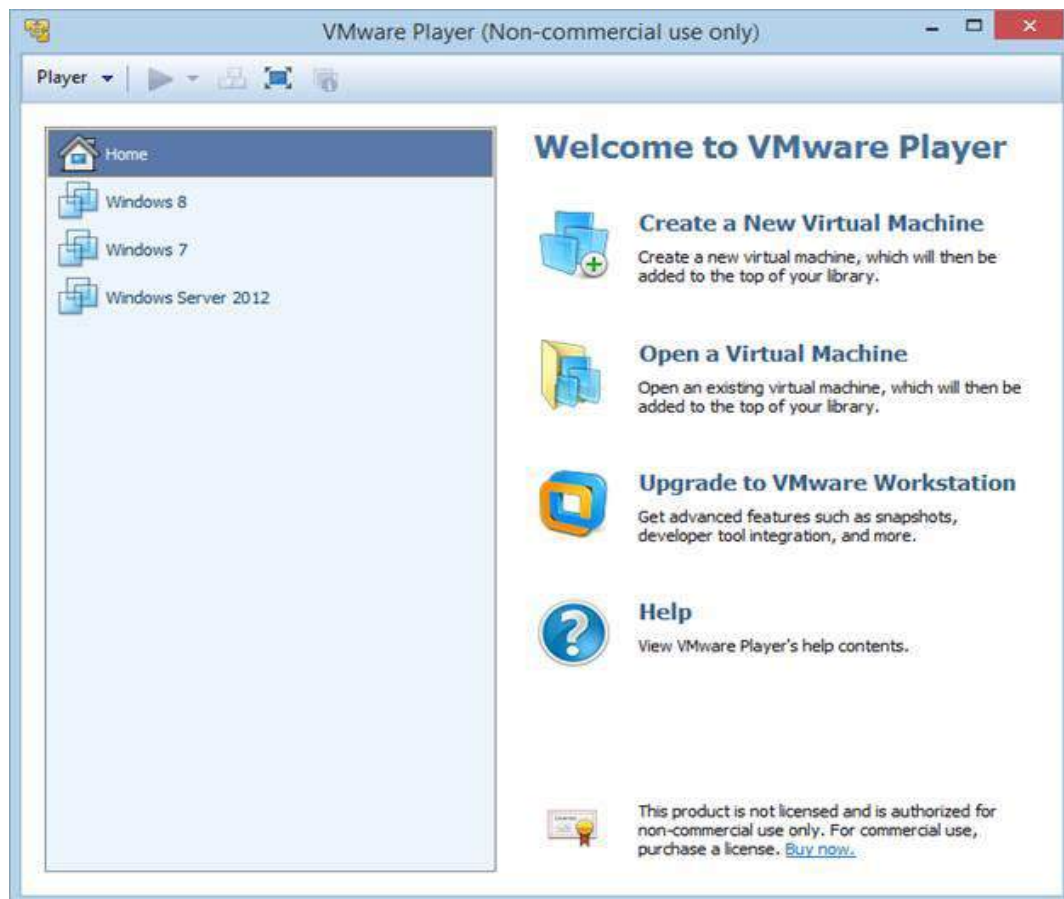
Figure 13-19: VMware Workstation

13.7   Terms used in Testing

Here we will discuss some terms used in software testing not covered earlier.

13.7.1   Bugs

A software bug is an error, flaw, failure, defect, 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 (a moth) inside a computer in 1947 that made the program not behave as expected. This was the "first" real bug.

13.7.2   Debugging

Debugging is about different techniques for finding and fixing bugs (errors that make your code not work as expected) in your code. It is difficult to write code without errors (bugs), but e.g., Visual Studio and other tools have powerful Debugging functionality (breakpoints, etc.). The Compiler will also find syntax errors, etc.

For more "advanced" bugs other methods are required (Unit Testing, Integration Testing, Regression Testing, Acceptance Testing, etc.). The focus here will be on these methods, while Debugging is something you learned in Programming courses.

13.7.3   Code Coverage

Code coverage is a measure used in software testing. It describes the degree to which the source code of a program has been tested.

Example:

```
int foo (int x, int y)
{
    int z = 0;
    if ((x>0) && (y>0))
    {
        z = x;
    }
    return z;
}
```

When we test this function, it depends on the input arguments which parts of the code will be executed. Unit Tests should be written to cover all parts of the code.

13.7.4   Eat your own Dog food

"Eating your own dog food", also called "dogfooding", is a slang term used to reference a scenario in which a company (usually, a computer software company) uses its own product to demonstrate the quality and capabilities of the product.

Example: Microsoft uses Windows PCs and Visual Studio to create their software.

[http://geek-and-poke.com]

13.7.5   Code/Feature Freeze

The Developer cannot add new features to the software, only fix bugs. When it is very close to release, they cannot fix bugs either.

[http://geek-and-poke.com]
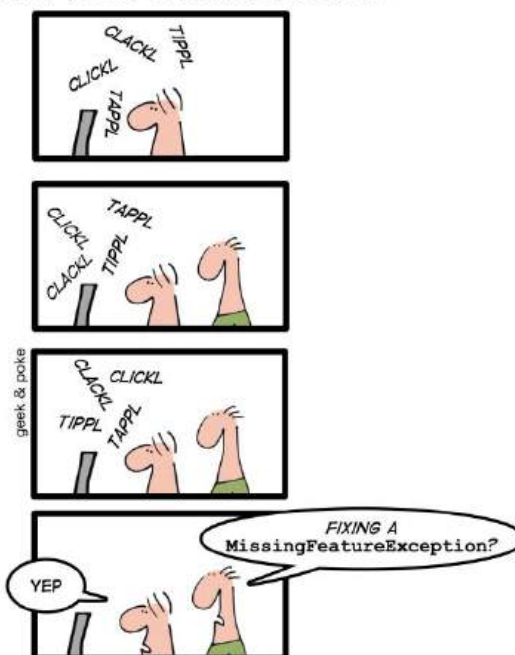
13.7.6   Test-Driven Development (TDD)

In TDD coding and testing are done in parallel. The tests are normally written <u>before</u> the code. TDD was introduced as part of eXtreme Programming (XP).

13.7.7   Development-Driven Testing (DDT)

DDT is all about giving more responsibility to developers specifically, and the development process in general. It works especially well when using test cases as requirements, and having the developers write these test cases. But it's not DDT unless those tests are written near the end of the process, when the code is checked in, and the developers figure they're done.

The advantages of Development-Driven Testing are many. Instead of tests driving the development, it's developers driving the tests, so you get just a few tests, and they almost always all pass. The project team can deliver on time for a change, with zero bugs found in every iteration. This makes management happy, and isn't that really the ultimate barometer of success? Also, velocity is increased dramatically when using this process.

Development-driven testing makes all sense in the world for those who practice Agile.

[http://geek-and-poke.com]
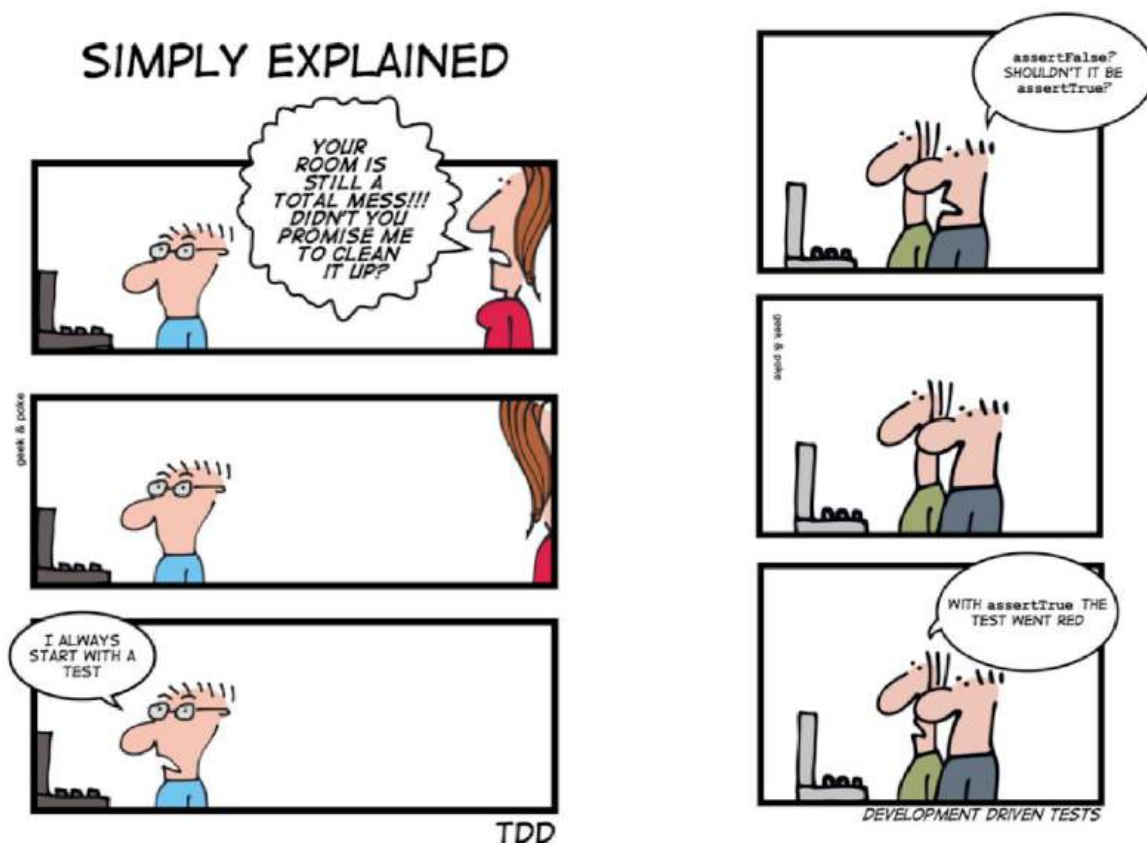
13.8   The 7 Principles of Testing

The 7 Principles of Testing are as follows:

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, 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 and requirements

For more information about these 7 principles of testing, see the following:

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

and

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

13.9   Testing Summary

Figure 13-20 gives an overview of different Test Categories, Test Levels and Test Methods.
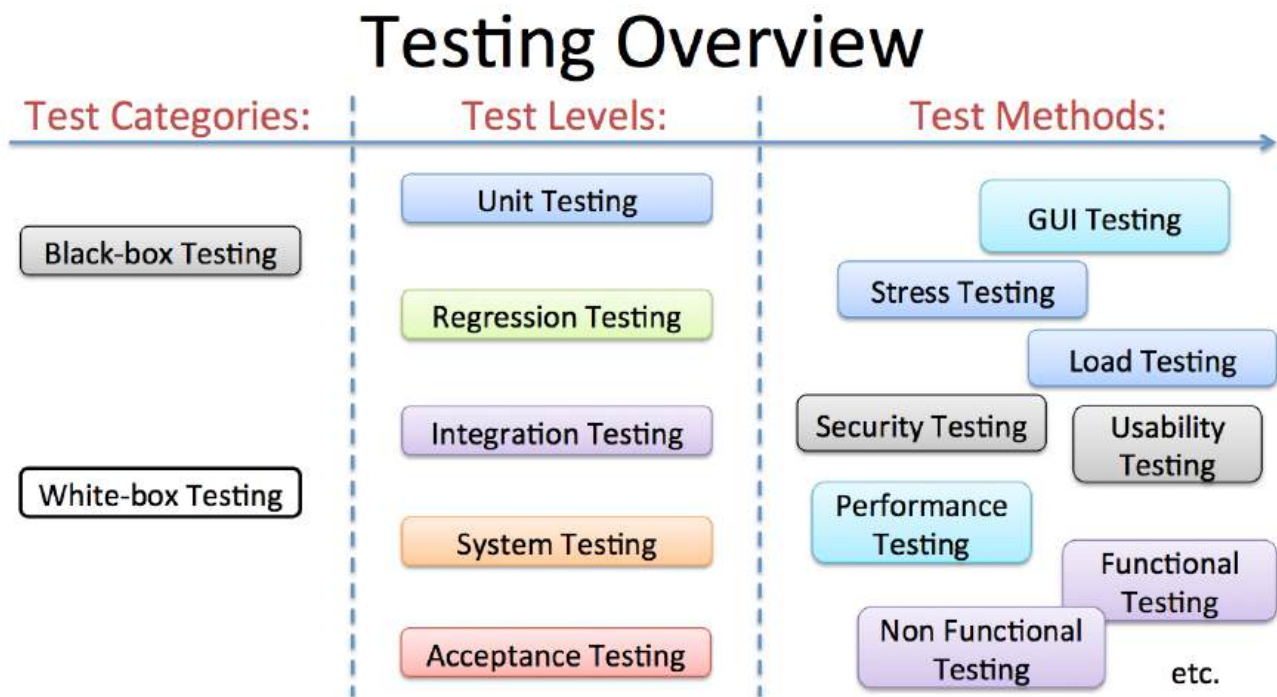
Figure 13-20: Test Categories, Test Levels and Test methods

13.10   Exercises

Make sure to discuss and reflect over the following:

1. Why do we need to test the software?

2. List different Test methods

3. We have 2 main categories of testing. Explain.

4. Explain the difference between a "Bug" and a "Feature"

5. What is Code/Feature Freeze?

6. What is "Dogfooding"?

7. What is a Code Review?

8. What Explain TDD and DDT?

9.   What is Unit Testing?

10. What is the difference between Functional and Non-Functional Testing?

14 Deployment and Installation

14.1 Introduction

Getting software out of the hands of the developers into the hands of the users. More than 50% of commissioned software is not used, mostly because it fails at the deployment stage. 80% of the cost of (commissioned) software comes at and after deployment.

Deploy a Windows Forms App using Visual Studio: https://youtu.be/qXS9ie3KZFE

Virtualization Deployment:

Install WinForm Desktop App in Virtual Test Environment using VirtualBox: https://youtu.be/g7CPEVFT8AA

Install ASP.NET Core Web App in Virtual Test Environment using VirtualBox: https://youtu.be/7XrRd7voasI

Microsoft Azure Deployment:

Microsoft Azure - SQL Databases and App Services: https://youtu.be/ca6Q6LdshIs

Software deployment is all the activities that make a software system available for use.

Examples:

- Get the software out to the customers

- Creating Installation Packages

- Documentation

  o Installation Guide, etc.

- Installation

- etc.

Deployment strategies may vary depending of what kind of software we create, etc.

14.2   Releases

Now we are finished with all the development and testing and are ready to start the deployment process.

Typically, we have the following "Internal" releases (see also Figure 14-1):

- **Alpha** Release(s)

- **Beta** Release(s)

- **RC** - Release Candidate(s)

You are finished:

- **RTM** – Release To Manufacturing

    - Your software is good enough and it is ready for Deployment!
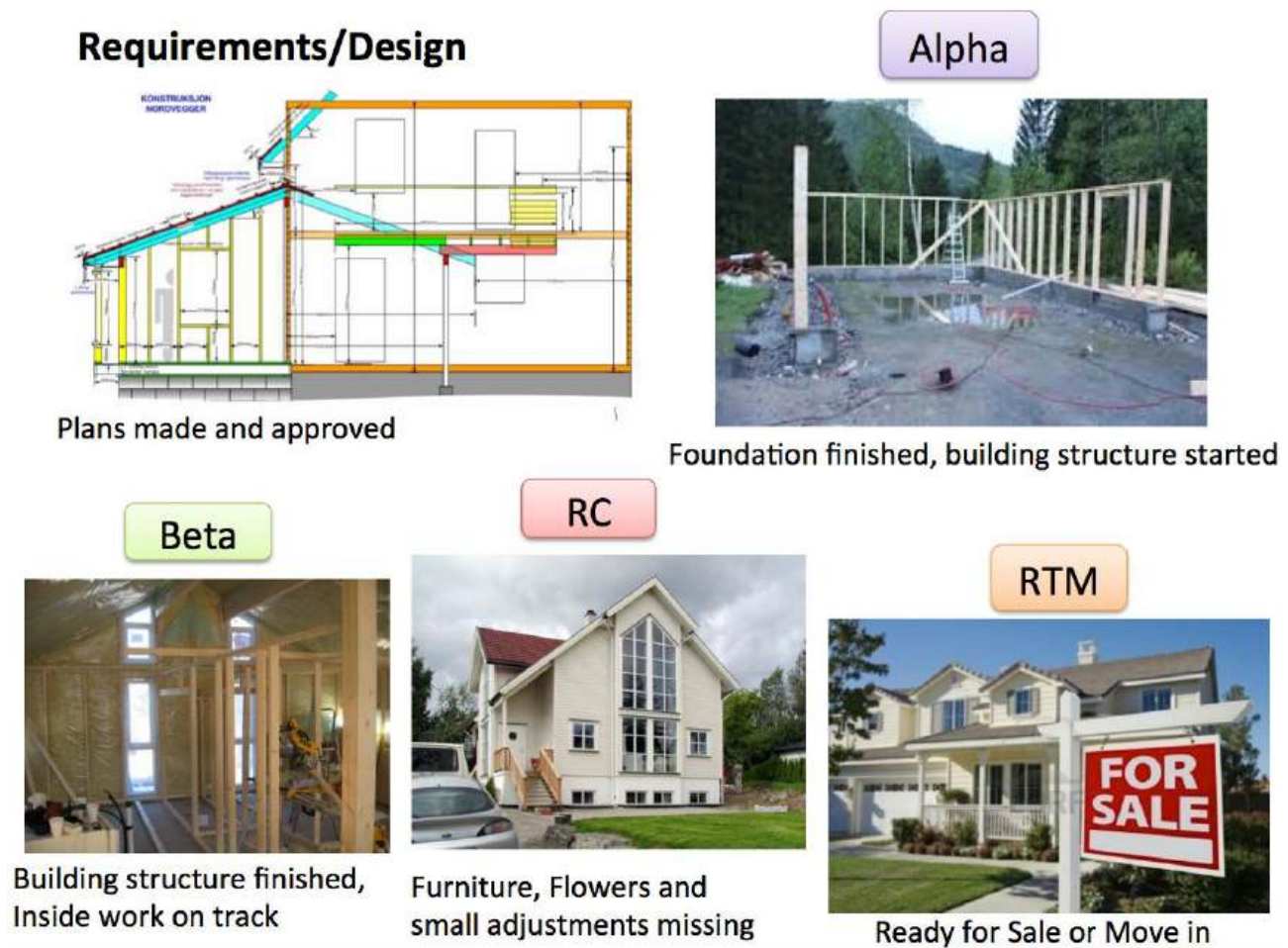
Figure 14-1: Software Releases before Releasing the Software

Below we see an example of the Windows 8 life cycle releases:

- Start planning and development of Windows 8, 2008/2009 (the planning started before Windows 7 was released)

    – Internal Builds xxxx…xxxx

    – Internal Alpha versions, Alpha 1, 2, 3

    – Internal Builds xxxx…xxxx

    – Internal Milestone1 Release (build 7850), 2010.09.22

    – Internal Milestone2 (build 7955), Milestone3 (build 7989)

- **Developer Preview** (build 8102), 2011.09.13

    – Internal Builds xxxx…xxxx

- **Consumer Preview** (build 8250), 2012.02.29

    – Internal Builds xxxx…xxxx

- **Release Preview** (build 8400), 2012.05.28

    – Internal Builds xxxx…xxxx

- **RTM** Release (build 9200), 2012.08.01

14.3   Deployment

What is Deployment?

Software deployment is all the activities that make a software system available for use.

Examples:

- Get the software out to the customers

- Creating Installation Packages

- Documentation

    o Installation Guide, etc.

- Installation

- etc.

Deployment strategies may vary depending of what kind of software we create, etc.

Key Issues around Deployment:

- **Business Processes**: Most large software systems require the customer to change the way they work.

- **Training**: No point in deploying software if the customers can't use it.

- **Support**: The need goes on, and on, and on.

- **Deployment**: How do you physically get the software installed.

- **Equipment**: Is the customer's hardware up to the job?

- **Expertise**: Does the customer have the IT expertise to install the software?

- **Upgrades**: Can't avoid them!

- **Integration**: Shall the software interact/integrate with other systems of the customer.

- **Performance**: The Customer may not have the same hardware as in the Development/Test Environment

14.4   Test and Production Environment

Typically, "everything" works on the computer that the developer of the code is using, but the customer's computer may use another OS, another version of the hardware, another version of a 3. party component or other software that your software relies on, etc. Therefore, it is very important to test the software on other computers and other environments, different versions of hardware, different versions of web browsers, etc.

During the software lifecycle, we have 3 different environments for the software we are creating:

- Development Environment
- Test Environment
- Production Environment

Figure 14-2 gives an overview of these different software environments.
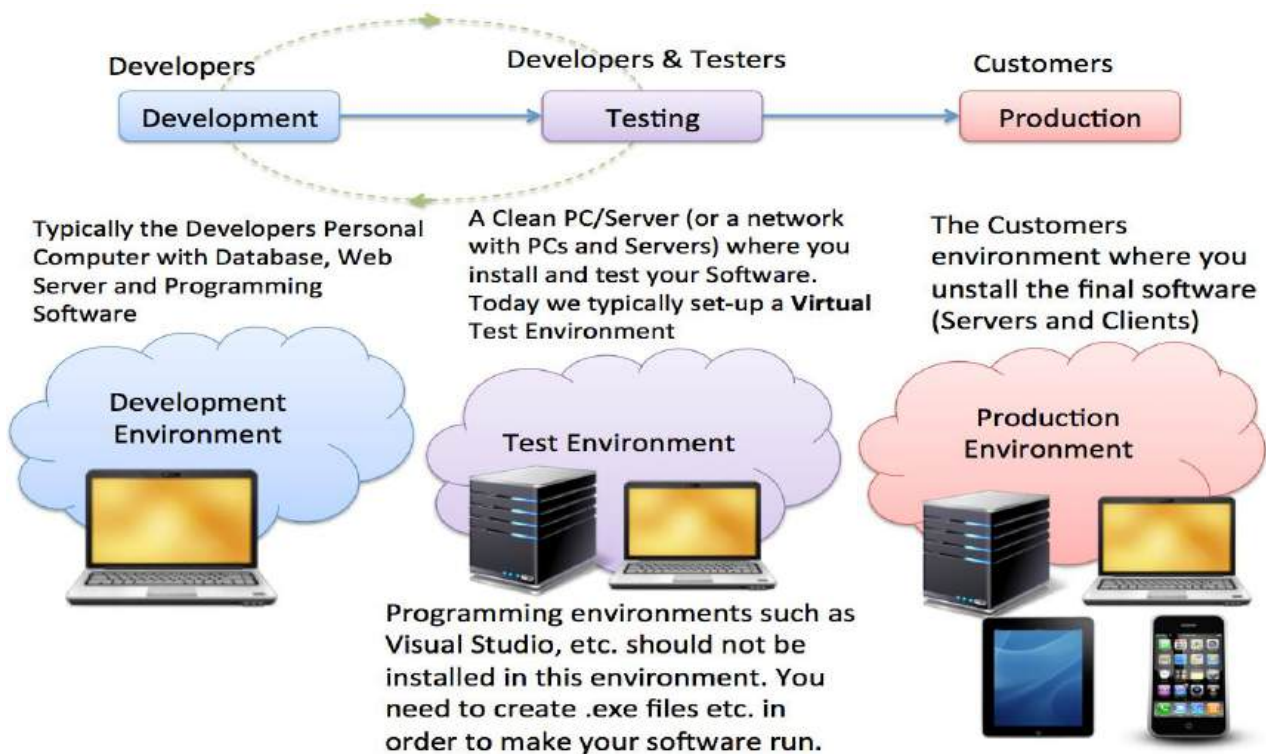
Figure 14-2: Development-, Test- and Production Environment

[http://geek-and-poke.com]
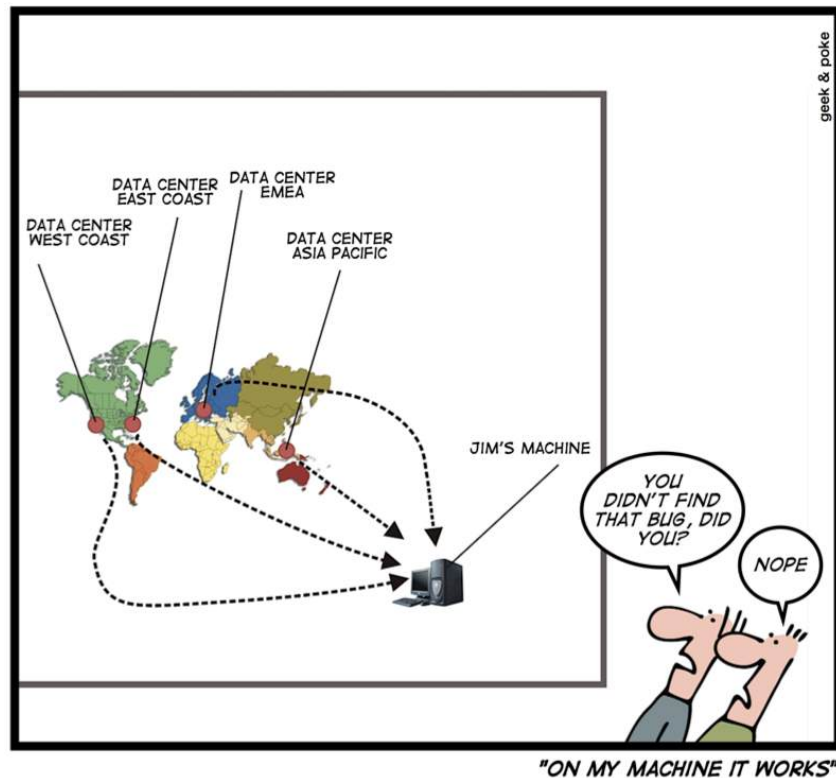
Test/Production Environment is an Infrastructure with Servers, Virtual Servers, Database Servers, Web Servers, etc.

- Local Infrastructure with Servers & Virtualization

- Cloud-based Infrastructure (monthly payment), e.g.:

  - Windows Azure www.windowsazure.com

  - Amazon Web Services (AWS) http://aws.amazon.com

  - Google Cloud Platform https://cloud.google.com

  - etc.

14.4.1   Development Environment

This is where the developers create the code, typically the developer's personal computer.

14.4.2   Production Environment

Production environment is a term used mostly by developers to describe the setting where software and other products are put into operation for their intended uses by end users.

A production environment can be thought of as a real-time setting where programs are run, and hardware setups are installed and relied on for organization or commercial daily operations.

14.4.3   Test Environment

A testing environment is a setup of software and hardware on which the testing team is going to perform the testing of the newly built software product.

This setup consists of a physical setup which includes hardware, and logical setup that includes Server Operating system, client operating system, database server, front end running environment, browser (if web application), IIS (version on server side) or any other software components required to run this software product.

This testing setup is to be built on both the ends – i.e., the server and client.

15   Project Documentation

During the software development, a lot of documentation (Figure 15-1) is created in the different phases of the development.
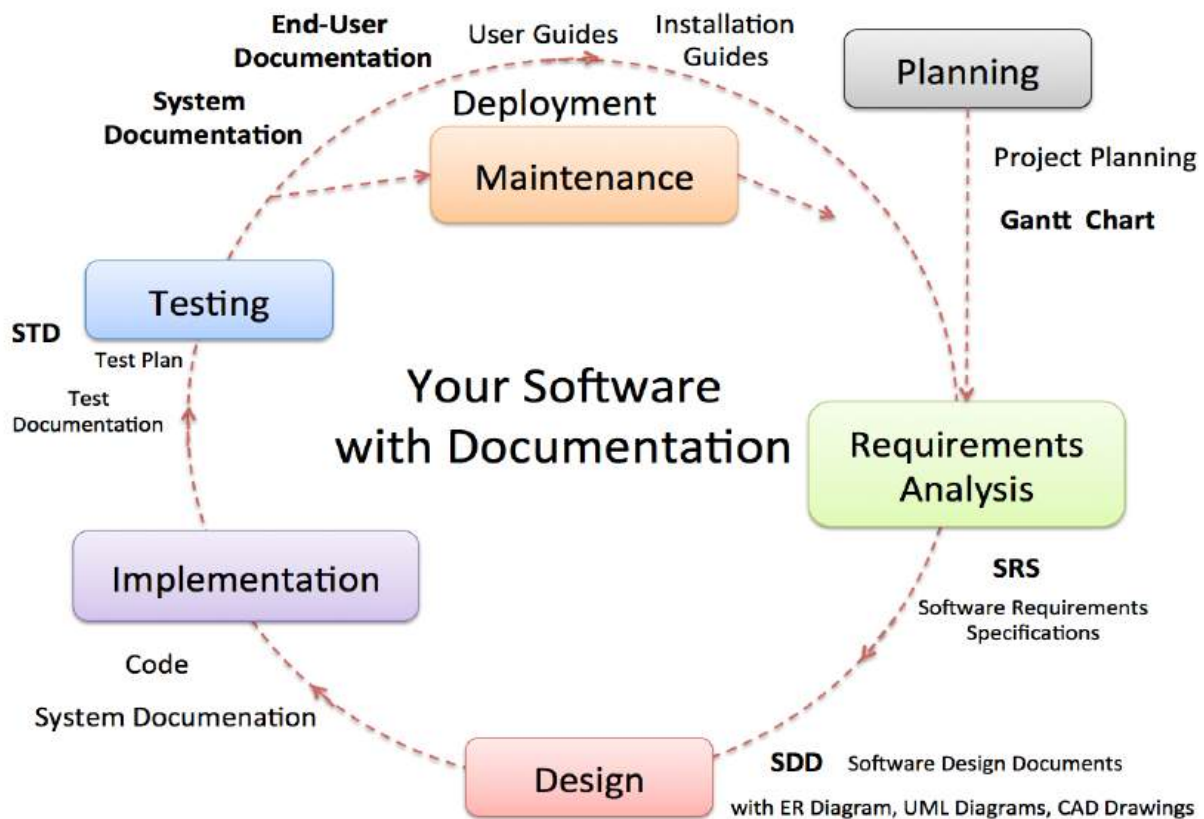
Figure 15-1: Example of Documentation during the SDLC

Write Technical Reports in Microsoft Word: https://youtu.be/ao_eDJOEUkA

Figures and Equations in Word and PowerPoint: https://youtu.be/b9f2bb2yn1Y

Citation and referencing with Microsoft Word: https://youtu.be/IgH7qmLa_L4

Some documents are for internal use inside the software company or inside the development team, while other documents are important for the stakeholders and customers that are going to use the software (Figure 15-2).
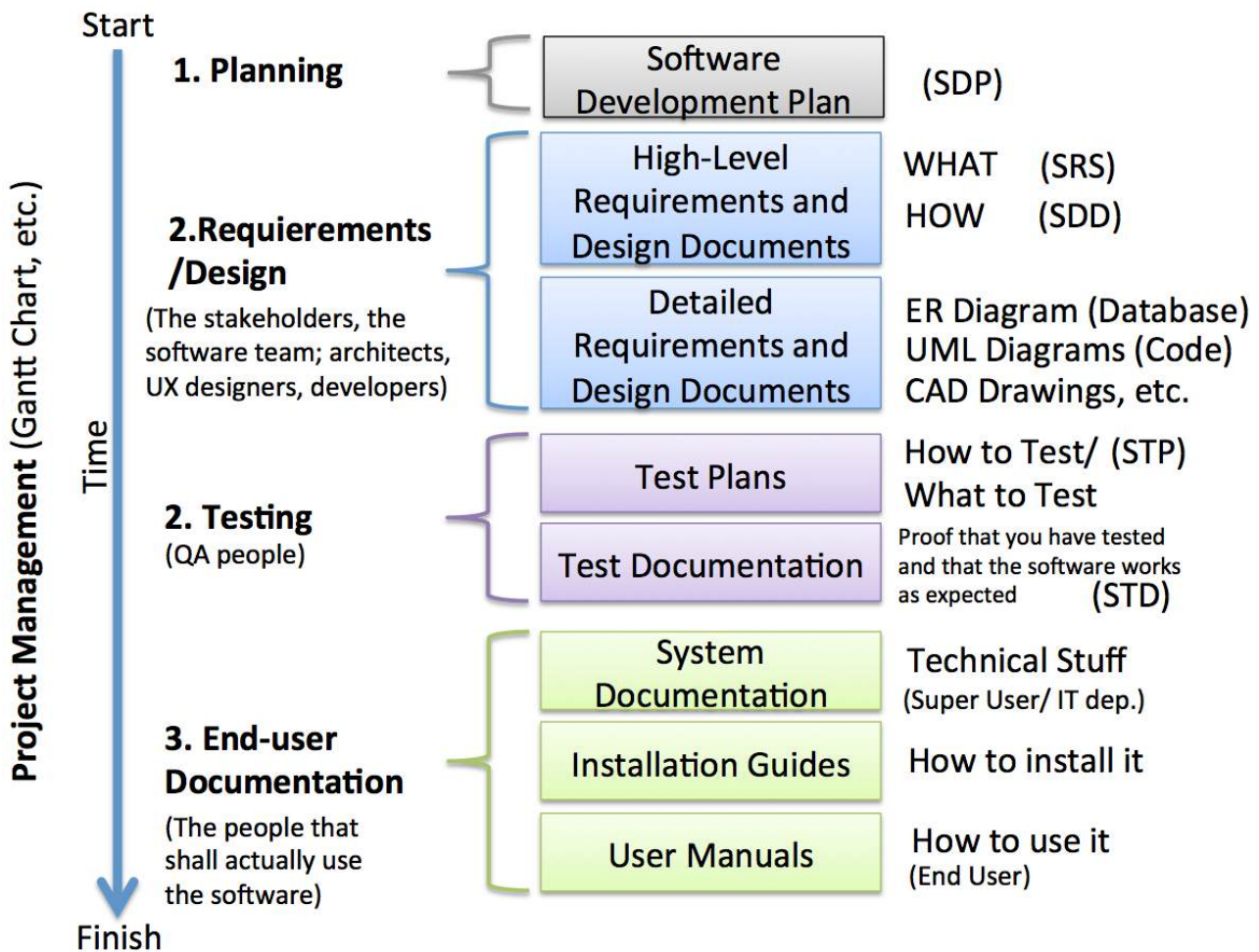
Figure 15-2: Software Documentation

Some important documents in software engineering are:

- **SDP** – Software Development Plan

- **SRS** – Software Requirements Specifications

  o A document stating what the application must accomplish.

- **SDD** – Software Design Document

  o A document describing the design of a software application.

- **STP** - Software Test Plan

  o Documentation stating what parts of an application will be tested, and the schedule of when the testing is to be performed.

- **STD** - Software Test Documentation

  o Introduction, Test Plan, Test Design, Test Cases, Test procedures, Test Log, ..., Summary

See Figure 15-3 for an overview of documentation categories used in a project.
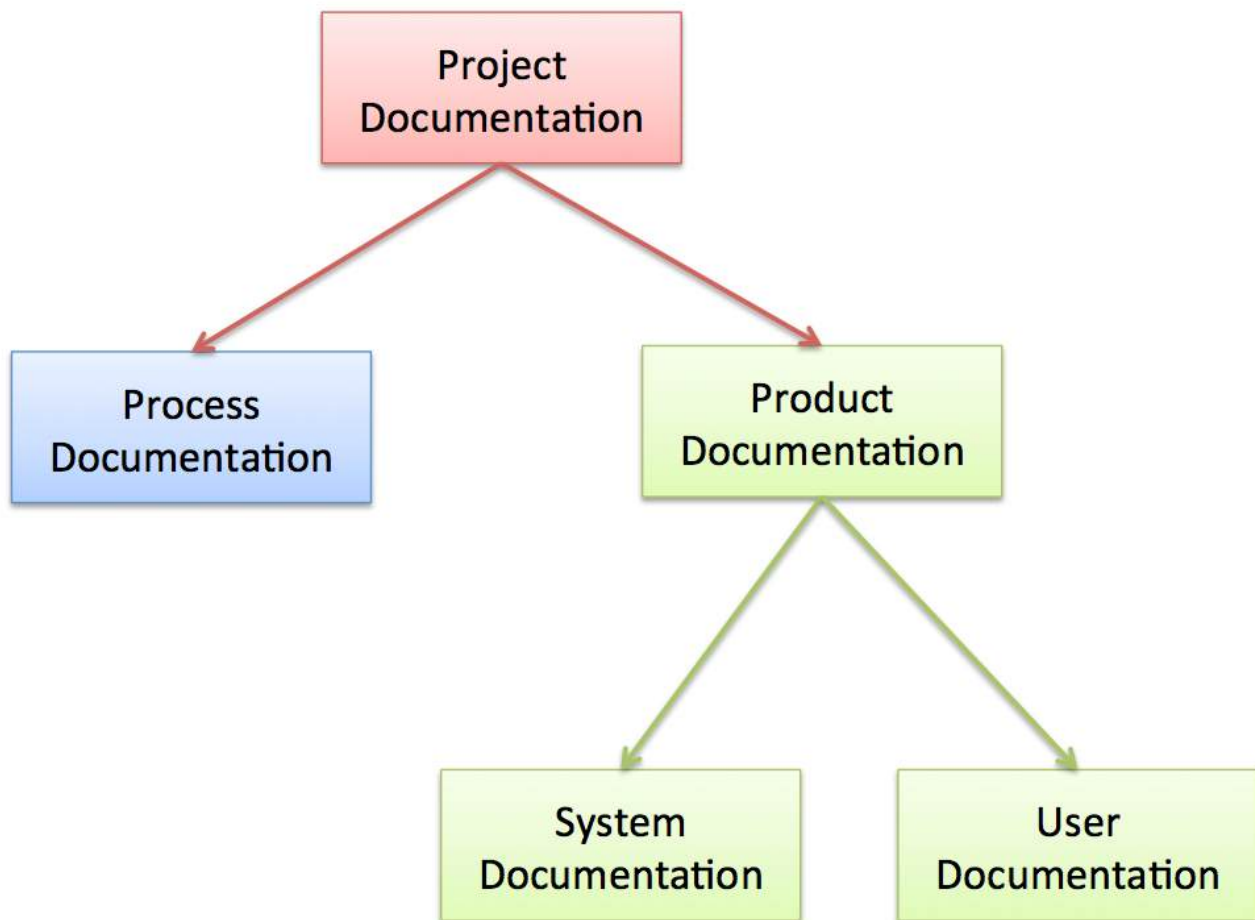
Figure 15-3: Software Project Documentation

Documentation produced during a software Project can be divided into 2 main categories:

- **Process Documentation**

    – These documents record the process of development and maintenance, e.g., Plans, Schedules (e.g., Gantt Charts), etc.

- **Product Documentation**

    – These documents describe the product that is being developed. Can be divided into 2 subcategories:

        - **System Documentation**

            - Used by engineers developing and maintaining the system.

        - **User Documentation**

            - Used by the people that is using the system.

Here are some Software Documentation Requirements:

- Should act as a communication medium between members of the Development Team (Process Documentation)

- Information repository used by Maintenance Engineers (Product Documentation)

- Information for Management to help them Plan, Budget, and Schedule the Software Development Process (Process Documentation)

- Some of the documents should tell users how to use and administer the system (Product Documentation)

- Documents for Quality Control, System Certification, etc. (Process/Product Documentation)

Satisfying these requirements requires different types of documents from informal working documents through professionally produced User Manuals

15.1   Process Documentation

Purpose:

1. Process Documentation is produced so that the development of the system can be managed.

2. It is an essential component of plan-driven approaches (e.g., Waterfall)

3. Agile Approaches: The Goal is to minimize the amount of Process Documentation

We have different categories of Process Documentation:

- **Plans, estimates, and schedules**. These are documents produced by managers which are used to predict and to control the software process.

- **Reports**. These are documents which report how resources were used during the process of development.

- **Standards**. These are documents which set out how the process is to be implemented. These may be developed by organizational, national, or international standards.

- **Working papers**. These are often the principal technical communication documents in a project. They record the ideas and thoughts of the engineers working on the project, are interim versions of product documentation, describe implementation strategies and set out problems which have been identified. They often, implicitly, record the rationale for design decisions.

- **E-mail messages, wikis, etc.** These record the details of everyday communication between managers and development engineers.

15.2   Product Documentation

Purpose:

- Describing the delivered software product

- Unlike most process documentation, it has a relatively long life. It must

- Evolve in step with the product that it describes. Product documentation includes.

    – User documentation, which tells users how to use the software product,

    – System Documentation, which is principally intended for maintenance engineers.

15.2.1   System Documentation

The system documentation describes how the system is designed and how it works in detail.

1. System documentation includes all the documents describing the system itself from the requirements specification to the final acceptance test plan.

2. Documents describing the design, implementation and testing of a system are essential if the program is to be understood and maintained.

3. Like user documentation, it is important that system documentation is structured, with overviews leading the reader into more formal and detailed descriptions of each aspect of the system.

In Figure 15-4 we see an overview of different product documentation and readers of such documents.
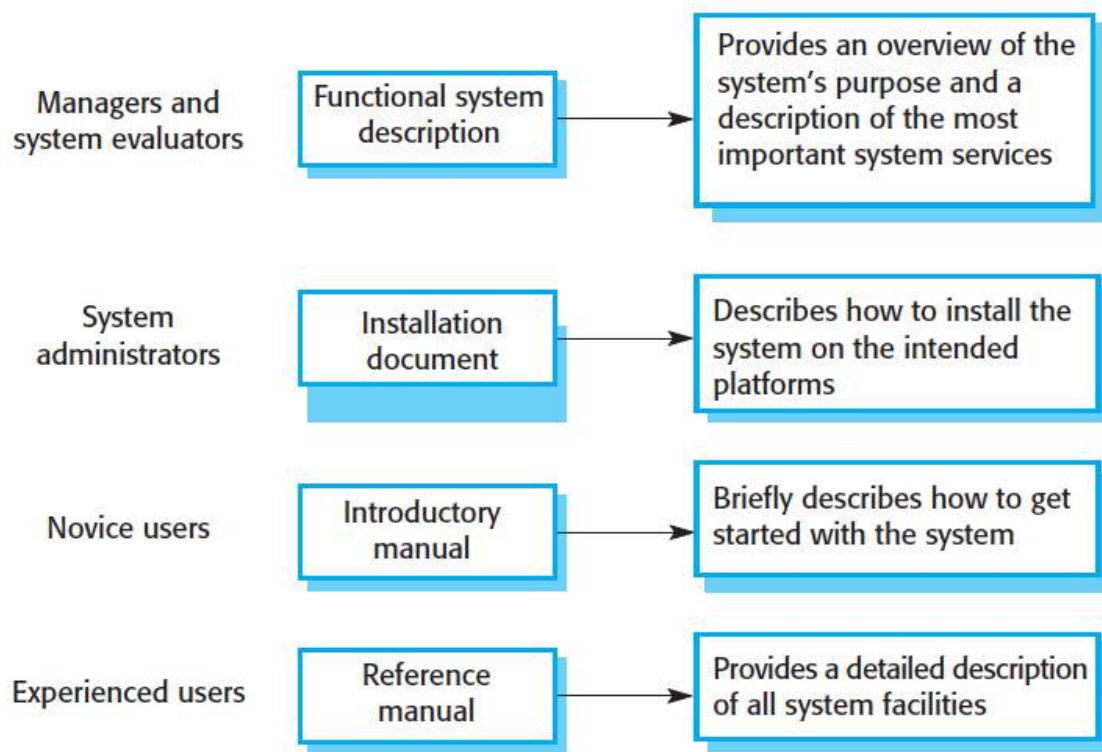
Figure 15-4: Product Documentation Types & Readers [1]

For large systems that are developed to a customer's specification, the system documentation should include:

- The **requirements** document.

- A document describing the **system architecture**.

- For each program in the system, a description of the architecture of that program.

- For each component in the system, a description of its functionality and interfaces.

- Program **source code** listings, which should be commented on, where the comments should explain complex sections of code and provide a rationale for the coding method used.

    - If meaningful names are used and a good, structured programming style is used, much of the code should be self-documenting without the need for additional comments.

    - This information is now normally maintained electronically rather than on paper with selected information printed on demand from readers.

- **Validation** documents describing how each program is validated and how the validation information relates to the requirements.

    - These may be required for the quality assurance processes in the organization.

- **A System Maintenance Guide**, which describes known problems with the system, describes which parts of the system are hardware and software dependent and which describes how evolution of the system has been considered in its design.

15.2.2   User Documentation

Users of a system are not all the same. The producer of documentation must structure it to cater for different user tasks and different levels of expertise and experience.

It is particularly important to distinguish between end-users and system administrators:

- **End-users** use the software to assist with some tasks.

  – This may be flying an aircraft, managing insurance policies, writing a book, etc. They want to know how the software can help them. They are not interested in computer or administration details.

- **System administrators** are responsible for managing the software used by end-users.

  – This may involve acting as an operator if the system is a large mainframe system, as a network manager is the system involves a network of workstations or as a technical guru who fixes end-users software problems and who liaises between users and the software supplier.

We have different user documentation, such as:

- User Manual
- Installation Guide
- Wiki
- etc.

[http://geek-and-poke.com]
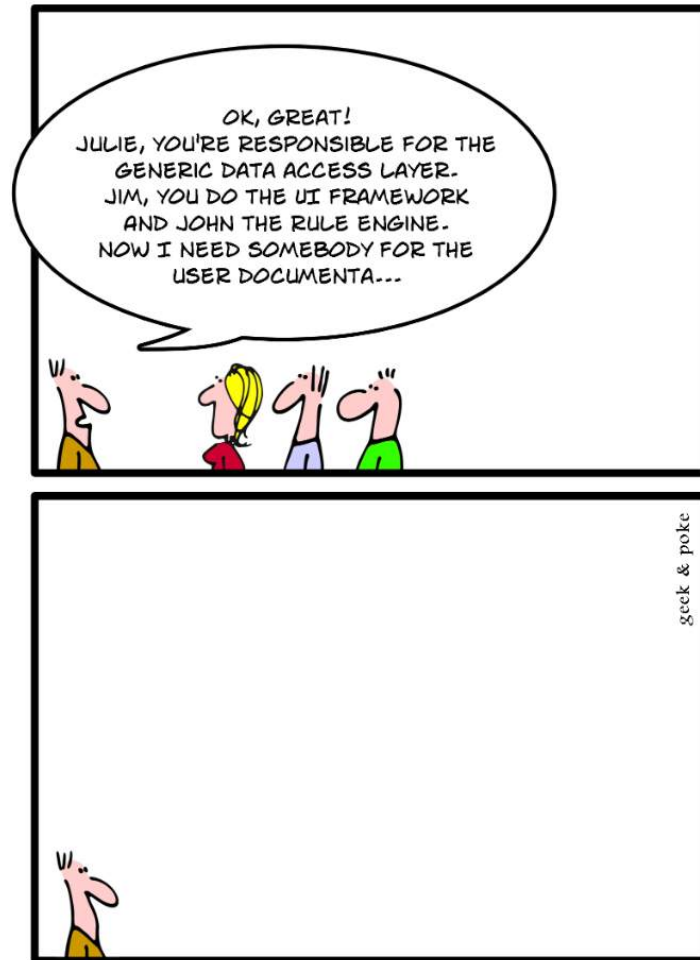
User Manual:

A user guide or user's guide, also commonly known as a manual, is a technical communication document intended to give assistance to people using a system. It is usually written by a technical writer, although user guides are written by programmers, product or project managers, or other technical staff, particularly in smaller companies.

The sections of a user manual often include:

- A Cover page

- A Title page and copyright page

- A Preface, containing details of related documents and information on how to navigate the user guide

- A Table of Contents page

- **A guide on how to use at least the main functions of the system (Text + Screen Shots)**

- A troubleshooting section detailing possible errors or problems that may occur, along with how to fix them

- A FAQ (Frequently Asked Questions)

- Where to find further help, and contact details

- A glossary and, for larger documents, an index

15.3   Setup & Distribution

As mentioned earlier we have two categories of software: generics products and customized products.

If we have a generic product, it is especially important that the customers can install the software you create.

For e.g., web products setup and distribution to the end user computers are not necessary, since the software is installed on a Web Server and can be accessed through an ordinary web browser.

Installation is about:

- Package the software

    - Executable files

    - Create installation packages

        - InstallShield, etc. (lots of tools available)

- Make it available (nowadays over Internet or on DVD)

- Give the customer turn-key installers, which will:

    - Check the system for missing dependencies or drivers etc. (e.g., Your software may need .NET X.x, etc.)

    - Install the software on the system

    - Set up any necessary license information, license managers, etc.

16 Software Maintenance

16.1 Introduction

Software Maintenance is about:

- Software has bugs (Bug /Support incidents need to be tracked and followed up -> A good tool is needed).
- New features are required.
- Circumstances change. Therefore, the software has changed. Who changes it?
- Development teams have broken up, maintenance may be done by different company!
- Repeated change leads to architectural degradation. Old systems may have been degraded from the start!
- Software rots. Even with no code changes, the systems change, and eventually you can't compile the software.

Software Maintenance is defined as [12]: "The process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment".

40-90% of the software life cycle cost is about maintenance.

Examples:

- The Y2K problem
- New versions of the OS often require adjustment to your software
- New requirements and customer needs

We may divide into 2 different types of Maintenance:

- **Repair**

  o Fixing defects/bugs

- **Enhancement**

  o New Requirements

  o Change in Design or Implementation (No functional change)

16.2 Categories

Again, we can divide maintenance into 4 categories:

- Corrective maintenance
- Adaptive maintenance
- Perfective maintenance
- Preventive maintenance

In Figure 16-1 we see an overview of the different software maintenance categories [12].

Figure 16-1: Software Maintenance Categories

In Table 9-1 we see the differences between these categories.

Table 16-1: Software Maintenance Overview

Maintenance	Description
Corrective	Repair of defects relative to existing requirements. These defects are typically discovered by customers as they start using your software.
Adaptive	Adapt your software to changes in the operating environment, e.g., when a new OS is released or a new version of the hardware. As software systems evolve, it is very likely that changes will happen in the external environment (OS, hardware, etc.) your software depends on.
Perfective	New features based on new user requests. The software must continuously adapt to new needs, or your software will become useless.
Preventive	Changes in your software to make it easier to maintain. Changes from Corrective, Adaptive and Perfective make your software more complex, more difficult to maintain, etc. Preventive maintenance in form of Refactoring should be done on a regular basis

While [1] only divide into 3 different categories, see Figure 16-2.

Software Maintenance

3 Categories (according to I. Sommerville, *Software Engineering*):

1. Fault Repairs
- Fixing Errors after Sofware is released

2. Environmental Adaption
- OS,Hardware, etc. changes

3. Functionality Addition
- The System Requirments change

Fault repair (17%)

Environmental adaptation (18%)

Functionality addition or modification (65%)

System 1

System 2

0   50   100   150   200   250   300   350   400   450   500 $

Development costs   Maintenance costs

I. Sommerville, *Software Engineering*, 9 ed.: Pearson, 2010.
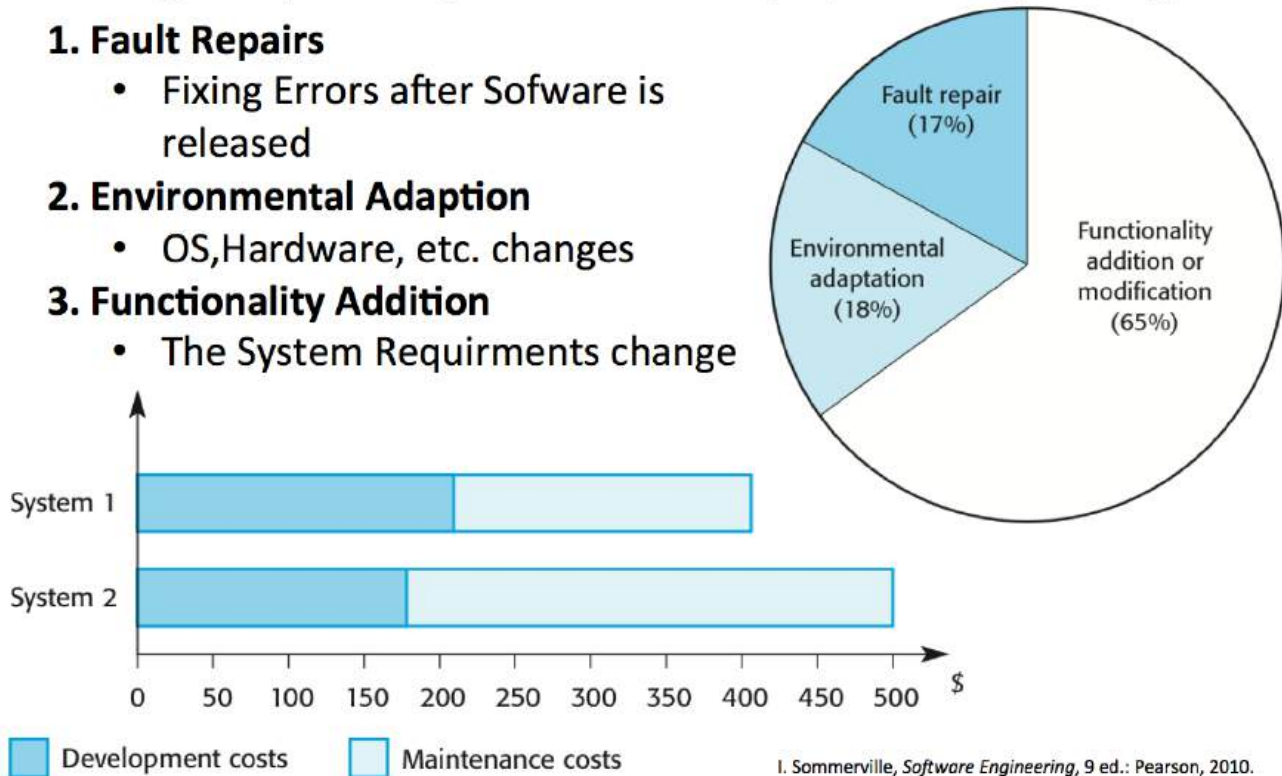
Figure 16-2: Maintenance Categories [1]

Part 3 : Platforms & Architecture

In this part, we give an overview of tools used (and needed) in modern software engineering, like collaboration tools, source code control tools, programming platforms, frameworks, languages, etc.

17 Software Platforms

17.1 Introduction

We have lots of different platforms today, here we will discuss the most common platforms used today. They are:

- **Desktop:** We have different Desktop platforms such as Windows, Mac OS X, Linux, etc. To create applications for Windows we can use, e.g., Visual Studio and C#. To create applications for Mac OS X we can use, e.g., Xcode and Objective-C/Swift. LabVIEW exists for both Windows, Mac OS X and Linux.
- **Mobile:** Today we have 3 major Mobile platforms: iOS (iPhone and iPad), Android (running on different Smartphones and Tablets) and Windows 8 (Tablets)/Windows 8 Phone (Smart Phones).
- **Web:** Web applications run inside a web browser, such as e.g., Internet Explorer, Chrome, Safari, Opera or Firefox. In the simplest form we can use HTML. For more dynamic web pages we can use ASP.NET, PHP, JavaScript, AJAX, etc.

Here are some examples:

Desktop

- Windows
- Mac OS
- Linux

Web

- ASP.NET
- PHP
- IIS
- Apache
- HTML
- JavaScript
- AJAX

Mobile Devices

- iOS (iPhone, iPad, iPod)
- Android
- Windows Phone

Server-side

- Databases
- Web Servers

Figure 17-1 shows some advantages and disadvantages with the different Platforms.
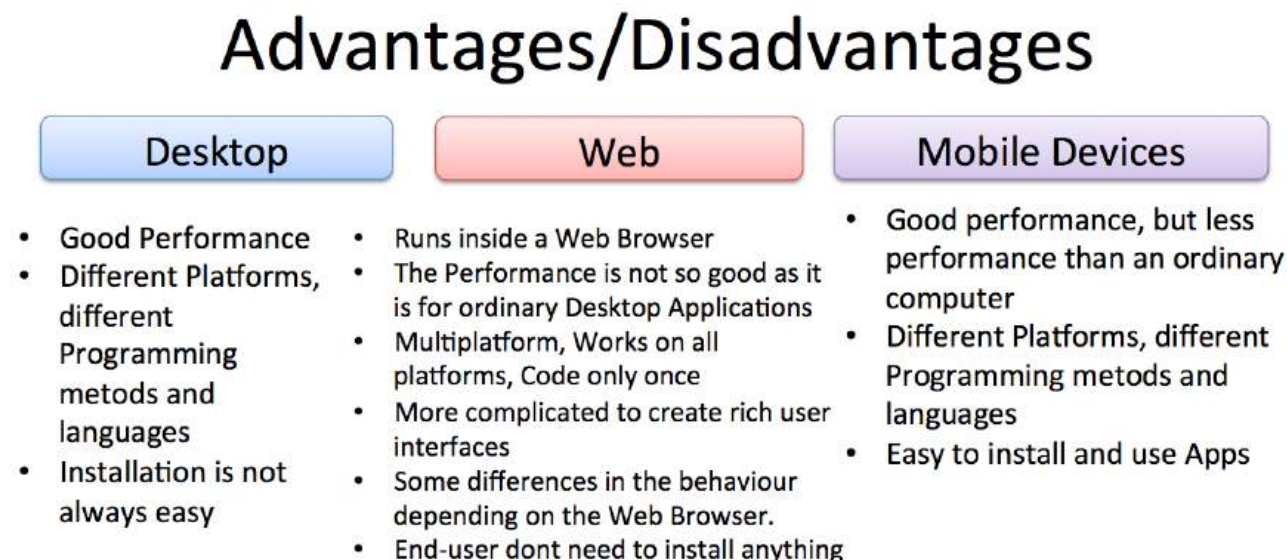
Figure 17-1: Advantages/Disadvantages with different Platforms

17.2   Platform Vendors

The 3 main vendors of such platforms are Microsoft, Apple, and Google. They all deliver platforms for Desktop and Mobile systems, but they have different approaches, see Figure 17-2.
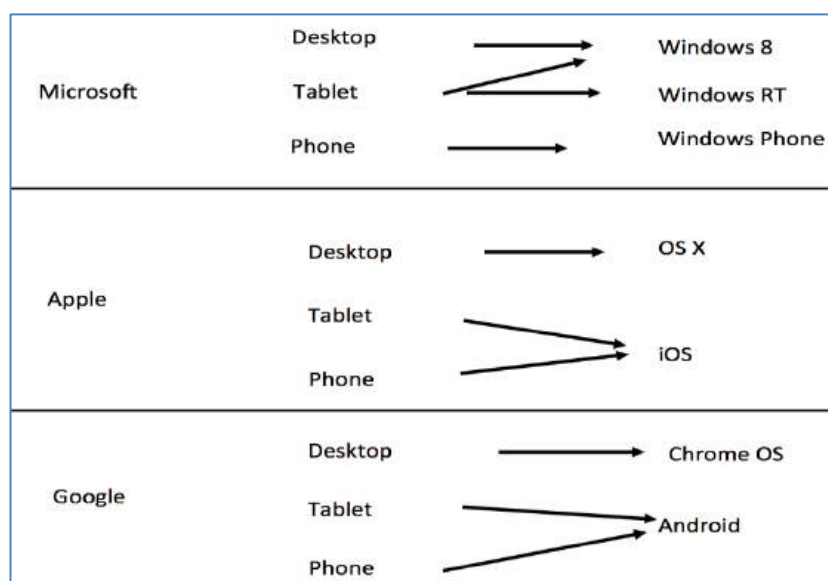
Figure 17-2: Desktop and Mobile Platforms Vendors

17.3   Desktop

On the desktop, we have 3 main platforms, namely Windows, Linux and macOS. In addition, we have Chrome OS from Google as a 4. alternative.
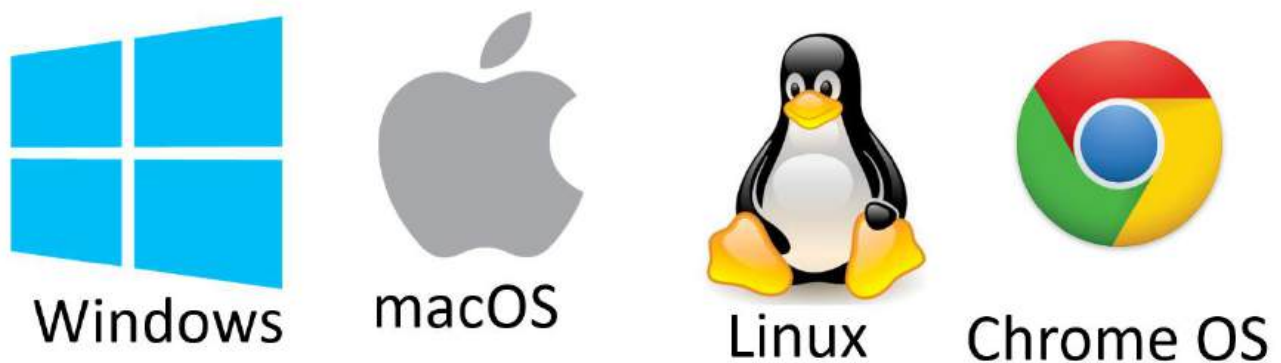
Figure 17-3: Desktop Platforms: Windows, Linux, macOS, Chrome OS

We have lots of development tools for these platforms, see Figure 17-4. These will be discussed in more details later.

Figure 17-4: Development Tools available for different Desktop Platform

17.3.1   Windows

Microsoft Windows is a series of graphical interface operating systems developed, marketed, and sold by Microsoft.
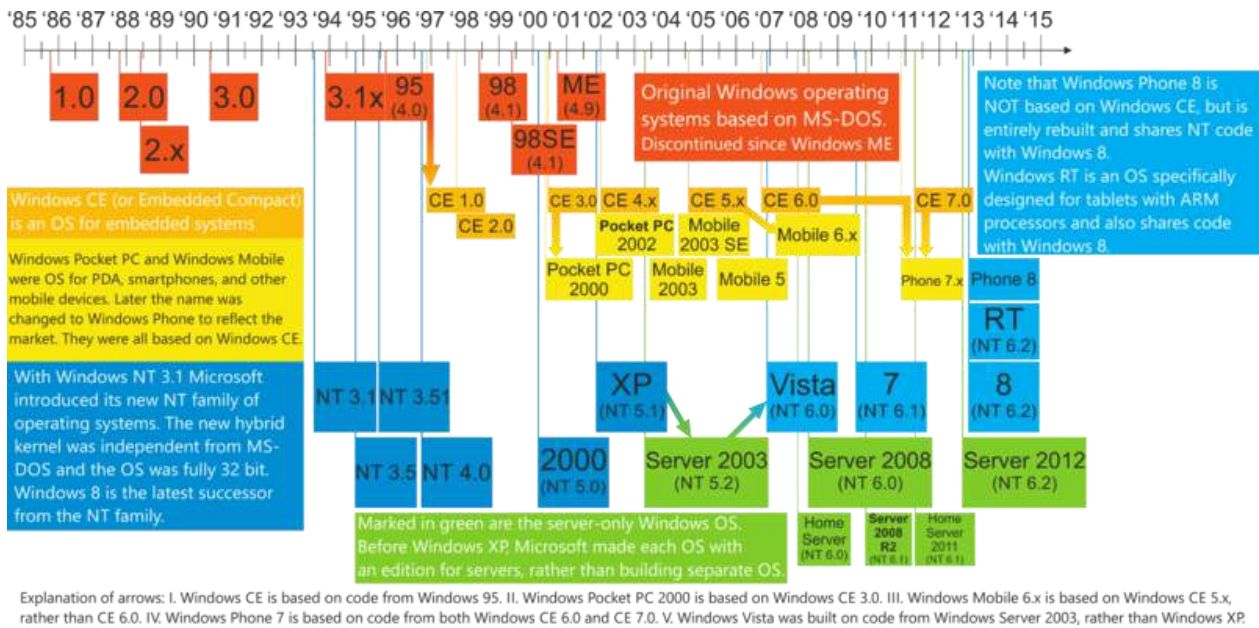
Figure 17-5: Windows Release History

There are a lot of Windows releases, some of them are:

- Windows 3
- Windows 95
- Windows NT
- Windows XP
- Windows 7
- Windows 8
- Windows 10
- Windows 11

In Figure 17-6 we see the first Windows version (Windows 1.0).
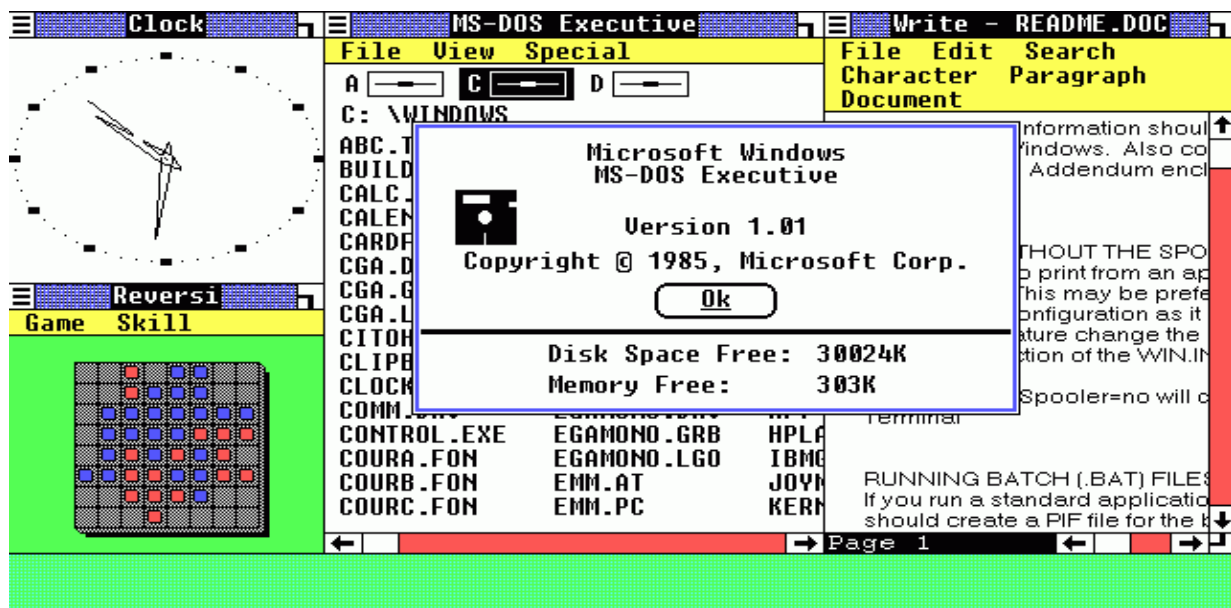
Figure 17-6: Windows 1.0

With Windows 8 Microsoft changes the user experience dramatically, see Figure 17-7.

Figure 17-7: Windows 8

17.3.2   macOS

macOS (Figure 17-8) is developed by Apple Inc. macOS runs only on Mac Computers from Apple. It is a UNIX-based OS based on NeXT OS (Apple bought NeXT, and Steve Jobs returned to Apple as CEO) because Apples classic OS from 1984 (the first Macintosh) and later (Mac OS 9) was lacking behind – they needed a fresh start.

In macOS software can be deployed to Mac App Store for easy installation.

A popular development platform on macOS is the Xcode IDE and the Swift programming language.

Figure 17-8: macOS

The different releases of Mac OS X is named after big cats like Puma, Jaguar, Tiger Leopard, Lion, see Figure 17-9.

Figure 17-9: Mac OS X Release History

The Mac OS X 10.9 was called "OS X Mavericks", so from this version they have stopped using names from big cats (all the names were taken?).

The latest version is called "macOS Catalina", so they have switched from big cats to famous places in California, USA. They have also stopped using OS X, now it is called macOS.

17.3.3   Linux

Linux is a UNIX-like operation system. It is a Free/Open-Source software platform. Linus Torvalds is the founder of the Linux kernel.   Linux was originally developed as a free operating system for Intel x86-based personal computers. It has since been ported to more computer hardware platforms than any other operating system.

From Figure 17-10 we see that both Linux and Mac OS X have their origins from the UNIX platform.
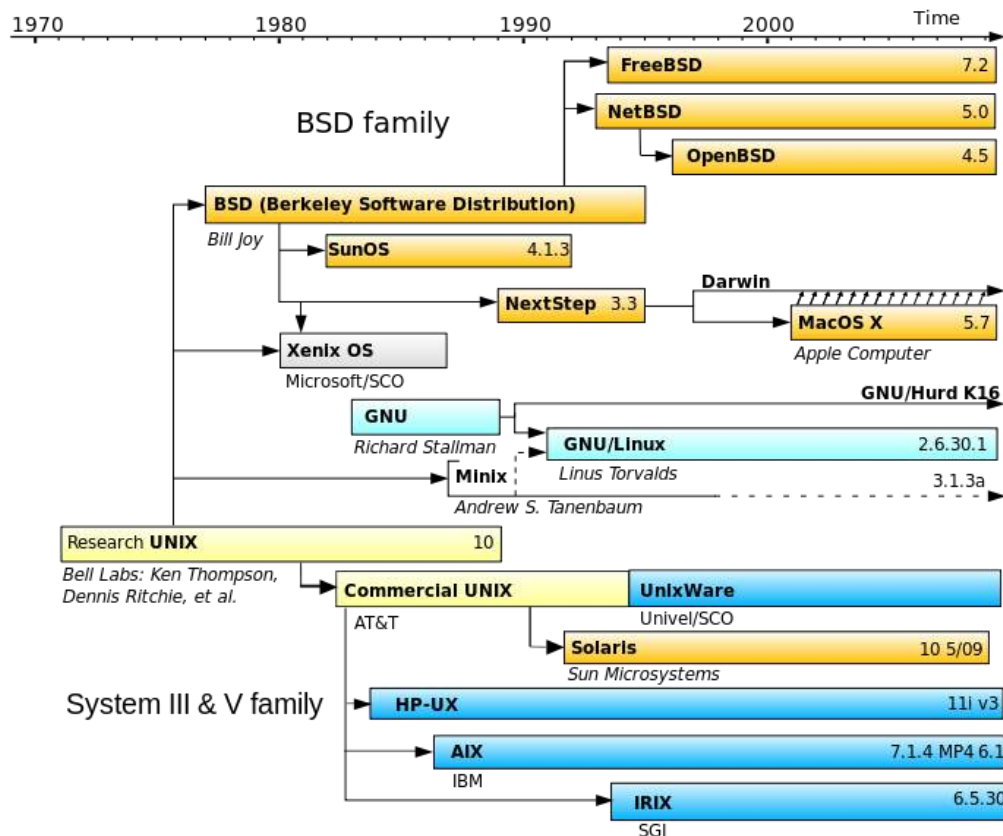
Figure 17-10: UNIX History

Different Vendors/different versions/distributions, e.g.:

- Fedora
- Red Hat Linux
- SUSE
- Mandriva
- Ubuntu
- etc.

The penguin symbol (Figure 17-11) is typical for Linux.

Figure 17-11: The Linux Mascot

17.4   Web

The Web have become more and more important as a platform for developing software.

Here are some keywords:

- HTML
- JavaScript
- ASP.NET
- PHP
- Internet Information Services (IIS)
- Apache

A Web Browser and HTML are the foundation for web pages. HyperText Markup Language (HTML) is the main markup language for creating web pages and other information that can be displayed in a web browser.

In Figure 17-12 we see the typical web architecture, including web browsers, HTML, CSS, JavaScript, and a web server for hosting the web pages.

Figure 17-12: Web Architecture

In Figure 17-13 we see the triangle of web programming. You cannot create a modern web page without knowing the basics of HTML, CSS and JavaScript. They are the basic building blocks when creating web pages.
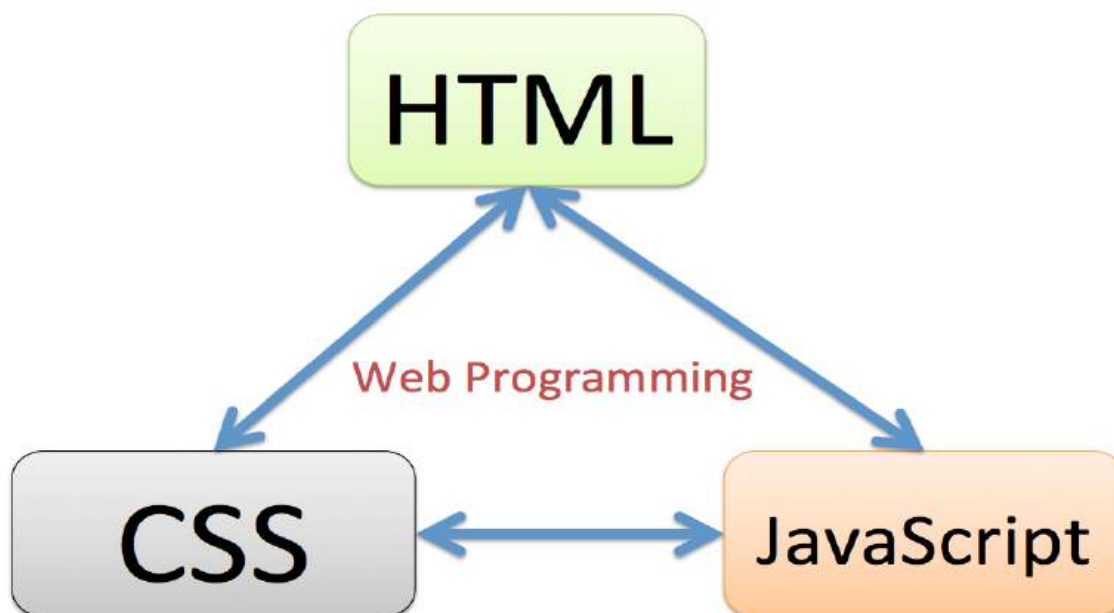
Figure 17-13: The triangle of Web Programming

HyperText Markup Language (HTML) is the visual appearance of a Web Site. All Web Browser understand HTML. The latest version is HTML 5. CSS (Cascading Style Sheets) define how to display

HTML elements.   CSS is used to control the style and layout of multiple Web pages all at once. JavaScript is the programming language of the Web. All modern HTML pages are using JavaScript.

Some relevant videos:

Web Programming Overview: https://youtu.be/plRBYKbQSuE

Create Web Pages with HTML and CSS: https://youtu.be/DUEHx7l5a3Y

17.4.1   Web Servers

Web Servers are used to host web sites and web pages. The web server can refer to either the hardware (the computer) or the software (the computer application) that helps to deliver web content that can be accessed through the Internet.

The following web servers (software) are very popular today:

- Internet Information Services (IIS) (included with Windows)
- Apache
- Nginx (pronounced "engine x")

See Figure 17-14.
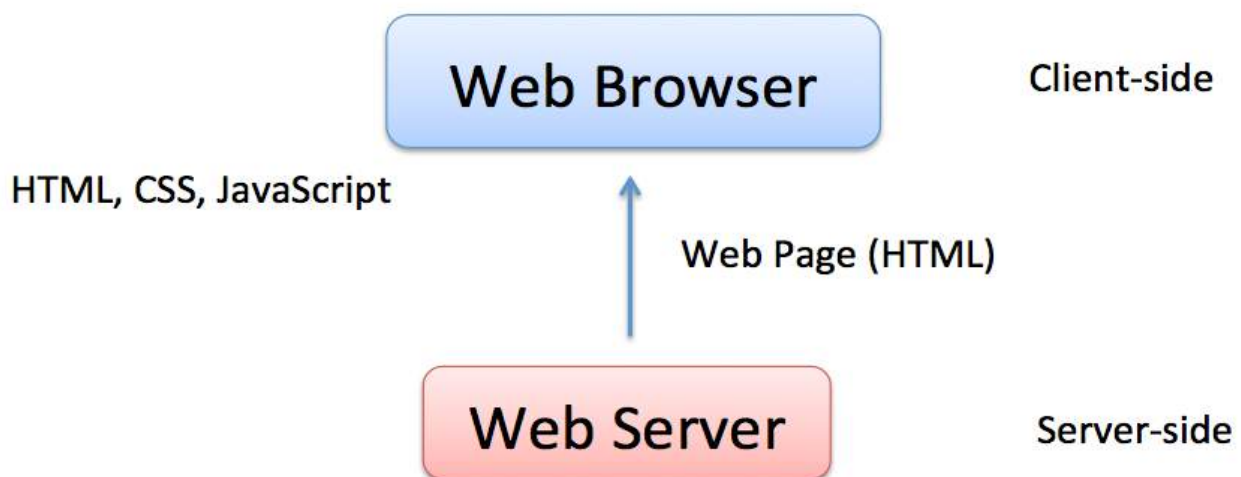
Figure 17-14: Web Server

17.4.2   Web Frameworks

ASP.NET is a web application framework developed by Microsoft to allow programmers to build dynamic web sites, web applications and web services.

ASP.NET is part of the Visual Studio package.

It was first released in January 2002 with version 1.0 of the .NET Framework and is the successor to Microsoft's Active Server Pages (ASP) technology. ASP.NET is built on the Common Language Runtime (CLR), allowing programmers to write ASP.NET code using any supported .NET language, such as C# and VB.NET.

ASP.NET web pages or webpages, known officially as Web Forms], are the main building block for application development. Web forms are contained in files with an ".aspx" extension.

See Figure 17-15.
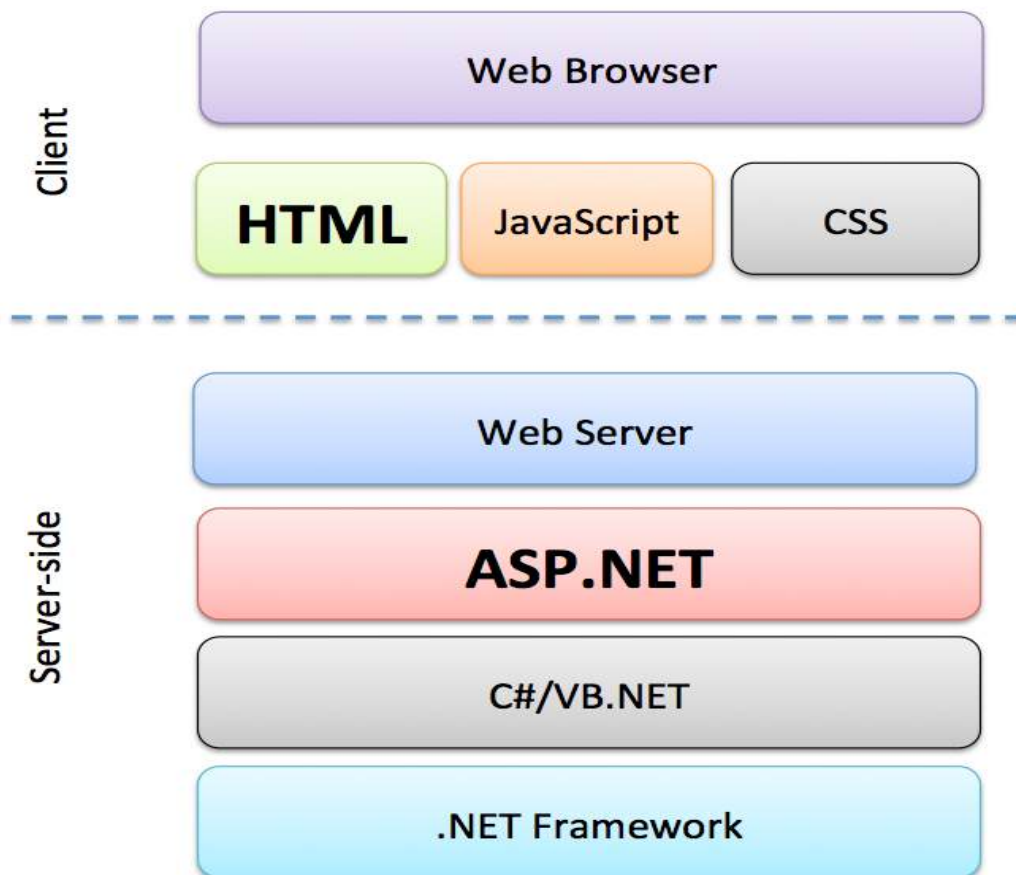
Figure 17-15: ASP.NET

For more information about ASP.NET, please see the Tutorial "ASP.NET and Web Programming" [17].

17.4.3   ASP.NET Core

The new .NET Core is a lightweight cross-platform subset of the full .NET Framework. See Figure 17-16.
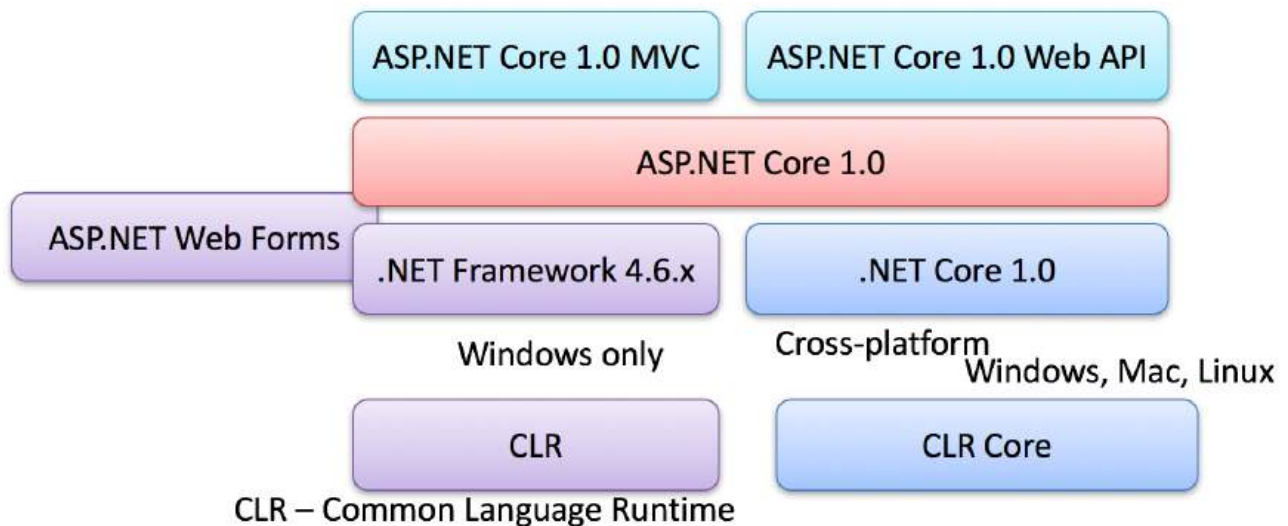
Figure 17-16: ASP.NET Core

Some relevant videos:

 ASP.NET Core - Hello World: https://youtu.be/lcQsWYgQXK4

 ASP.NET Core – Introduction: https://youtu.be/zkOtiBcwo8s

17.4.4   Web Scripting Languages

A scripting language is a lightweight programming language. JavaScript and PHP is programming code that can be inserted into HTML pages. JavaScript inserted into HTML pages, can be executed by all modern web browsers.

JavaScript:

JavaScript is THE scripting language of the Web. JavaScript is used in billions of Web pages to add functionality, validate forms, communicate with the server, and much more.

To insert a JavaScript into an HTML page, use the <script> tag. The <script> and </script> tells where the JavaScript starts and ends. The lines between the <script> and </script> contain the JavaScript. Below we see an example.

```
<!DOCTYPE html>
<html>

<head>
<script>
function myFunction()
{
```

```
      document.getElementById("demo").innerHTML="My First JavaScript
Function";
}
</script>
</head>

<body>
<h1>My Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>
</body>
</html>
```

PHP:

PHP is a server scripting language and is a powerful tool for making dynamic and interactive Web pages. PHP is free to use (open source) and it is widely used today.

The PHP code is merged between the HTML code, and the PHP code is executed on the web server and translated to pure HTML syntax. Below we see an example.

```
<!DOCTYPE html>
<html>
<body>

<h1>My first PHP page</h1>

<?php
echo "Hello World!";
?>

</body>
</html>
```

Typically, you have a combination of PHP code, HTML and JavaScript on a web page.

PHP runs on different platforms (Windows, Linux, Unix, Mac OS X, etc.) and it is compatible with almost all web servers used today (Apache, IIS, etc.). PHP has support for a wide range of databases as well.

17.5   Mobile Devices

Today we have the following Mobile platforms:

- iPhone (iOS)/iPad OS
- Android

Below we will give a short overview of these platforms.

In general, we have 2 different kinds of apps for mobile devices (Figure 17-17), i.e., we can distinguish between "native apps" and "web apps". Web apps are created using HTML 5

technology and run inside a standard web browser, while native apps are created specifically for a specific device or platform, such as an iOS device, Android device, etc. If you want to support more than one platform, you need to develop and maintain one app for each of these platforms.

Different platforms need different programming methods and languages, but native apps provide better performance and usability compared to web-based apps. Native apps can use APIs that is provided by the vendor, they have access to built-in sensors, GPS, etc.
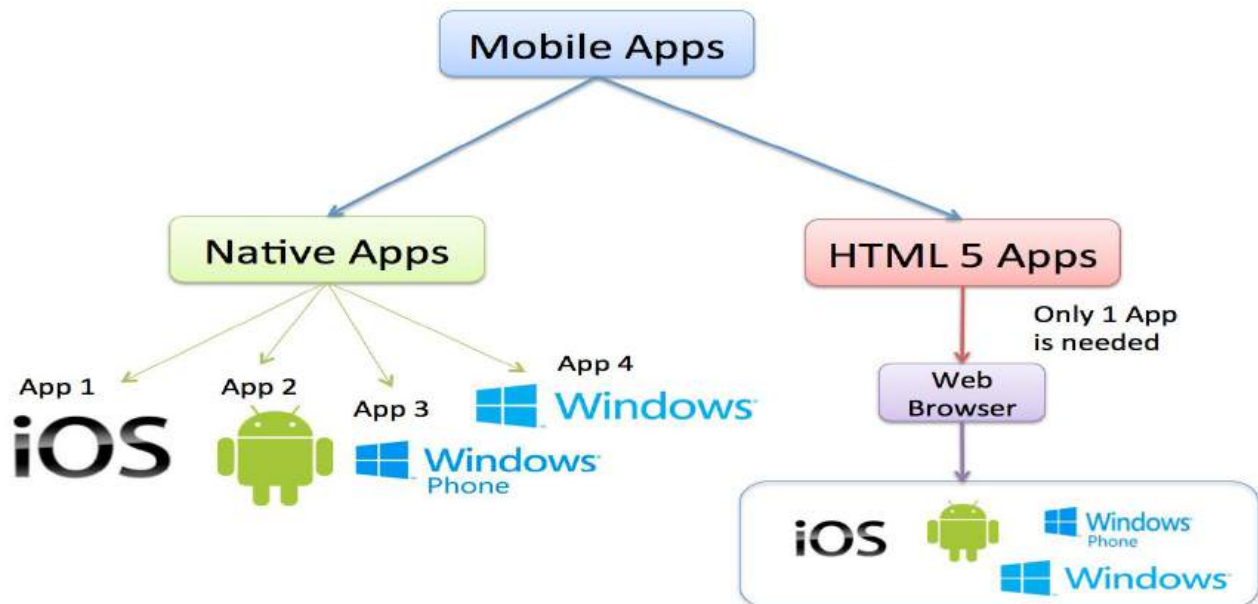
Figure 17-17: Native vs. Web Apps

Web apps are not real applications; they are websites that, in many ways, look and feel like native applications, but are not implemented as such. They are run by a browser and typically written in HTML5. Users first access them as they would access any web page: they navigate to a special URL and then have the option of "installing" them on their home screen by creating a bookmark to that page.

Web apps became popular when HTML5 came around and people realized that they can obtain native-like–functionality in the browser. Today, as more and more sites use HTML5, the distinction between web apps and regular web pages has become blurry.

Native Apps vs. Web Apps:

Some important aspects of native apps and web apps:

- Native apps live on the device and are accessed through icons on the device home screen.
- Native apps are installed through an application store (such as Google Play or Apple's App Store).
- They are developed specifically for one platform, and can take full advantage of all the device features — they can use the camera, the GPS, the accelerometer, the compass, contact list, etc.

- Native apps can use the device's notification system and can work offline.

17.5.1 iOS

iOS (see Figure 17-18) is a mobile operating system developed and distributed by Apple Inc. Created in 2007 together with the iPhone. It has been extended to support other Apple devices such as the iPod touch (2007) and iPad (2010). With iOS 7 and later the OS have gone through a large makeover, compared to previous versions.

Figure 17-18: iOS

iOS is derived from OS X, which is the operation system used on Apple Mac computers.

Apps can be downloaded from the App Store. To create Apps, you use the Xcode IDE and the Objective-C or Swift programming language. You need the iOS SDK, which is included with Xcode.

Xcode is only available for Mac OS X, this means you need a Mac computer to create apps for the iOS platform.

17.5.2 Android

Android (see Figure 17-19) is a Linux-based operating system designed for mobile devices such as smartphones and tablets. Android is developed by Google.

The first Android phone was sold in 2008.

Figure 17-19: Android

Android works on hardware from different vendors.

The source code for Android is available under a free and open-source software license, which means everybody may change it and create their own version of it. Vendors like Samsung, etc. do this.

You use the Eclipse IDE and Java programming language to create apps for Android. Apps can be downloaded from Google Play, Amazon AppStore for Android, etc. Google Play is the official App Store for Android.

Figure 17-20 gives an overview of the different Android versions:

Figure 17-20 Android Versions

Notice that all Android versions are named after a dessert, a cake, or other sorts of candy. They are also in alphabetical order:

- C (Cupcake)
- D (Donut)
- E (Eclair)
- F (Froyo)
- G (Gingerbread)
- H (Honeycomb)
- I (Ice Cream Sandwich)
- J (Jelly Bean)
- K (KitKat) – Android 4.4x
- L (Lollipop) – Android 5.x
- M (Marshmallow) – Android 6
- N (Nougat) – Android 7
- O (Oro) – Android 8
- P (Pie) – Android 9

Now they have run out of cakes and cookies and the latest versions are just called Android 10 11, 12, etc.

More about Android for Developers here:

http://developer.android.com/

Here you can get detailed information about Android and download resources, development tools, etc., including Android Studio, which is the tool you should use when developing Apps for the Android platform.

Android Studio is the official IDE for developing Apps for Android, but you may use many other IDEs as well. Especially, many use the Eclipse software. In that case, you need to download and install the Android SDK Tools.

Android has become a widely used platform for many kinds of devices, including smartphones, tablets, TV's (Android TV), watches (Android Wear), and even cars (Android Auto).

17.5.3   Windows 10 and Windows 11

Windows 11 is the newest version of Windows. From version 8 (see Figure 17-21), Windows was designed to work on both ordinary computers as well as tablets.
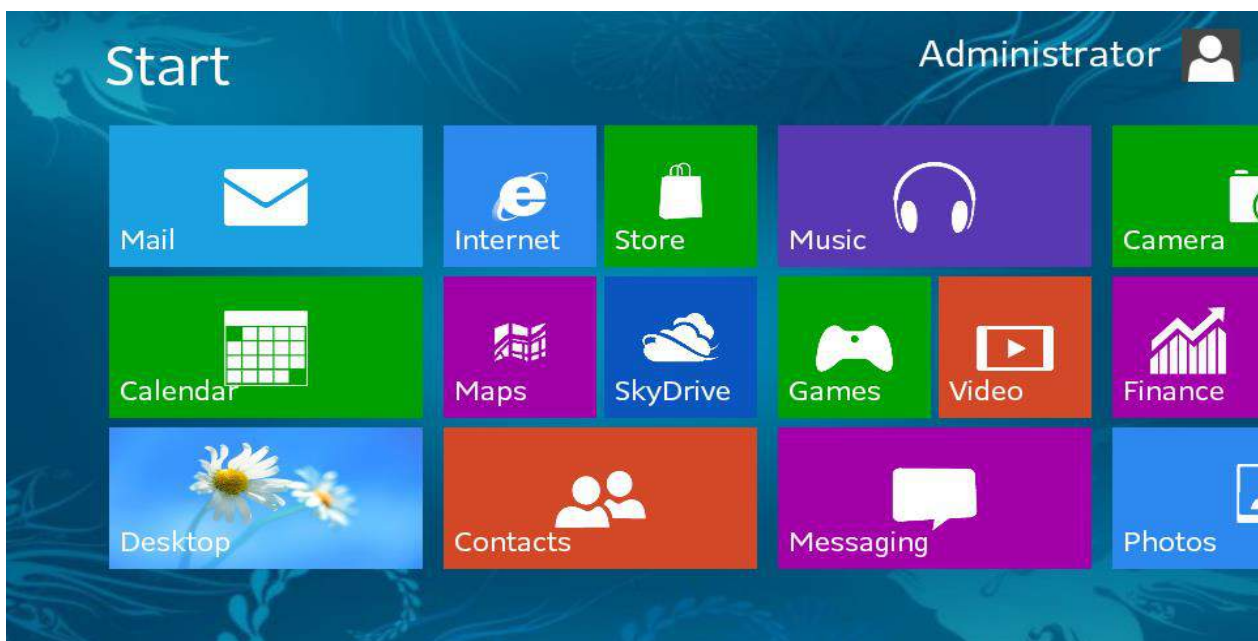
Figure 17-21: Windows 8

Universal Apps may be downloaded from Windows Store which is integrated into Windows 10.

In Figure 17-22 we see an overview of the Windows release history, from Windows 1.0 released in 1985 to Windows 8/Windows RT released in 2012.
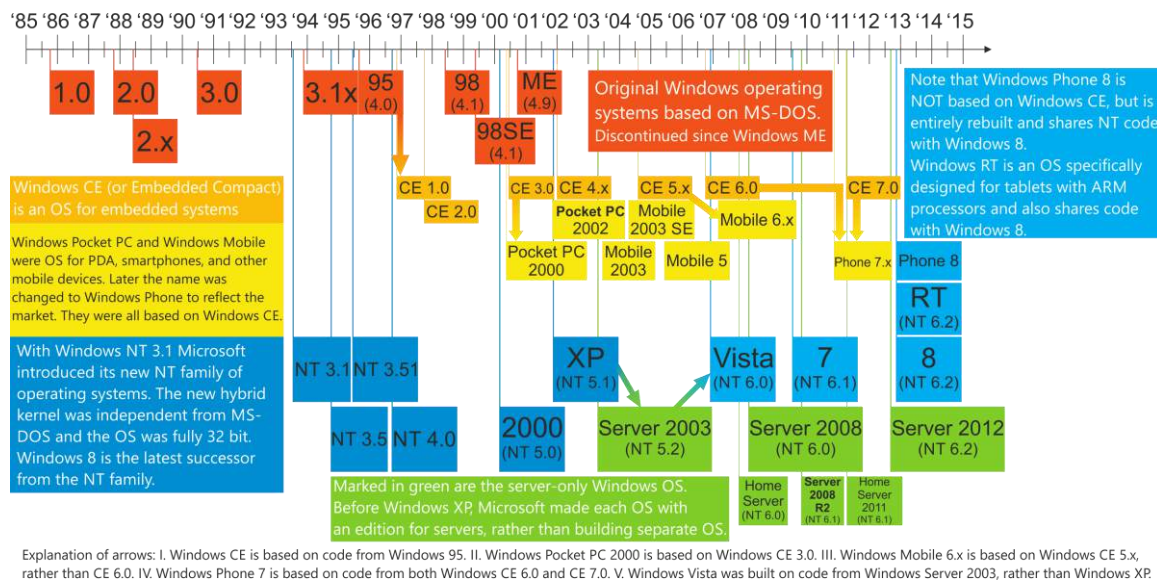
Figure 17-22: Windows Release History

17.6   Cloud Computing

Cloud computing (Figure 17-23) is the use of computing resources (hardware and software) that are delivered as a service over a network (typically the Internet)

Examples:

1. Team Foundation Service

2. iCloud

3. Windows Azure

4. Amazon Web Services

5. Google Cloud Platform

6. etc.
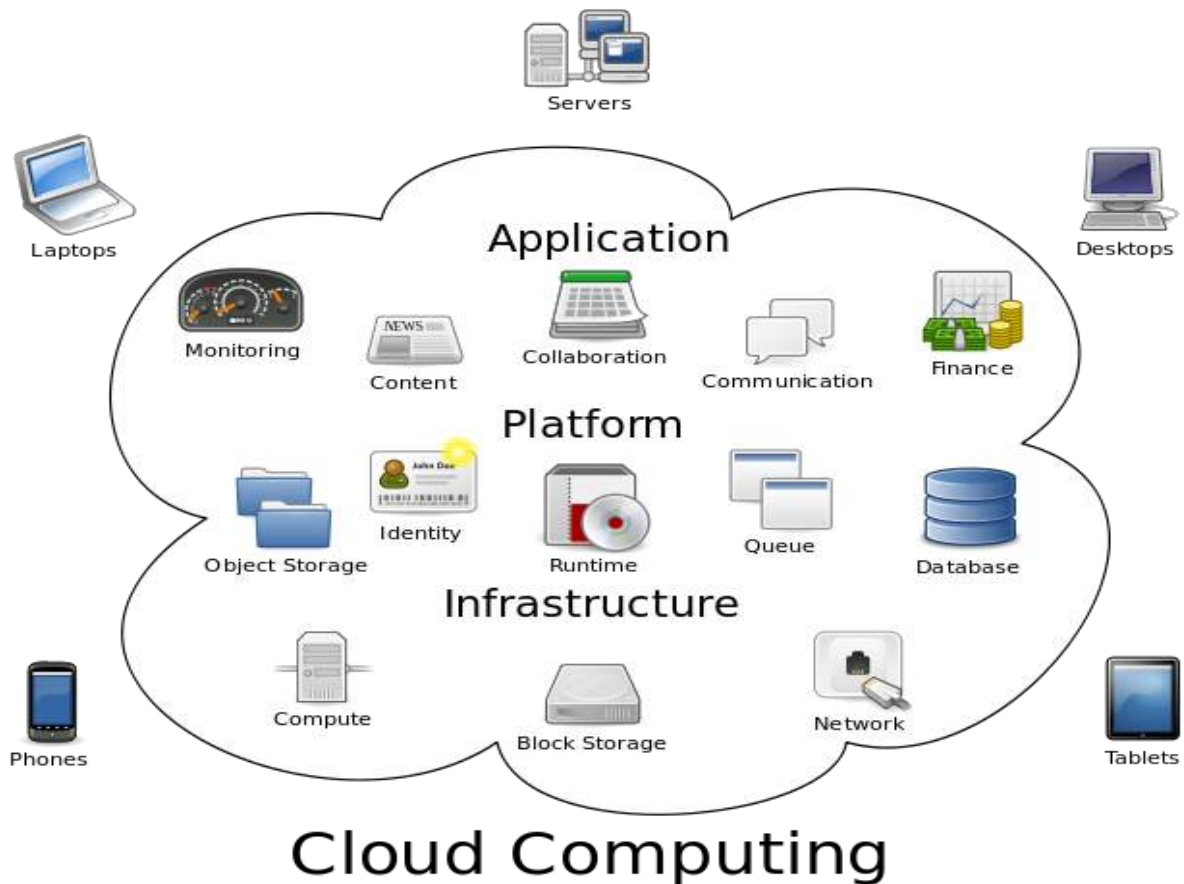
Figure 17-23: Cloud Computing (Wikipedia)

17.7   Open Source

For more information about Open Source, see the following:

http://en.wikipedia.org/wiki/Open-source_software

18 Software Frameworks & Languages

There are probably thousands of different programming languages. Each of these programming languages has good and bad qualities and is preferable in different situations. Some language is good to use when you need to communicate with a database, while others are good to use when you want to develop web applications, etc. So, in most situations you probably need to know and use more than one programming language.

In this chapter, we will discuss some of the most used (probably) programming languages today.

18.1 Object-Oriented Programming (OOP)

Object-oriented programming (OOP) is a programming language model organized around "objects" rather than "actions" and data rather than logic. Historically, a program has been viewed as a logical procedure that takes input data, processes it, and produces output data.

The first step in OOP is to identify all the objects you want to manipulate and how they relate to each other, an exercise often known as data modeling. Once you've identified an object, you generalize it as a class of objects and define the kind of data it contains and any logic sequences that can manipulate it. Each distinct logic sequence is known as a method. A real instance of a class is called an "object" or an "instance of a class". The object or class instance is what you run in the computer. Its methods provide computer instructions, and the class object characteristics provide relevant data. You communicate with objects - and they communicate with each other.

Important features with OOP are:

- Classes and Objects
- Inheritance
- Polymorphism
- Encapsulation

Simula was the first object-oriented programming language. Simula was developed in the 1960s by Kristen Nygaard from Norway.

Java, Python, C++, Visual Basic .NET, and C# are popular OOP languages today.

Since Simula-type objects are reimplemented in C++, Java and C# the influence of Simula is often understated. The creator of C++ (1979), Bjarne Stroustrup (from Denmark), has acknowledged that Simula was the greatest influence on him to develop C++.

18.2   Popular Programming Languages

There are probably thousands of different programming languages today. In Figure 18-1 we see some of them.
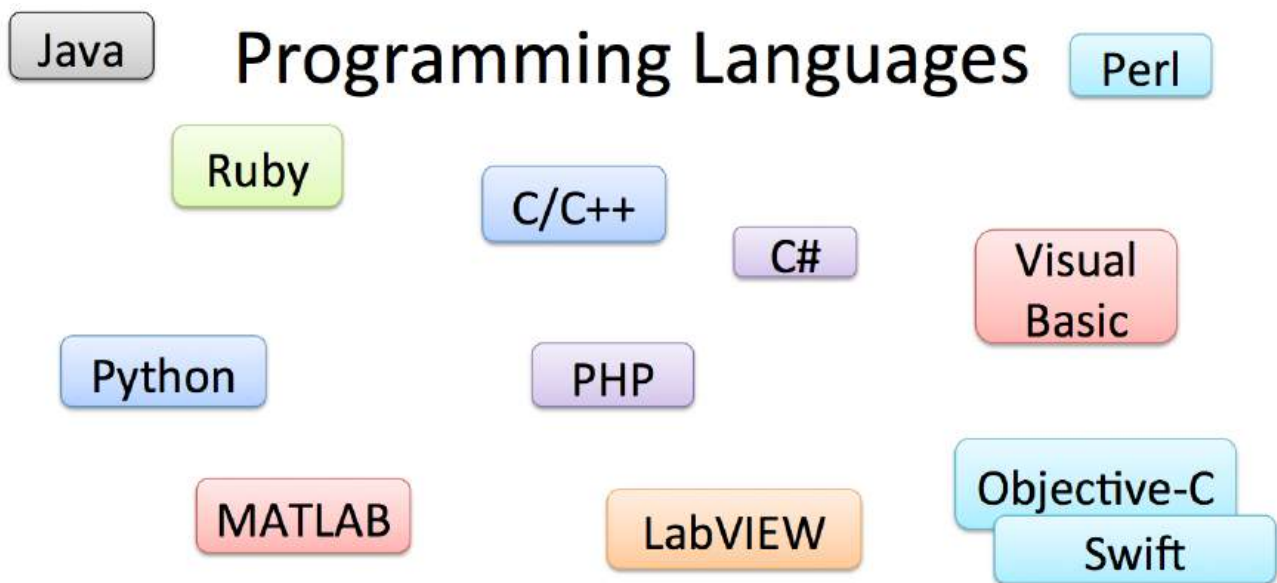
Figure 18-1: Programming Languages

Here is a list of some of the most popular programming languages in use today.

- C, C++
- C#
- Java
- Visual Basic
- Perl
- Python
- PHP
- JavaScript
- SQL
- MATLAB
- LabVIEW

Each of these programming languages has good and bad qualities and is preferable in different situations. Some language is good to use when you need to communicate with a database, while

others are good to use when you want to develop web applications, etc. So, in most situations you probably need to know and use more than one programming language.

Some programming languages are interpreted ("interpreted language"), while others are compiled ("compiled language"). Compiled languages need to be compiled and transform to "machine code" before you can run the program. Interpreted languages translate the code step-by-step at run-time.

Compiled languages are known in general to be faster than interpreted languages. Compiled languages can also easily be compiled into executable programs that can run on their own, while interpreted languages normally need to be run inside the development environment.

Visual Basic, C, C++ and C# are typically compiled languages, while Python, PHP, MATLAB are typically interpreted languages.

Figure 18-2 shows the Top 20 list from RedMonk (this is just one of many similar lists).

Figure 18-2: Popular Programming Languages – Top 20

Below we will give a very short introduction to some of the most popular programming languages.

18.2.1   C

C is a general-purpose computer programming language developed between 1969 and 1973 by Dennis Ritchie at the Bell Telephone Laboratories for use with the Unix operating system.

C is one of the most popular programming languages of all time and there are very few computer architectures for which a C compiler does not exist.

C has greatly influenced many other popular programming languages, most notably C++, which began as an extension to C. C is a procedural language, i.e., no object-oriented programming. C is a compiled language.

18.2.2   C++

C++ is a compiled, general-purpose object-oriented programming language. It is regarded as an intermediate-level language, as it comprises a combination of both high-level and low-level language features.

It was developed by Bjarne Stroustrup in 1979 as an extension to C. C++ is one of the most popular programming languages and its application domains include systems software (such as Microsoft Windows), application software, device drivers, embedded software, high-performance server and client applications, and entertainment software such as video games.

Several groups provide both free and proprietary C++ compiler software.

18.2.3   C#

C# is pronounced "see sharp". C# is an object-oriented programming language and part of the .NET family from Microsoft. C# is intended to be a simple, modern, general-purpose, object-oriented programming language. Its development team is led by Anders Hejlsberg.

C# is very like C++ and Java. C# is developed by Microsoft and works only on the Windows platform. C# is based on the .NET Framework (pronounced "dot net").

.NET is a software framework that runs primarily on Microsoft Windows. The .NET Framework 1.0 and C# 1.0 was released in 2002 as part of Visual Studio .NET 2002.

Visual Studio is the Integrated Development Environment (IDE) you use when programming in C# and the .NET platform.

"Hello World" C# Example:

```csharp
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
```

```
        InitializeComponent();
    }


    private void Form1_Load(object sender, EventArgs e)
    {

        textBox1.Text = "Hello World";


    }

}
}
```

For an introduction to basic C#, please see [18].

For more information about C#:

https://www.halvorsen.blog/documents/programming/csharp/

18.2.4   Java

Java is a programming language originally developed by James Gosling at Sun Microsystems (now owned by Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities.
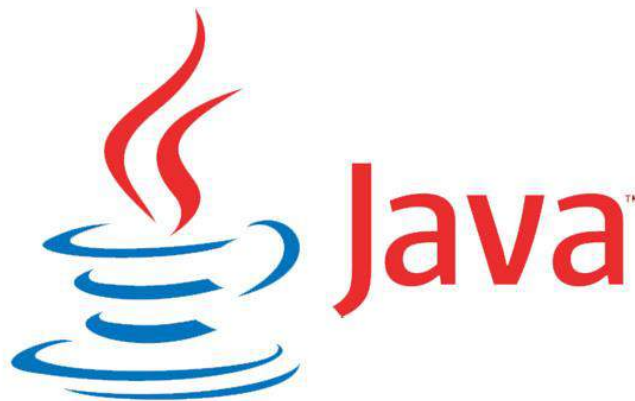
Java is currently one of the most popular programming languages in use and is widely used from application software to web applications. Java applications are typically compiled, and it runs on any Java Virtual Machine (JVM) regardless of the computer architecture. Java is a general-purpose object-oriented. It is intended to let application developers "write once, run anywhere".

A common IDE for programming with Java is the Eclipse IDE. Java and Eclipse are used to create Android Apps.

18.2.5   Objective-C/Swift

Objective-C is a general-purpose, high-level, object-oriented programming language that is based on the C programming language.

It is the main programming language used by Apple for the OS X and iOS and their respective APIs, Cocoa, and Cocoa Touch.

Originally developed in the early 1980s, it was selected as the main language used by NeXT for its NeXTSTEP operating system, from which OS X and iOS are derived.

Swift is the new programming language created by Apple (now open source) that is intended to take over for Objective-C. Swift is now the preferred language when creating Apps for the iOS platform and macOS.

18.2.6   Visual Basic

Visual Basic (VB) is the third-generation event-driven programming language and integrated development environment (IDE) from Microsoft. The first version appeared in 1991. Visual Basic is relatively easy to learn and use and it is a good language for beginners.

Visual Basic was derived from BASIC and enabled the rapid application development (RAD) of graphical user interface (GUI) applications, access to databases using Data Access Objects, Remote Data Objects, or ActiveX Data Objects, and creation of ActiveX controls and objects.

Scripting languages such as VBA (Visual Basic for Applications) and VBScript are syntactically like Visual Basic but perform differently.

The final release was version 6 in 1998 (VB6). The successor is Visual Basic .NET (now known simply as Visual Basic). The .NET Framework 1.0 and Visual Basic .NET 1.0 (VB.NET or just Visual Basic) were released in 2002 as part of Visual Studio .NET 2002.

18.2.7   Perl

Perl is a high-level, general-purpose, interpreted, dynamic programming language. Perl was originally developed by Larry Wall in 1987 as a general-purpose Unix scripting language to make report processing easier. Since then, it has undergone many changes and revisions and become widely popular amongst programmers.
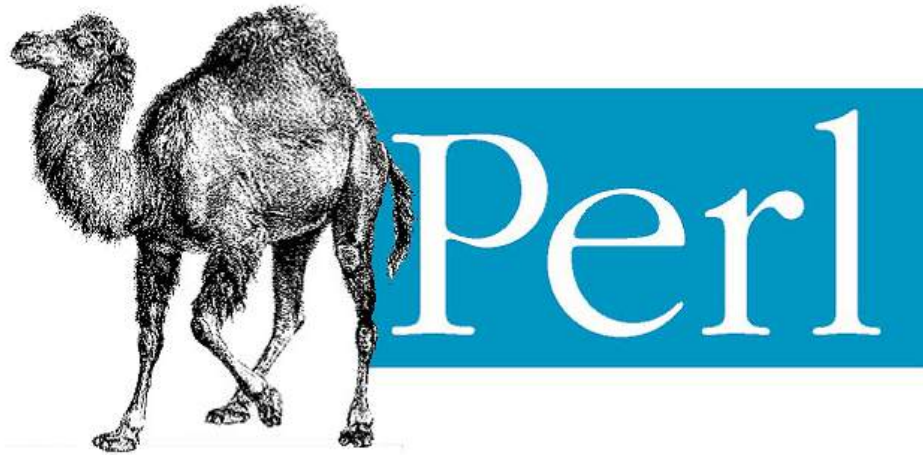
18.2.8   Python

Python is an interpreted high-level programming language and object-oriented programming, and structured programming are fully supported. The reference implementation of Python (CPython) is free and open-source software and has a community-based development model. In addition, Python has alternative implementations.

Python logo:

 Introduction to Python: https://youtu.be/eC0ZgSl4h3k

 Basic Python Programming: https://youtu.be/Wr8Ku5yYeTM

Python interpreters are available for many operating systems, and Python programs can be packaged into stand-alone executable code for many systems using various tools. Guido van Rossum is the creator of Python (1989).

Python example:

```
>>> x = int(input("Please enter an integer: "))
Please enter an integer: 42
>>> if x < 0:
...      x = 0
```

```
...     print('Negative changed to zero')
... elif x == 0:
...     print('Zero')
... elif x == 1:
...     print('Single')
... else:
...     print('More')
...
More
```

For more information about Python:

https://www.halvorsen.blog/documents/programming/python/

18.2.9   PHP

PHP is a general-purpose scripting language originally designed for web development to produce dynamic web pages (server-side scripting). For this purpose, PHP code is embedded into the HTML source document and interpreted by a web server which generates the web page document.

PHP can be deployed on most web servers and as a standalone interpreter, on almost every operating system and platform free of charge.

PHP was originally created by Rasmus Lerdorf in 1995. PHP is installed on more than 20 million websites and 1 million web servers including Facebook. PHP originally stood for "Personal Home Page", while it is now said to stand for "PHP: Hypertext Preprocessor". PHP is free open-source software.

For more information about PHP:

https://www.halvorsen.blog/documents/programming/web/php.php

18.2.10 JavaScript

JavaScript is an object-oriented scripting language that is dynamic, weakly typed. JavaScript is primarily used in the form of client-side JavaScript, implemented as part of a web browser to provide enhanced user interfaces and dynamic websites.

As in most scripting languages, types are associated with values, not with variables. For example, a variable x could be bound to a number, then later rebound to a string. JavaScript uses syntax influenced by that of C.

JavaScript copies many names and naming conventions from Java, but the two languages are otherwise unrelated and have very different semantics. JavaScript was first shipped in 1995.

JavaScript very quickly gained widespread success as a client-side scripting language for web pages. JavaScript is officially managed by Mozilla Foundation.

18.2.11   SQL

SQL, often referred to as Structured Query Language, is a database computer language designed for managing data in relational database management systems (RDBMS).

SQL has become the most widely used database language today. All popular Database Systems support SQL, such as Oracle, SQL Server, etc. SQL was developed at IBM by Donald D. Chamberlin and Raymond F. Boyce in the early 1970s.

 SQL Server and Structured Query Language (SQL): https://youtu.be/sI6skicZse0

SQL Support all kind of CRUD (Create, Read, Update, Delete) operations on database systems.

Example of SQL Syntax:

```
insert into STUDENT (Name , Number, SchoolId)
values ('John Smith', '100005', 1)

select SchoolId, Name from SCHOOL

select * from SCHOOL where SchoolId > 100

update STUDENT set Name='John Wayne' where StudentId=2

delete from STUDENT where SchoolId=3
```

For more information about SQL, please see [19] and Chapter 25.

You may also see the following resource:

https://www.halvorsen.blog/documents/technology/database/

18.2.12   MATLAB

MATLAB (matrix laboratory) is a numerical computing environment and fourth-generation programming language. Developed by MathWorks, MATLAB allows matrix manipulation, plotting of functions and data, and implementation of algorithms.

MATLAB is intended primarily for numerical computing. MATLAB is widely used in academic and research institutions, but also in industry.

MATLAB is an interpreted language. MATLAB is written in C and Java. MATLAB is a weakly dynamically typed programming language.

Getting Started with MATLAB: https://youtu.be/u0bK3AlC9k0

In Figure 18-3 we see the MATLAB IDE.
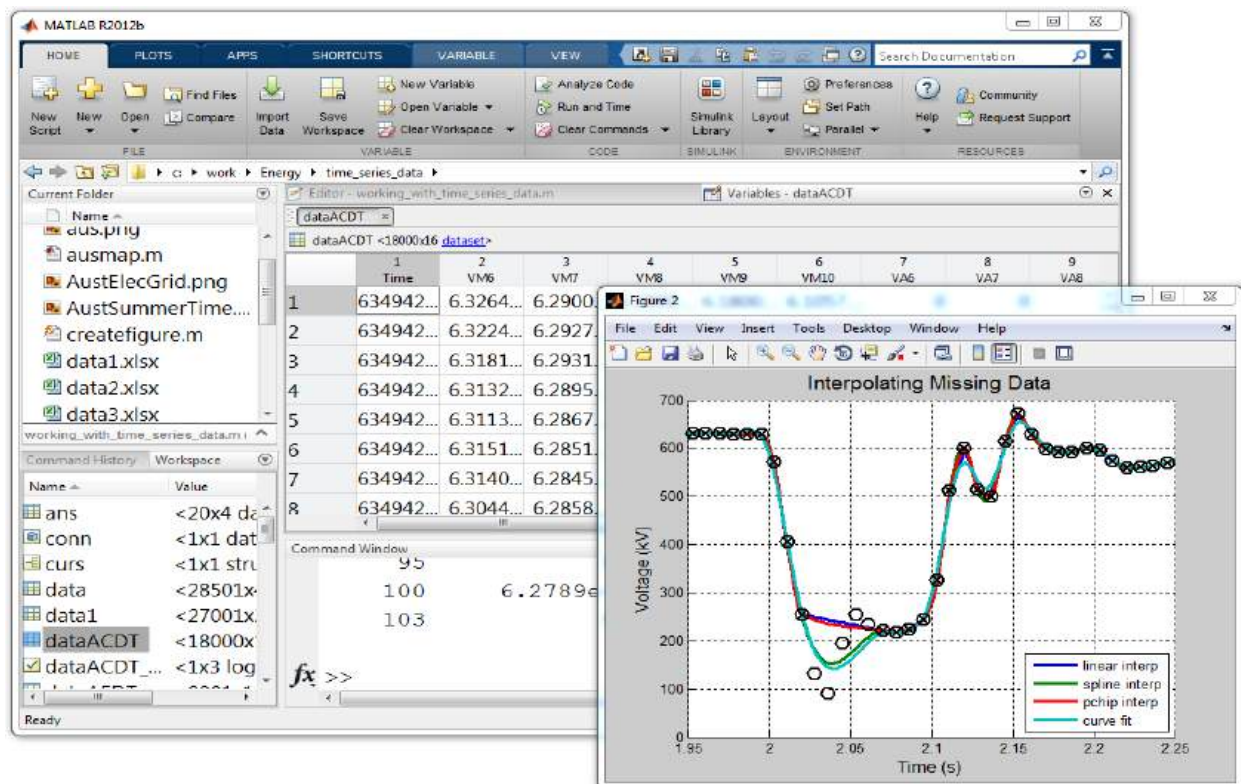
Figure 18-3: MATLAB

For more information about MATLAB, please see [20].

MATLAB Training:

https://www.halvorsen.blog/documents/programming/matlab/

18.2.13 LabVIEW

LabVIEW is a graphical programming language. LabVIEW (short for Laboratory Virtual Instrumentation Engineering Workbench) is a platform and development environment for a visual programming language from National Instruments.

LabVIEW Fundamentals: https://www.youtube.com/watch?v=EOfWjUgolmA&list=PLdb-TcK6Aqj0NeLk7K66_mvc-HNKS1-PJ

LabVIEW was originally released for the Apple Macintosh in 1986, and it is commonly used for data acquisition, instrument control, and industrial automation on a variety of platforms including Microsoft Windows, various versions of UNIX, Linux, and Mac OS X.

In LabVIEW, you can create and run executable files. To do so you need to have the LabVIEW Run-time Engine installed on the target computer. LabVIEW can be extended with additional modules and Toolkits. LabVIEW MathScript is an add-on which is a miniature version of MATLAB.

In Figure 18-4 we see a typical LabVIEW Program.
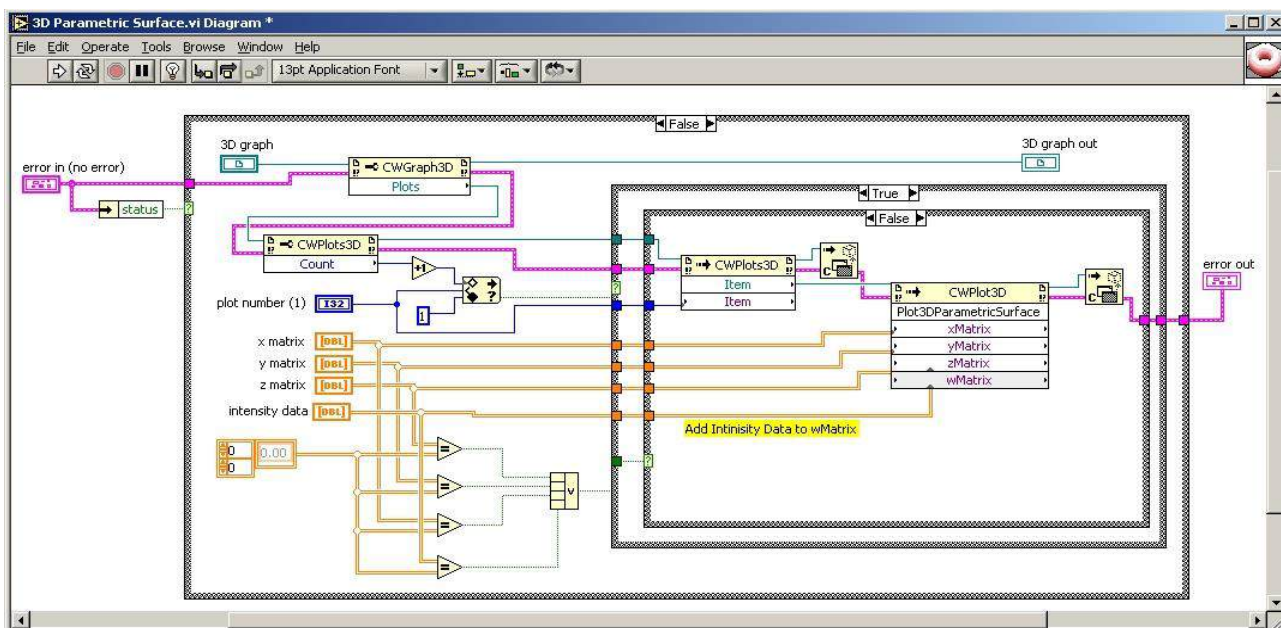
Figure 18-4: LabVIEW Program

For more information about LabVIEW, please see [21].

LabVIEW Training:

https://www.halvorsen.blog/documents/programming/labview/

18.3   Naming Convention

There are different name conventions for how to specify your variables, classes and Methods, etc.

Camel notation:

For **variables** and **parameters/arguments** we normally use "Camel notation".

Examples:

```
string myCar;
int number;
string backColor;
```

→ In Camel casing the first letter of an identifier is lowercase, and the first letter of each subsequent concatenated word is capitalized.

Pascal notation:

For **classes**, **methods** and **properties**, we normally use "Pascal notation".

Examples:

```
class Car
{
    void ShowCarColor()
    {
       …
    }
}
```

→ In Pascal casing the first letter in the identifier and the first letter of each subsequent concatenated word are capitalized.

For Namespaces we use Pascal casing and a dot separator.

Examples:

```
System.Drawing
System.Collections.Generics
```

Controls:

For controls on your user interface, we either use "Pascal notation" or "Hungarian notation", but stick to one of them!

Examples:

"Pascal notation":

```
LoginName
LoginPassword
```

"Hungarian notation":

```
txtName
txtPassword
lblName
btnCancel
```

Where "txt" means that it is a Text Control, "lbl" a Label Control, "btn" a Button Control, etc.

Acronyms:

The casing of acronyms depends on the length of the acronym. All acronyms are at least two characters long. If an acronym is exactly two characters, it is considered a short acronym. An acronym of three or more characters is a long acronym.

In general, you should not use abbreviations or acronyms. These make your names less readable. Similarly, it is difficult to know when it is safe to assume that an acronym is widely recognized.

But if you must, the rules are as follows:

Short acronym Examples (two characters):

```
DBRate
```

A property named DBRate is an example of a short acronym (DB) used as the first word of a Pascal-cased identifier.

```
ioChannel
```

A parameter named ioChannel is an example of a short acronym (IO) used as the first word of a camel-cased identifier.

Long acronym Examples (three or more characters):

```
XmlWriter
```

A class named XmlWriter is an example of a long acronym used as the first word of a Pascal-cased identifier.

```
htmlReader
```

A parameter named htmlReader is an example of a long acronym used as the first word of a camel-cased identifier.

18.4   Defensive Programming

In programming error and exception handling is very important. C# has built-in and ready to use mechanism to handle this. This mechanism is based on the keywords *try*, *catch*, *throw* and *finally*.

Exceptions are unforeseen errors that happen in your programs. Most of the time, you can, and should, detect and handle program errors in your code. For example, validating user input, checking for null objects, and verifying the values returned from methods are what you expect, are all examples of good standard error handling that you should be doing all the time.

However, there are times when you don't know if an error will occur. For example, you can't predict when you'll receive a file I/O error, run out of system memory, or encounter a database error. These things are generally unlikely, but they could still happen, and you want to be able to deal with them when they do occur. This is where exception handling comes in.

18.4.1   Error Handling

Error handling is an important part of the coding process, making the applications robust when some unexpected things happen.

Exception Handling:

When exceptions occur, they are said to be "thrown". C# uses the keywords *try*, *catch*, *throw* and *finally*. It works like this: A method will try to execute a piece of code. If the code detects a problem, it will *throw* an error indication, which your code can *catch*, and no matter what happens, it *finally* executes a special code block at the end.

The syntax in C# is as follows:

```
MyMethod()
{

    try
    {
      ... //Do Something that can cause an Exception
    }
     catch
    {
      ... //Handle Exceptions
    }

     finally
    {
      ... //Clean Up
    }

}
```

Example:

```
public void WriteDaqData(double analogDataOut)
    {


        Task analogOutTask = new Task();

        AOChannel myAOChannel;
```

```
        try
        {

            myAOChannel = analogOutTask.AOChannels.CreateVoltageChannel(
                aoChannel,
                "myAOChannel",
                0,
                5,
                AOVoltageUnits.Volts
                );

            AnalogSingleChannelWriter writer = new
              AnalogSingleChannelWriter(analogOutTask.Stream);

            writer.WriteSingleSample(true, analogDataOut);
        }
        catch (Exception e)
        {
            string errorMessage;

            errorMessage = e.Message.ToString();

        }

        finally
        {
            analogOutTask.Stop();
        }

    }
```

18.5   Software Frameworks

Some popular software frameworks are:

- .NET Framework
- ASP.NET

They will be discussed in more detail below.

18.5.1   .NET Framework

The .NET Framework (pronounced "dot net") is a software framework that runs primarily on Microsoft Windows. It includes a large library and supports several programming languages which allow language interoperability (each language can use code written in other languages). The .NET library is available to all the programming languages that .NET supports. Programs written for the .NET Framework execute in a software environment, known as the Common Language Runtime (CLR), an application virtual machine that provides important services such as security, memory management, and exception handling. The class library and the CLR together constitute the .NET Framework.

18.5.2   ASP.NET

ASP.NET is an open-source web framework, created by Microsoft, for building web apps and services using the .NET Framework or the .NET Core. We have both ASP.NET and ASP.NET Core. ASP.NET Core is the new approach built on .NET Core.

Some relevant videos:

ASP.NET Core - Hello World: https://youtu.be/lcQsWYgQXK4

ASP.NET Core – Introduction: https://youtu.be/zkOtiBcwo8s

For more information about ASP.NET:

https://www.halvorsen.blog/documents/programming/web/aspnet

19 Software Architecture

When creating software, we use different architecture depending on the platform and the purpose with the software.

In this document, we will focus on client-server, 3-tier architecture and creating and using Web Services and APIs.
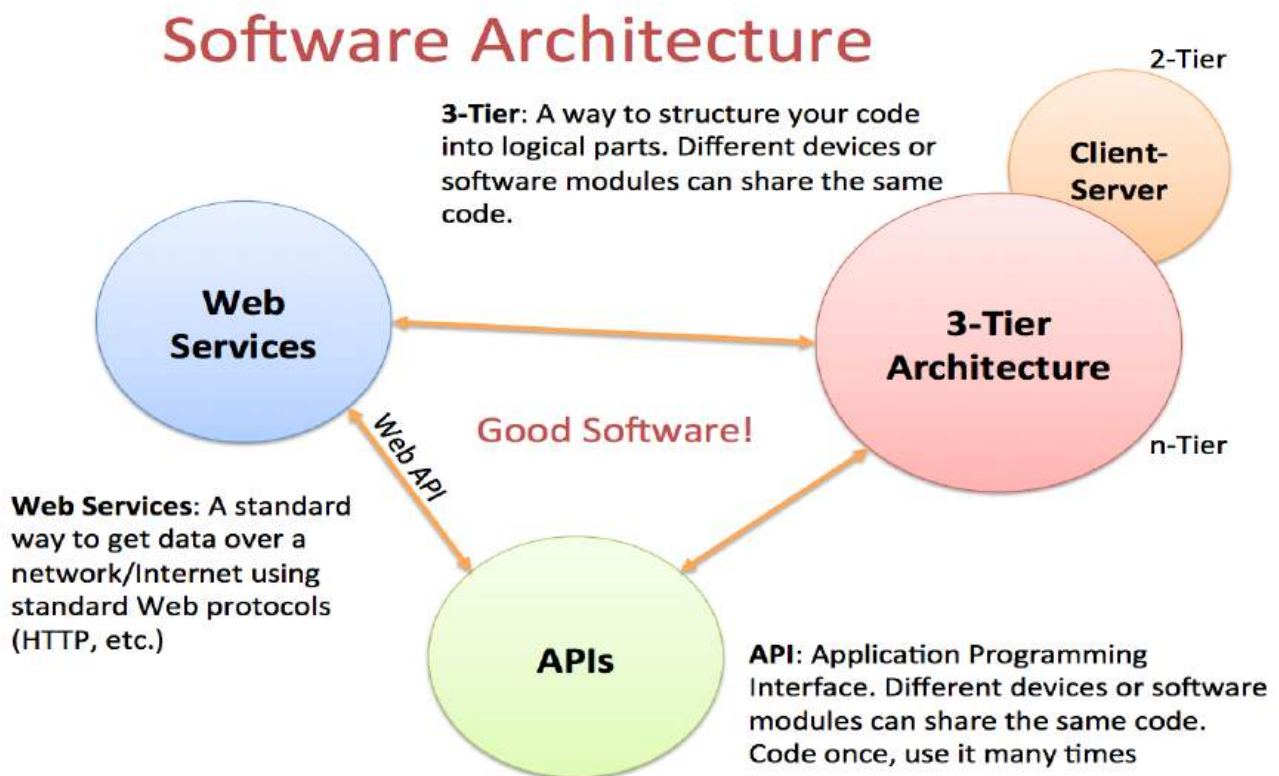
Figure 19-1: Software Architecture

In Figure 19-2 we see how a typical software application is interacting with the surrounding environments, such as the users of the software and the underlying operating system (which is also software) and hardware.
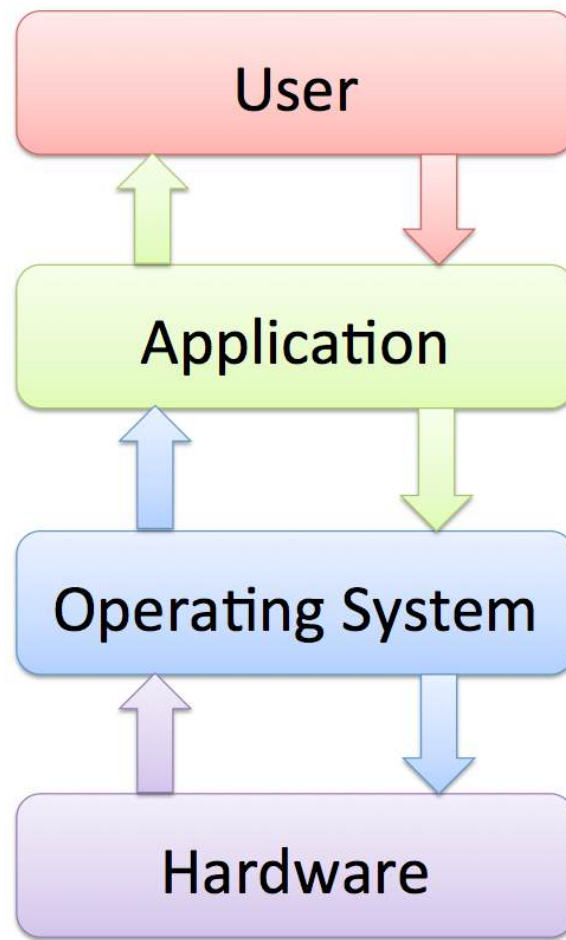
Figure 19-2. Software Interaction with the Environment

Based on the fundamental interaction between the software and the environment we have different kinds of software architecture.

Software Architecture Examples:

- Client – Server
- n-tier architecture, 3-tier architecture
- Model-View-Controller (MVC)
- Web Services
- Interfaces
- APIs
- Service-oriented architecture (SOA)
- Microservices

In Figure 19-3 we see some examples of different network and software architecture typically used in software development.
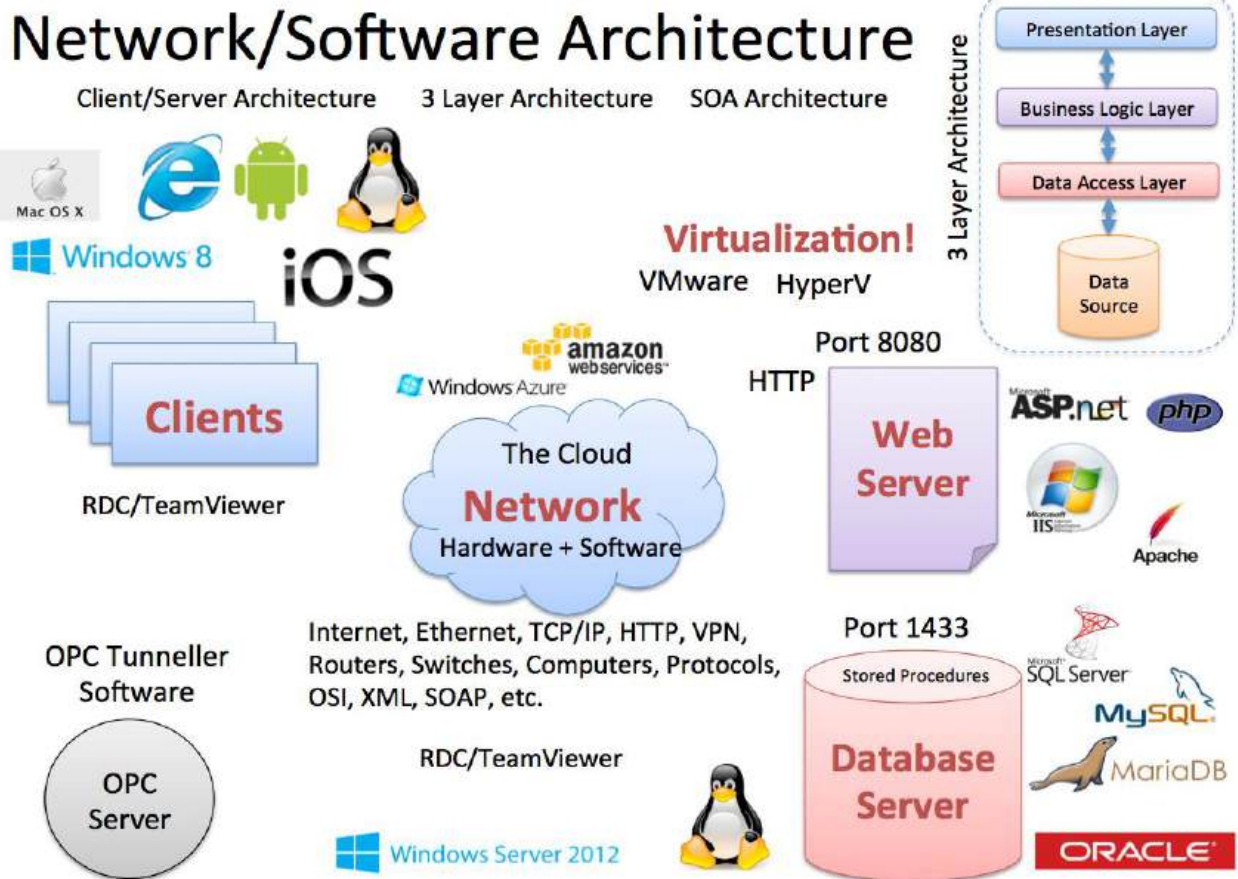
Figure 19-3: Network/Software Architecture Examples

Make sure to document your system and your architecture in your documentation. You should typically include several drawings and sketches of your architecture.

Make System Sketch in PowerPoint: https://youtu.be/9mmBXFOjV3s

19.1   API

API - Application Programming Interface. An API is a specification of how some software components should interact with each other. Typically, it is a library with functions, etc. you can use in your code.

Examples:

- Windows API
- Java API

But you can also create your own API that you use internally in the team or expose to others.

Creating APIs is good practice and makes it easy to reuse your code in other components or applications. If all the developers in the team create the same code without thinking of reusing code from others or creating code in such manners it can't be used by others, the software project is doomed to fail.

Software Design without APIs [22]:

Here are some pros and cons regarding APIs:

Pros:

- Fast to implement small projects.

- Agile – can serve as a starting point for API design.

- No need to consider how code interfaces with other software.

- Can be appropriate for small "dead end" projects.

Cons:

- Inappropriate for large projects.

- Code has a limited (as opposed to general) functionality.

- Code is not reusable.

- Code is hard to maintain/modify.

- Prone to errors and bugs.

Why a Good API is hard to Design?

Here are some examples:

- Forces designer to anticipate future usage of code.

- Requirements are incomplete (may never be complete).

- Requires abstraction.

- Requires modularization.

- Requires skills in programming languages.

- Requires code rewrites – time consuming and labor intensive.

[http://geek-and-poke.com]
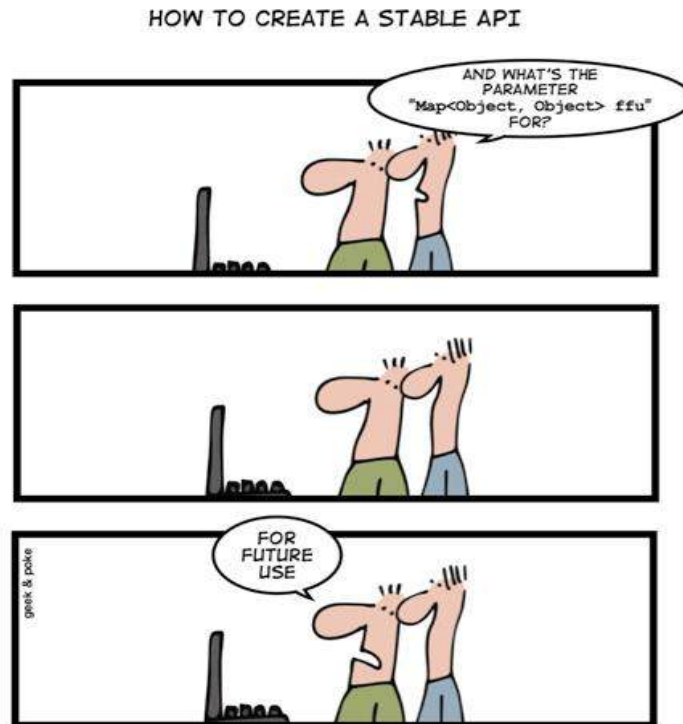
When an API is used in a project, it

- Allows us to focus on the project.

- Saves development time.

- Reduces errors and debugging.

- Facilitates modular design.

- Provides a consistent development platform.

API driven design requires planning and programming skills. API driven design is costly initially, but it pays in the long run. So, obviously, creating APIs is good software practice in most cases.

It is impossible to imagine how anyone would design a car today without taking advantage of existing modules or vehicle subsystems – it is the same with software!

19.2   Client-Server

Client/server describes the relationship between two computer programs in which one program, the client, makes a service request from another program, the server, which fulfills the request. Although the client/server idea can be used by programs within a single computer, it is a more

important idea in a network. In a network, the client/server model provides a convenient way to interconnect programs that are distributed efficiently across different locations.

19.3   Web Services

The "problem": How do we share data between different devices in a network (see Figure 19-4)?
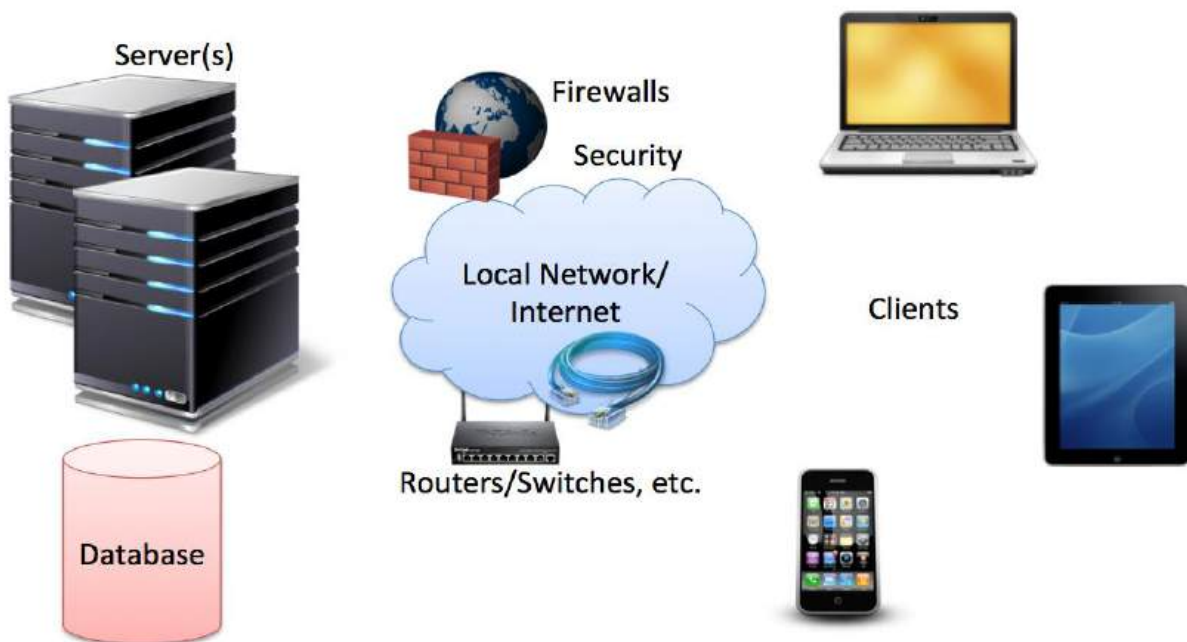
Figure 19-4: Data sharing between devices in a network

Direct connection between the database and the clients that need the data is normally not possible, due to security, compatibility issues, etc. (firewalls, hacker attacks, etc.). Direct connection in a local network (behind the firewall) is normally OK – but not over the Internet (see Figure 19-5).
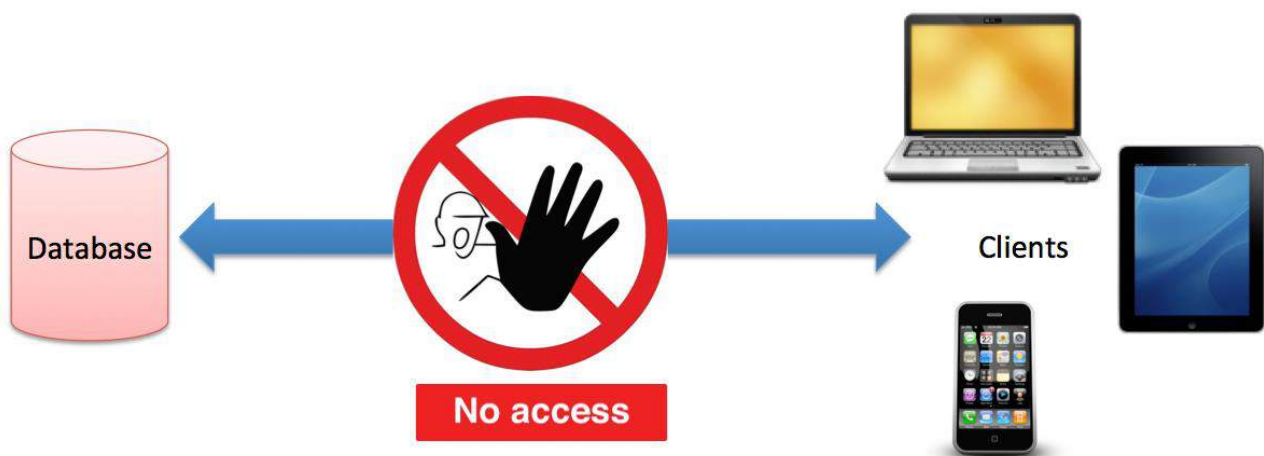
Figure 19-5: Limited access to the database in a network

The solution: Web Services. Web Services use standard web protocols like HTTP, etc. HTTP is supported by all Web Browser, Servers, and many Programming Languages.

Today Web Services have been very popular. A Web service is a method of communication between two devices over the World Wide Web and makes it easy to share data over a network or the internet.

A Web Service is:

- A Web API
- A Standard defined by W3C
- Cross-platform and Platform-independent Communication
- Distributed Application Development

Web Services can be implemented and used in most Programming Languages (C#/ASP.NET, PHP, LabVIEW, Objective-C, Java, etc.)

Web Services uses standard Web technology (Web protocols) such as HTTP, REST, SOAP, XML, WSDL, JSON, etc.
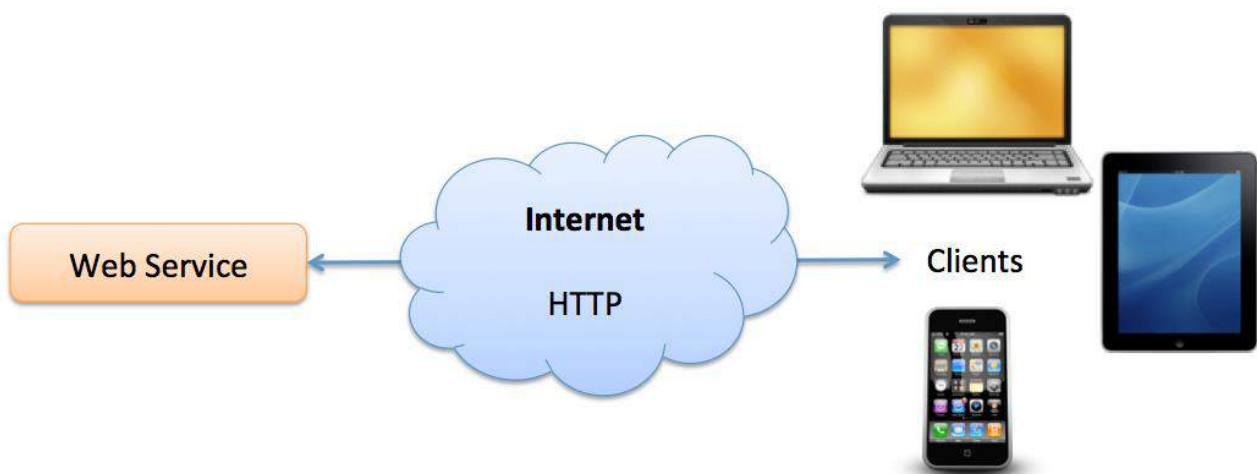
Figure 19-6: Web Services

Web Services technology used in Web Services:

- HTTP - Hypertext Transfer Protocol

- XML – Extensible Markup Language

- WSDL - Web Services Description Language

- SOAP - Simple Object Access Protocol

- REST - Representational State Transfer

(We will not go into details)

A Web Service is typically deployed on a web server, similar as ordinary web pages, see Figure 19-7.
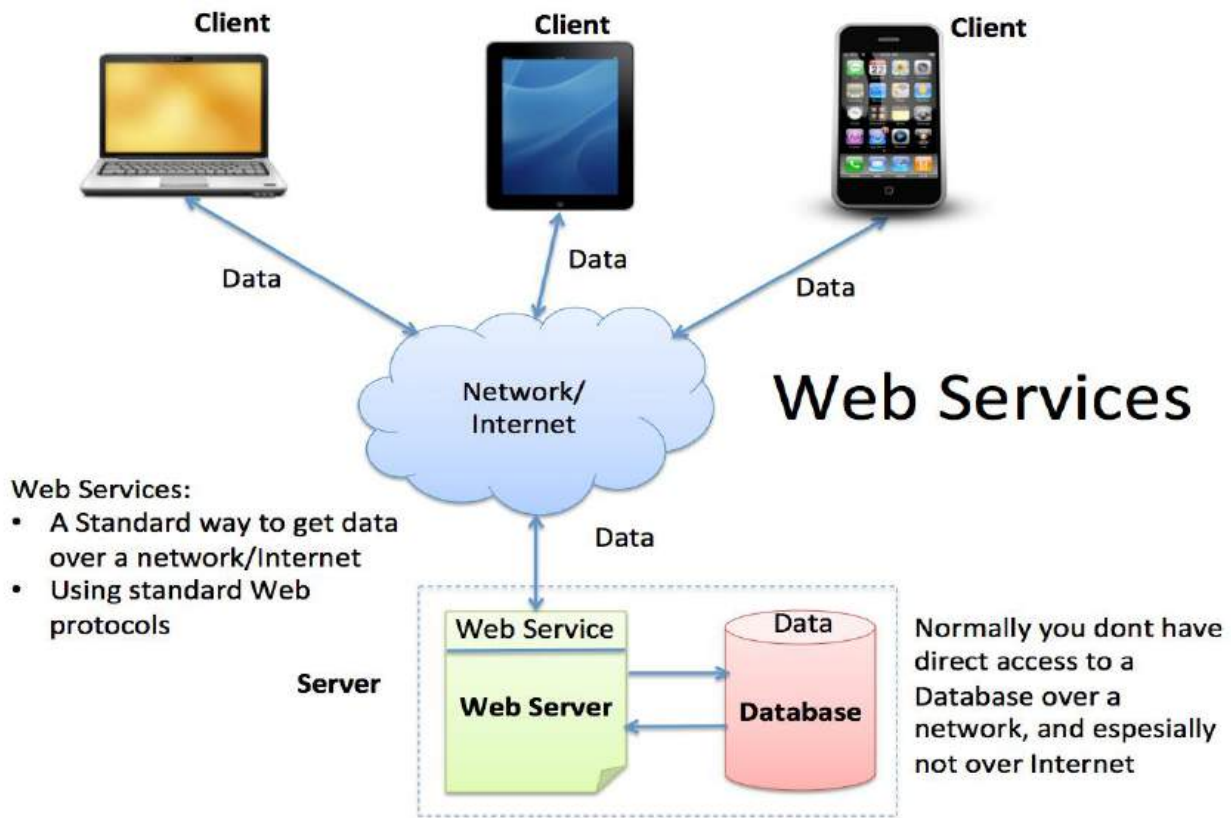
Figure 19-7: Web Service Infrastructure

We have 2 different types of Web Services:

- Web Services 1.0: SOAP Web Services

   "Complex"

- Web Services 2.0: REST Web Services

  - Lightweight and Flexible

  - A new and simpler version of WS

  - All major WS on the Internet today use REST

In Figure 19-8 we summarize Web Services 1.0 vs. 2.0.

Figure 19-8: Different kind of Web Services

This document only describes the basic principles of Web Services. For practical code examples, see the Tutorial "Introduction to Web Services" [23] and other relevant resources.

19.3.1   SOAP Web Services

In Figure 19-9 we see the different "layers" a "SOAP Web Service" consists of.
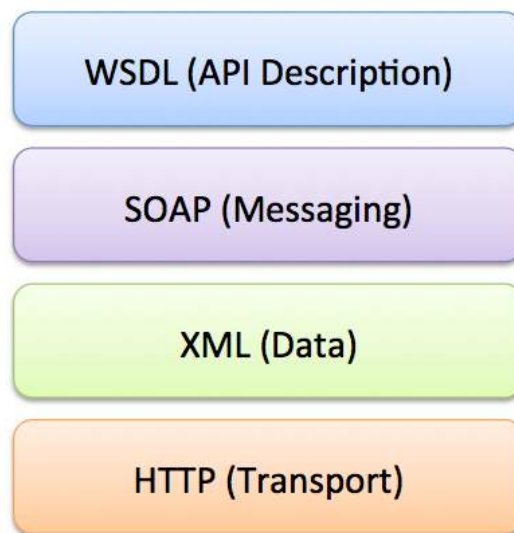
Figure 19-9: SOAP Web Services Architecture

XML:

XML stands for eXtensible Markup Language. XML is designed to transport and store data.

Below we see an XML document example.

```
<?xml version="1.0"?>
<note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

19.3.2   REST Web Services

In Figure 19-10 we see the different "layers" a "REST Web Service" consists of.
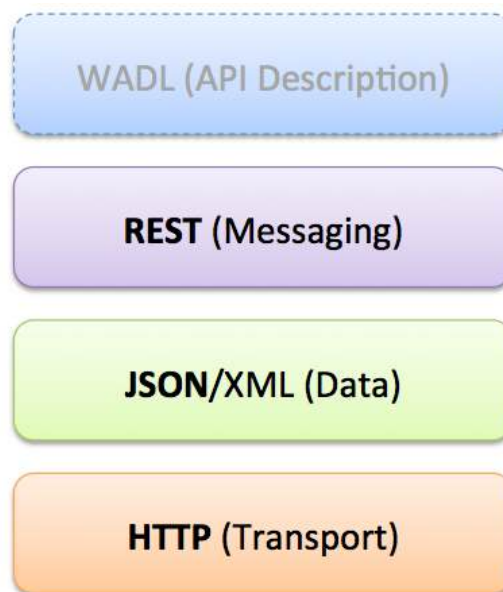
Figure 19-10: REST Web Services Architecture

19.3.3   Creating Web Services with Visual Studio

Visual Studio has powerful features for creating Web Services and Web APIs (Figure 19-11).
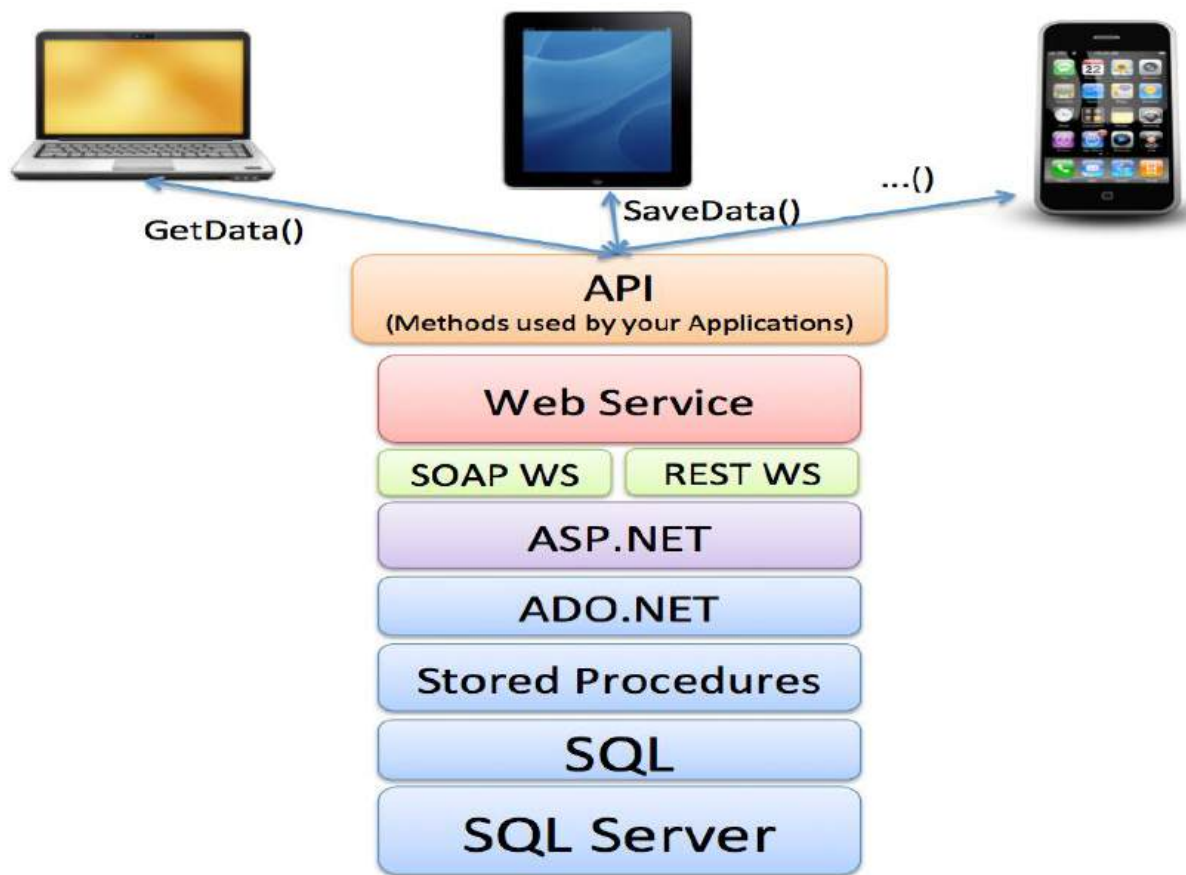
Figure 19-11: Creating Web Services using Visual Studio and ASP.NET

19.4   3-tier Architecture

In general, we have so-called n-tier architecture, but the most common version is a 3-tier architecture.

3-tier architecture is a client-server architecture in which the functional process logic, data access, computer data storage and user interface are developed and maintained as independent modules on separate platforms. Three-tier architecture is a software design pattern and a well-established software architecture. Its three tiers are the presentation tier, application tier and data tier.

The 3 tiers or layers are as follows:

- Presentation Layer
- Business/Application Layer
- Data Access Layer

These layers may be on the same computer, but normally they are distributed on different computers.
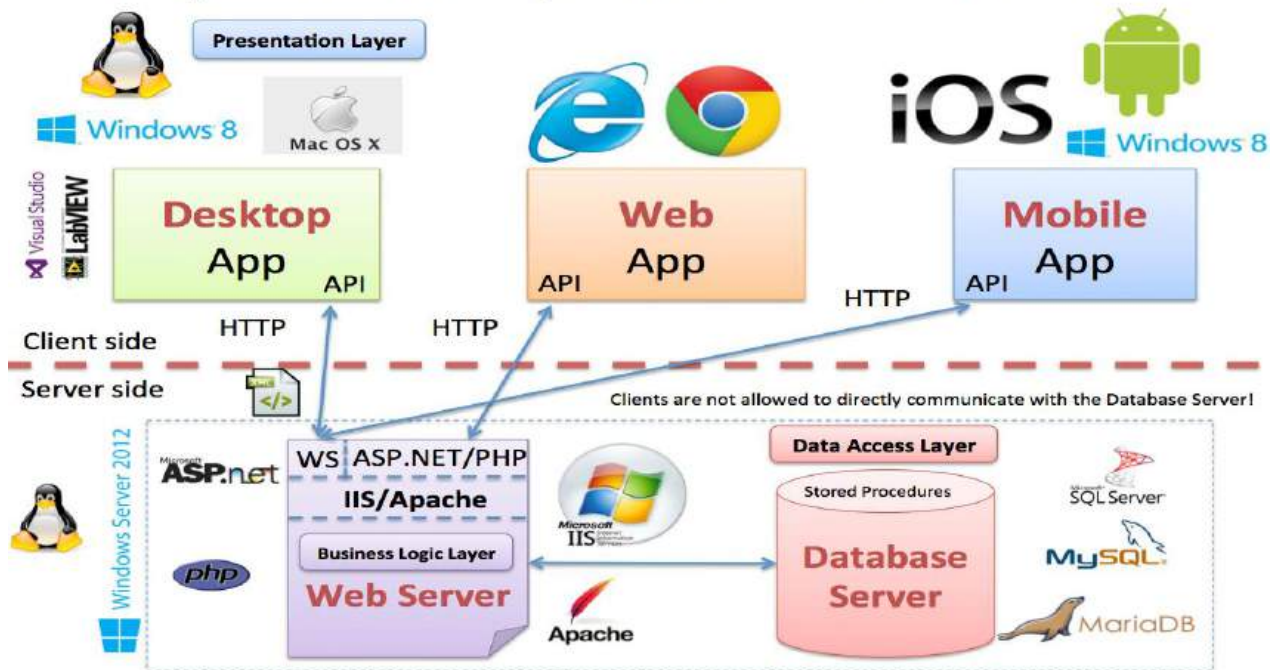
Figure 19-12: Software Architecture

A short description of the different layers:

Presentation Tier:

This is the topmost level of application. The presentation tier displays information related to such services as browsing merchandise, purchasing, and shopping cart contents. It communicates with other tiers by which it puts out the results to the browser/client tier and all other tiers in the network.

In simple terms, it is a layer which users can directly access such as a web page, or an operating systems GUI

Application tier:

(Other term used: Business logic, logic tier, data access tier, or middle tier) The application tier is pulled out from the presentation tier and, as its own layer. It controls an application's functionality by performing detailed processing.

Data tier:

This tier consists of database servers. Here information is stored and retrieved. This tier keeps data neutral and independent from application servers or business logic. Giving data its own tier also improves scalability and performance.

Figure 19-13 shows a sketch of the typical logical layers or tiers in a 3-tier/layer architecture.
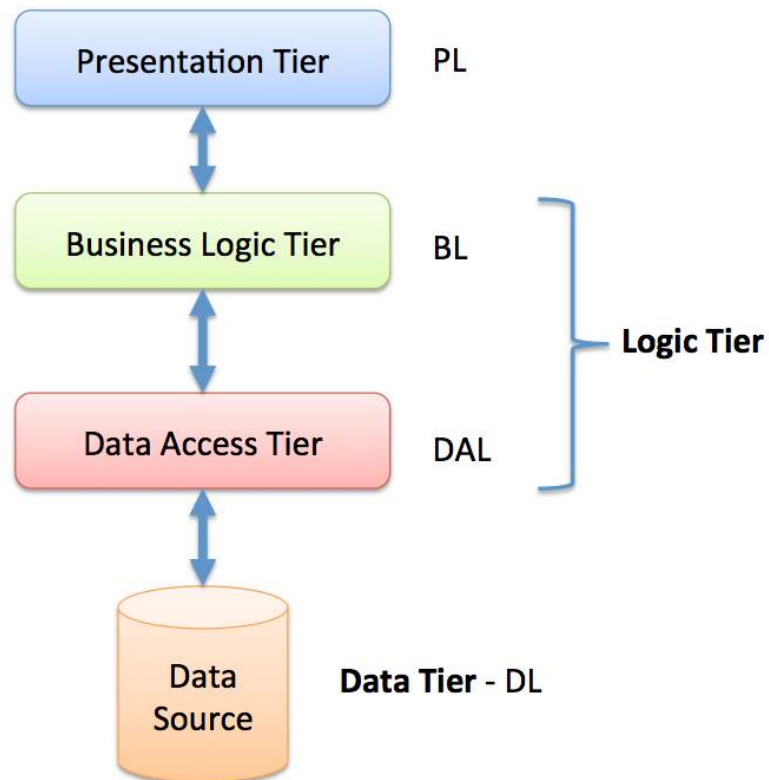
Figure 19-13: 3-tier Architecture

Figure 19-14 gives an overview of the 3-tier architecture and gives a short description of the different layers.

In web development, 3-tier is often common, including the following

- A Web Server
- An Application Server
- A Database

Web Services could be used for communication between the different layers.

Presentation tier

The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

Logic tier

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

Data tier

Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.

Figure 19-14: 3-tier Overview

In Figure 19-15 we see an example of 3-layer architecture software.
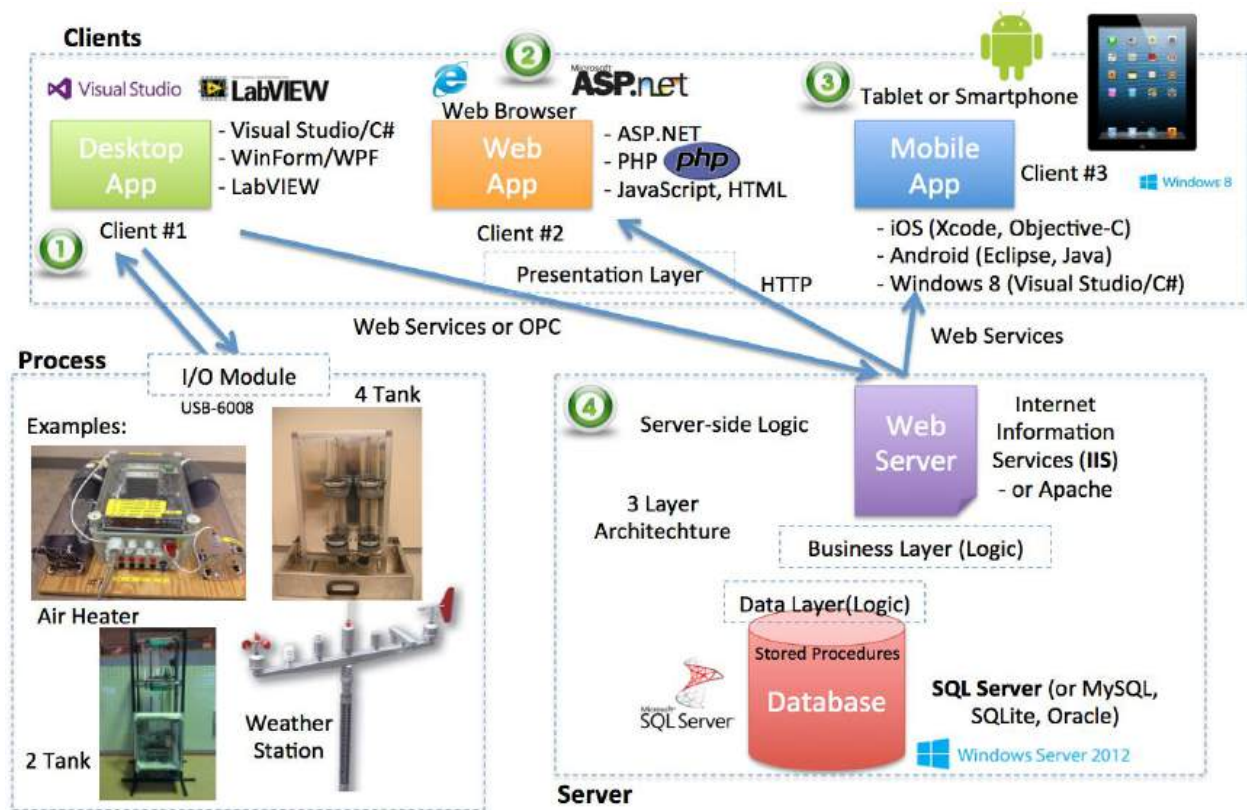
Figure 19-15: Example of 3-layer Architecture Software

Part 4 : Management and Development Tools

In this part, we give an overview of tools used (and needed) in modern software engineering, like collaboration tools, source code control tools, programming platforms, frameworks, languages, etc.

20 Integrated Development Environment (IDE)

What is an IDE? What is the difference between an IDE and a Programming Language?

Popular IDEs:

- Visual Studio
- Visual Studio Code
- Xcode
- Eclipse
- Android Studio
- Spyder
- etc.

Some of these will be discussed below.

20.1 Visual Studio

Microsoft Visual Studio (see Figure 20-1) is an integrated development environment (IDE) from Microsoft.
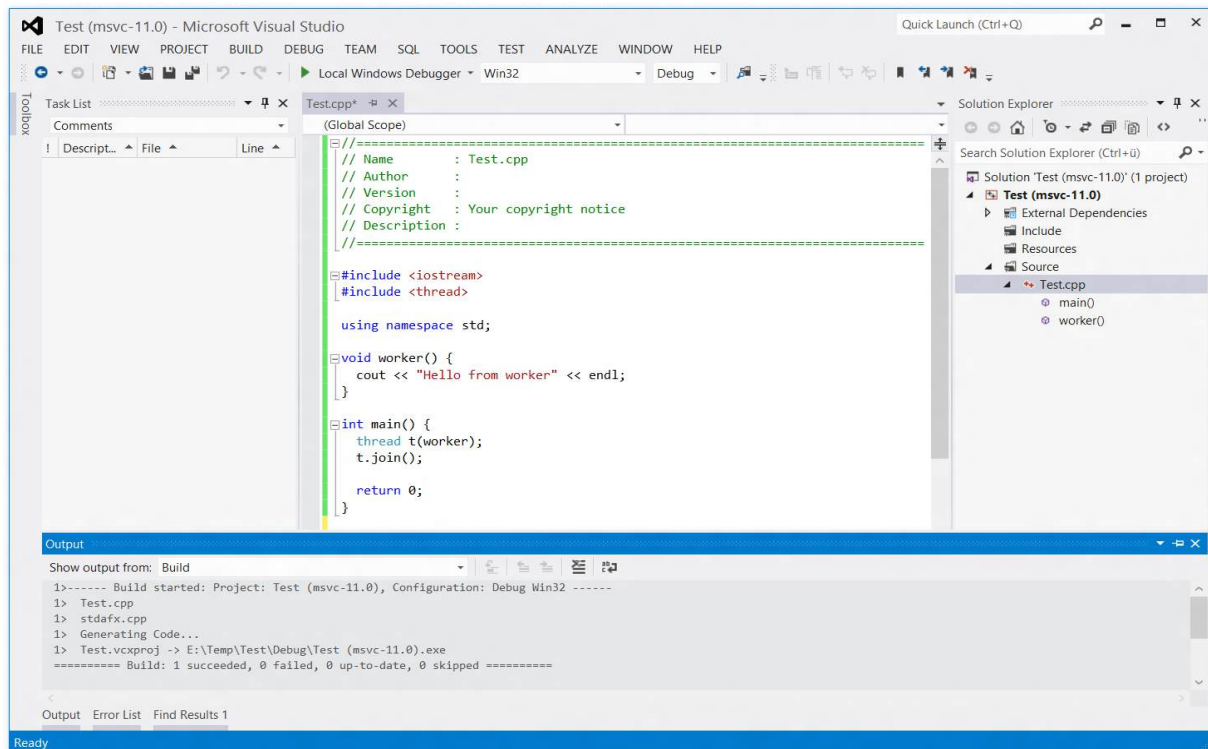
Figure 20-1: Visual Studio

It can be used to develop console and graphical user interface applications along with Windows Forms applications, web sites, web applications, and web services in both native code together with managed code for all platforms supported by Microsoft Windows, Windows Phone, Windows CE, .NET Framework, .NET Compact Framework and Microsoft Silverlight.

A simplified version of Visual Studio has also been released on MacOS.

Visual Studio Code is also an alternative.

For more information about Visual Studio and C#, see [18].

20.2   Visual Studio for Mac

A simplified version of Visual Studio has also been released on MacOS. Many of the features from the Windows edition are not supported, while ASP.NET core is fully supported, which is a new cross-platform version of the ASP.NET framework.

For more information:

https://www.visualstudio.com/vs/visual-studio-mac/

20.3   Visual Studio Code

An open source and cross platform (Windows, MacOS and Linux) and simple and very downscaled version of Visual Studio. It is very easy to use, it has IntelliSense, etc. Everything is done in code; you cannot create your user interface graphically like you can do in Visual Studio. In Visual Studio Code is the code in focus.

For more information:

https://code.visualstudio.com/

20.4   Xcode

Xcode (see Figure 20-2) is the IDE created by Apple for developing software for Mac OS X and the iOS platform (iPhone, iPad).

The programming languages used within the Xcode environment is Swift.

Swift is a new a general-purpose, high-level, object-oriented programming language that is based on the C programming language and Objective-C, etc.

It is the main programming language used by Apple for the OS X and iOS and their respective APIs, Cocoa and Cocoa Touch. Apple released a new programming language, called Swift. Swift has now replaced Objective-C as the official language for iOS and Mac programming.
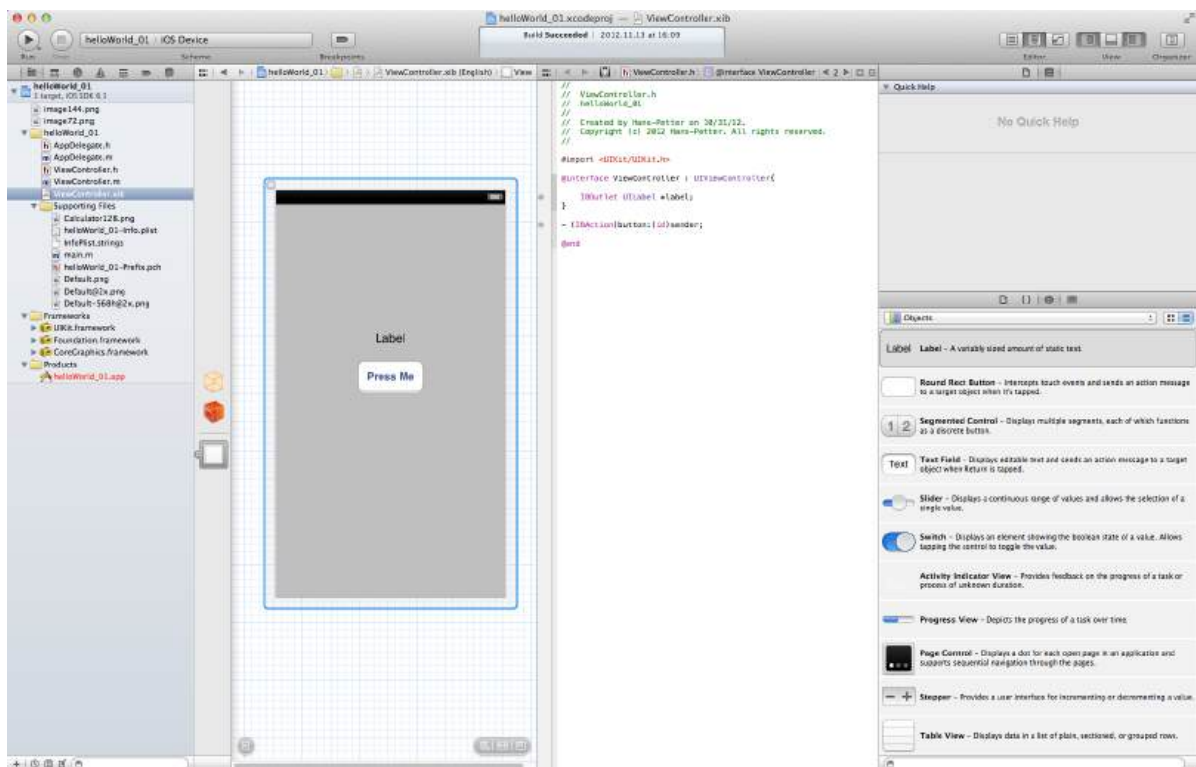
Figure 20-2: Xcode

20.5   Eclipse

Eclipse (see Figure 20-3) is a multi-language Integrated Development Environment (IDE). It is written mostly in Java. It can be used to develop applications in Java, C, C++, JavaScript, Perl, PHP, Python, Ruby, etc.
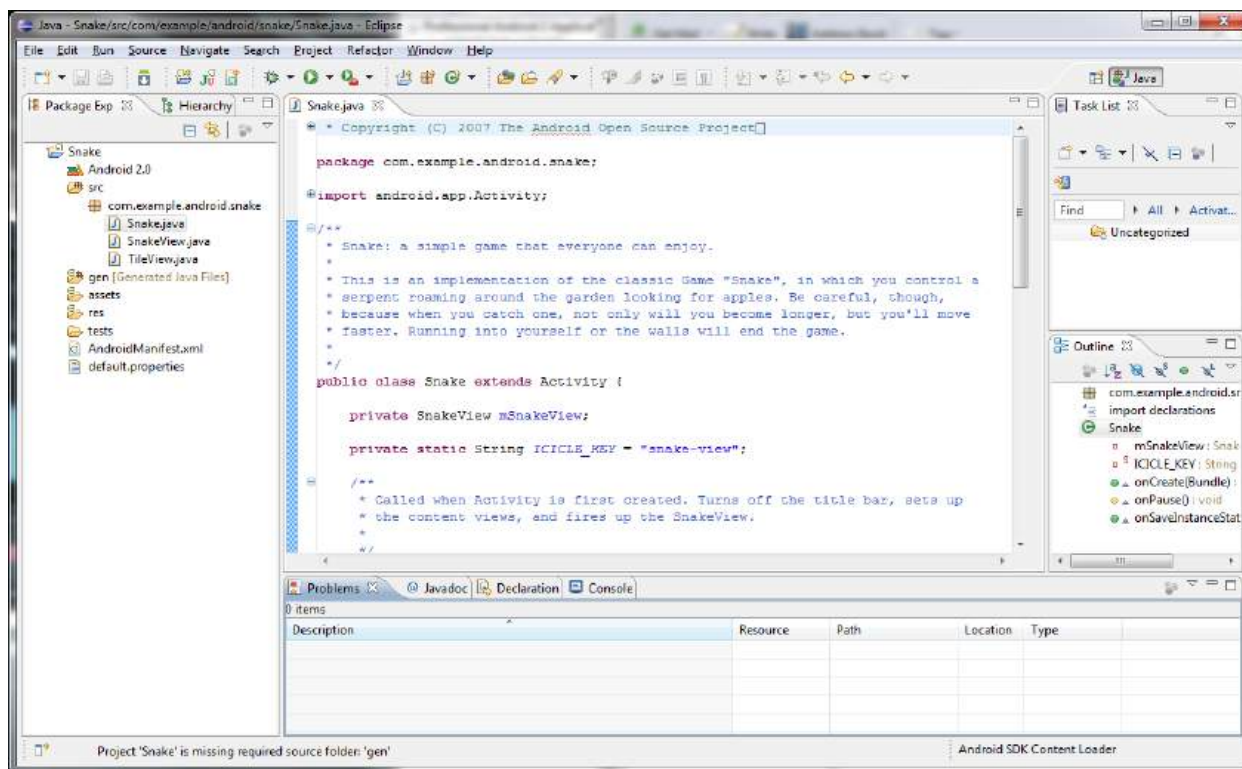
Figure 20-3: Eclipse

Eclipse is typically also used for Android programming. Then you will need the Android SDK plug-in. Eclipse is available on Windows, macOS and Linux.

20.6   Android Studio

Google has released a new IDE for Android development called Android Studio (see Figure 20-4). It was created to make it easier to develop Android Apps.

Android Studio is available on Windows, macOS and Linux.

Figure 20-4: Android Studio

21 UML Software

There exist hundreds of different software for creating UML diagrams, here we mention just a few:

- Enterprise Architect
- StarUML
- Rational Rose
- Diagram tools like Lucidchart, Miro, Draw.io, etc.
- etc.

21.1 StarUML

StarUML is cross-platform, which makes it possible to use it on Windows, Linux and macOS. You can evaluate for free without time limit.

Figure 21-1 show the StarUML software.
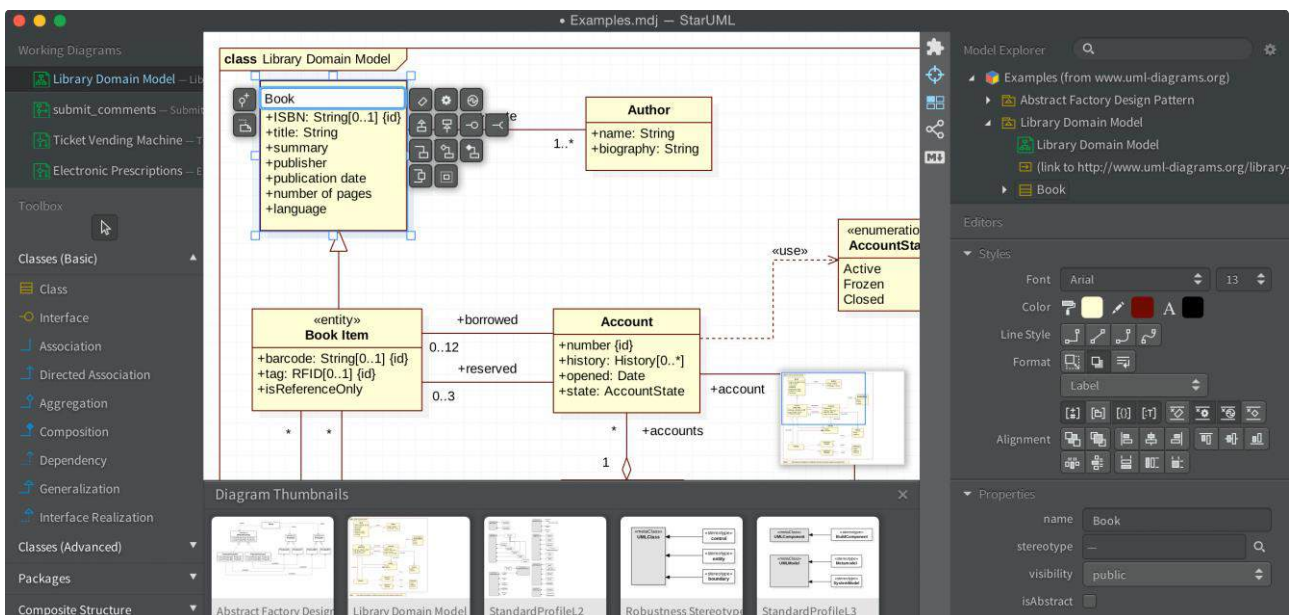
Figure 21-1: StarUML

Web Site: http://staruml.io

Download: http://staruml.io/download

22 Source Code Control (SCC)

22.1 Introduction

What is Source Code Control (SCC) or a version control system? A version control system keeps track of all work and all changes in a set of files. It Allows several developers (potentially widely separated in space and time) to collaborate.

In this chapter, we will give a short overview of some of the most popular source code control (SCC) systems on the market today.

 Azure DevOps with Git and Visual Studio: https://youtu.be/tlFazeYmI_U

Here is a list with some of the most popular SCC systems on the market today:

- Azure DevOps
- CVS
- SVN (Subversion)
- Git
- Mercurial
- Bazaar
- LibreSource
- Monotone
- BitKeeper

The focus will be on Azure DevOps from Microsoft because this software is tightly connected to Visual Studio. In addition, it has lots of other features in addition to SCC.

Typical SCC Features:

- Checkout, Check-in/Commit
- Branching, Merging
- File Locking (avoid concurrent access)
- Label/Tag
- Change/Change List
- Conflict
- Revision, Iteration

We have two main kinds of SCC systems:

- Centralized/client–server architecture
- Distributed Version Control System (DVCS)

In Figure 22-1 we see an overview of which architecture the different SCC systems are using.
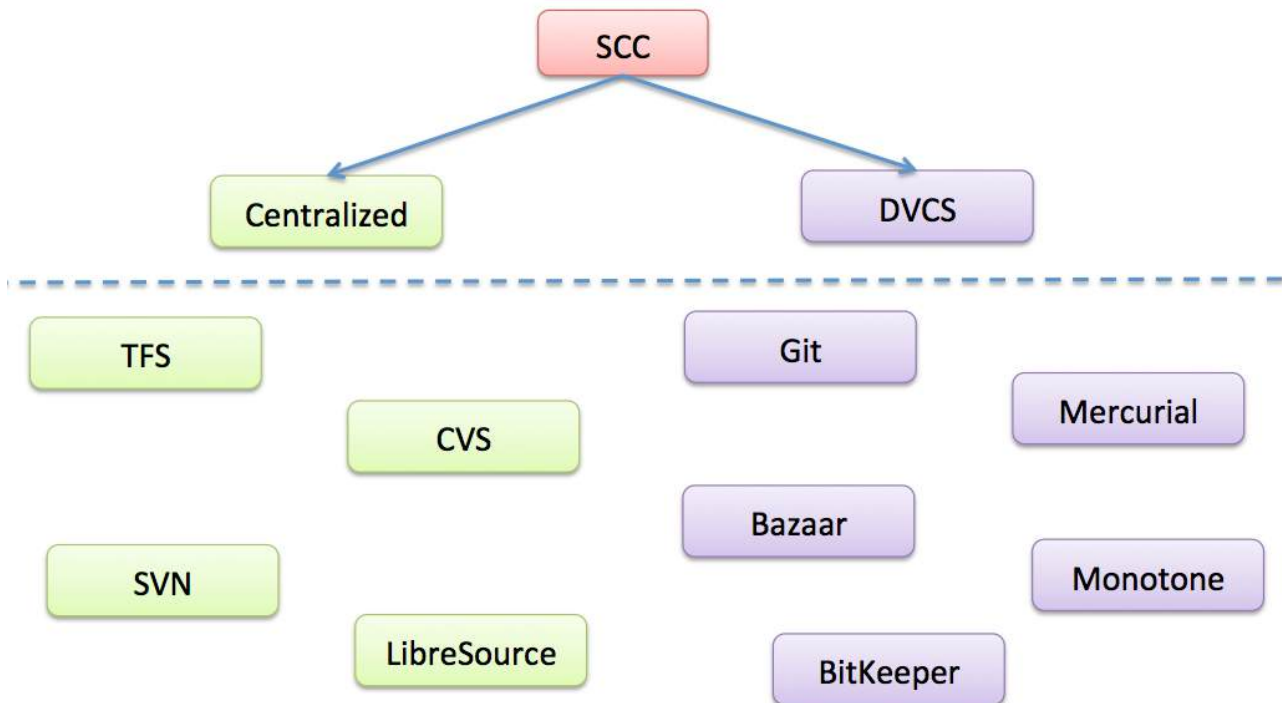
Figure 22-1: SCC Architecture

Centralized/Client–Server architecture:

A server stores the current version(s) of a project and its history, and clients connect to the server to "check out" a complete copy of the project, work on this copy and then later "check in" their changes.

Distributed Version Control System (DVCS):

With a distributed version control system, there isn't one centralized code base to pull the code from. Different branches hold different parts of the code. Git is a DVCS. Other version control systems, such as SVN and CVS, use centralized version control, meaning that only one master copy of the software is used. DVCS systems use a peer-to-peer approach.

In some cases, the SCC system is integrated in a so-called Application Lifecycle Management system.

Application Lifecycle Management (ALM) systems are systems that take care of all aspects in software development from planning, requirements, coding, testing, deployment, and maintenance.

ALM is short for Application Lifecycle Management. An ALM tool typically facilitate and integrate things like:

- Requirements Management
- Architecture
- Coding
- Testing
- Bug Tracking
- Release Management
- etc.

22.2   Azure DevOps

Azure DevOps an Application Lifecycle Management (ALM) system, i.e., the system takes care of all aspects in software development from planning, requirements, coding, testing, deployment and maintenance.

Azure DevOps is a product designed specifically for software engineering teams with developers, testers, architects, project managers, etc.

Azure DevOps is a Source Code Control (SCC), Bug Tracking, Project Management, and Team Collaboration platform. Azure DevOps is tightly integrated with Visual Studio as Microsoft is the vendor of both Visual Studio and Azure DevOps. For more information, see [24].

Azure was previously called Team Foundation Server (TFS), then it was renamed to Visual Studio Team Services (VSTS). Now it has again changed name to Azure DevOps.

Here are some main features:

- SDLC Management (SDLC – Software Development Life Cycle)
- Software Team Collaboration
- Source Code Management
- Supports Agile, Scrum, CMMI
- Integrated Test Tools
- Automated Builds
- Built in Team Foundation Version Control (TSVC) + Support for Git repositories
- Built-in support for Azure DevOps in Visual Studio (Team Explorer)
- Plug-in for Eclipse (Team Explorer Everywhere)
- MSSCCI Provider for other IDEs like LabVIEW
- etc.

We will go through Azure DevOps in more detail in a later chapter.

22.3   SVN

SVN or Subversion uses an Open-Source License. SVN was established in 2000. Subversion is probably the version control system with the widest adoption today. Many different Subversion clients are available (Tortoise SVN, Mac: Versions, Xcode (built-in support for SVN)).

SVN uses a Centralized/Client–Server architecture.

22.4   CVS

CVS, or Concurrent Versions System, was established between 1986-1990. It is free of charge. CVS uses a client–server architecture. It is widely supported in different IDEs (Eclipse, Xcode, etc.).

22.5   Git

Git has become very popular today. Git is a Distributed Version Control System (DVCS). It was initially designed and developed by Linus Torvalds (Linux Guru) in 2005. Git is free of use.

22.6   Others

Above we have discussed the most popular SCC systems today. Here are some other systems as well:

- Mercurial
- Bazaar
- LibreSource
- Monotone
- BitKeeper

Look them up if you are interested!

22.7   Cloud-based SCC Hosting Services

For those who don't want to install their own SCC repository in their own network, can use the services in the cloud, either for free or for monthly payments.

Below we list some popular SCC hosting services:

- Azure DevOps Services (formerly known as Visual Studio Team Services)

- GitHub
- Bitbucket

22.7.1   Azure DevOps Services

Azure DevOps Services is an online SCC hosting service based on the Azure DevOps Server.

You can setup a Git repository.

You can use this solution for free for up to 5 users, then you need to pay a monthly fee for additional users.

Web site: www.visualstudio.com

22.7.2   GitHub

GitHub uses (as the name says) a Git repository.

Web site: www.github.com

GitHub is now part of Microsoft.

22.7.3   Bitbucket

With Bitbucket you can either use a Mercurial or a Git repository. It is free for 5 users.

Web Site: www.bitbucket.org

23   Bug Tracking Systems

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 (a moth) inside a computer in 1947 that made the program not behave as expected. This was the "first" real bug.

A "bug tracking system" or "defect tracking system" is a software application that is designed to help keep track of reported software bugs in software development efforts.

Bug Reporting and Tracking with Azure DevOps: https://youtu.be/0tlWdqWdFeQ

Having a bug tracking system is extremely valuable in software development, and they are used extensively by companies developing software products.
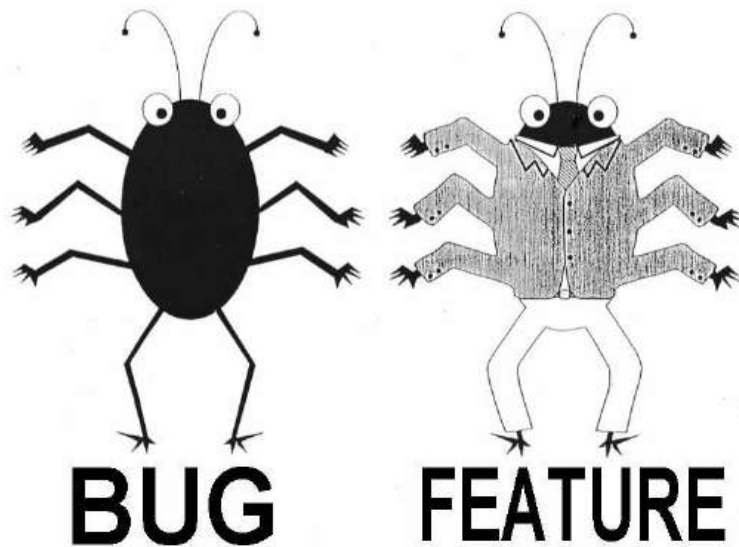
Here are some popular Bug Tracking Systems in use today:

- Azure DevOps
- Jira
- Bugzilla
- ClearQuest

We will focus on Azure DevOps in this document. The bug tracking features in Azure DevOps will be discussed in another chapter

24 Azure DevOps

Azure DevOps is an Application Lifecycle Management (ALM) system, i.e., the system takes care of all aspects in software development from planning, requirements, coding, testing, deployment, and maintenance.

Azure DevOps is a product designed specifically for software engineering teams with developers, testers, architects, project managers, etc.

It is free for up to 5 users and it is a good choice for small teams. It is also handy for personal use or students.

Getting started with Azure DevOps:

- Goto https://dev.azure.com
- Create an Account (You need a Windows Live ID) and specify an URL for your account
- Create a New Team Project
- You are ready to start
  - Connect to Azure DevOps from Visual Studio
  - Or use the Web based interface provided (except for SCC)
- Assign Team members
- Add Areas, Iterations, etc.
- Add your Source Code
- Check-in/Check-out your code

Azure DevOps with Scrum: https://youtu.be/-QmfMhtrxp0

Azure DevOps with Git and Visual Studio: https://youtu.be/tlFazeYmI_U

Bug Reporting and Tracking with Azure DevOps: https://youtu.be/0tlWdqWdFeQ

Azure DevOps (see Figure 24-1) is a Source Code Control (SCC), Bug Tracking, Project Management, and Team Collaboration platform. Azure DevOps is tightly integrated with Visual Studio as Microsoft is the vendor of both Visual Studio and Azure DevOps. For more information, see [24].
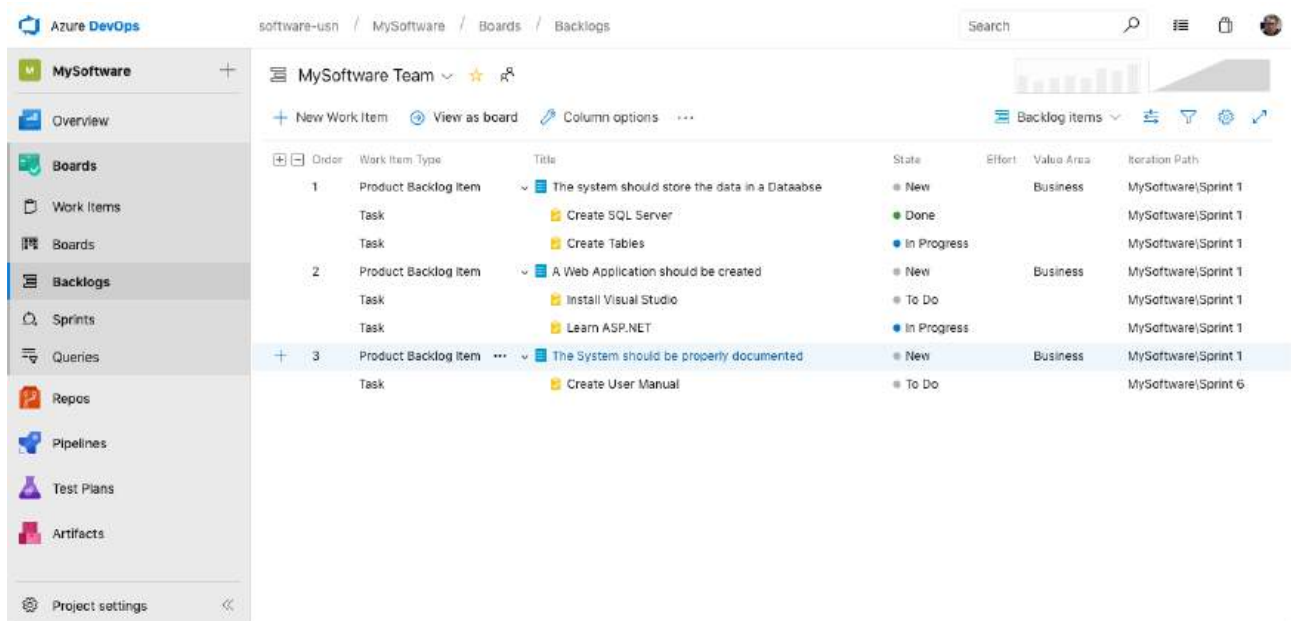
Figure 24-1: Azure DevOps

Here are some main features:

- SDLC Management (SDLC – Software Development Life Cycle)
- Software Team Collaboration
- Source Code Management
- Supports Agile and Scrum
- Integrated Test Tools
- Automated Builds
- Support for Git repositories
- Built-in support for Azure DevOps in Visual Studio
- etc.

Azure DevOps has plenty of competitors. The main benefit of Azure DevOps is that all the systems mentioned above is integrated in one package, normally you would need lots of different software for this.

24.1   Source Code Control (SCC)

With Azure DevOps you may use Git as source code repositories, see Figure 24-2.

Git is a Distributed Version Control System (DVCS) that uses a local repository to track and version files. Changes are shared with other developers by pushing and pulling changes through a remote, shared repository.
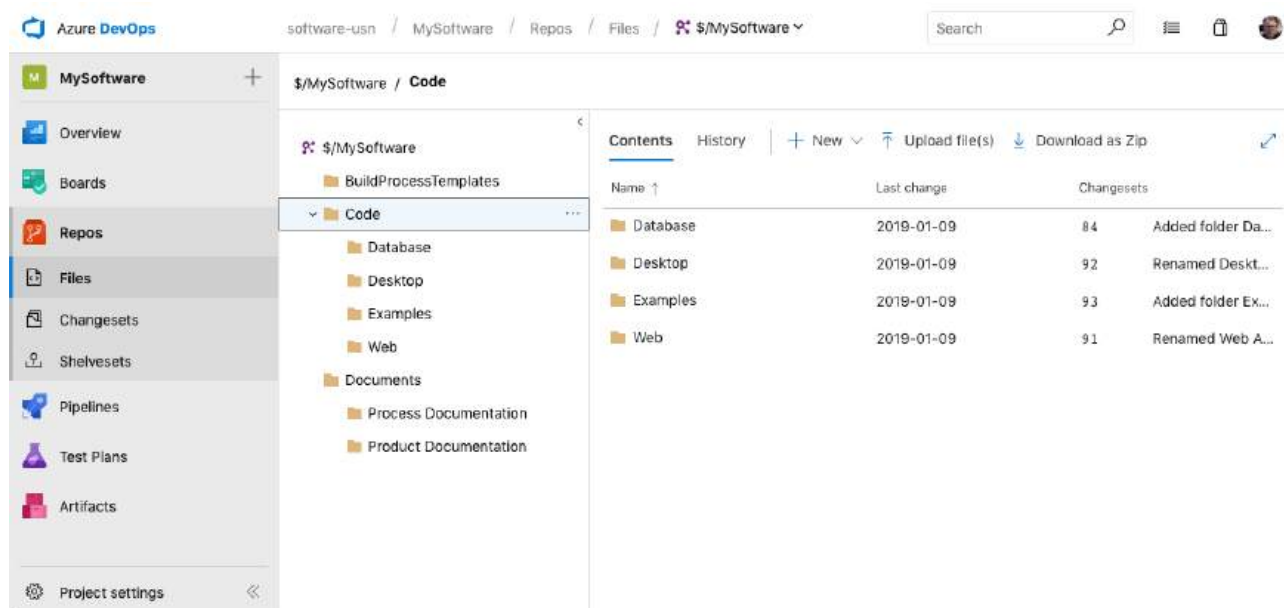
Figure 24-2: Start by creating a proper folder structure within your new Azure DevOps project

24.2   Areas and Iterations

Figure 24-3 shows an example of different Areas in Azure DevOps. The different software modules could be divided into different Areas.
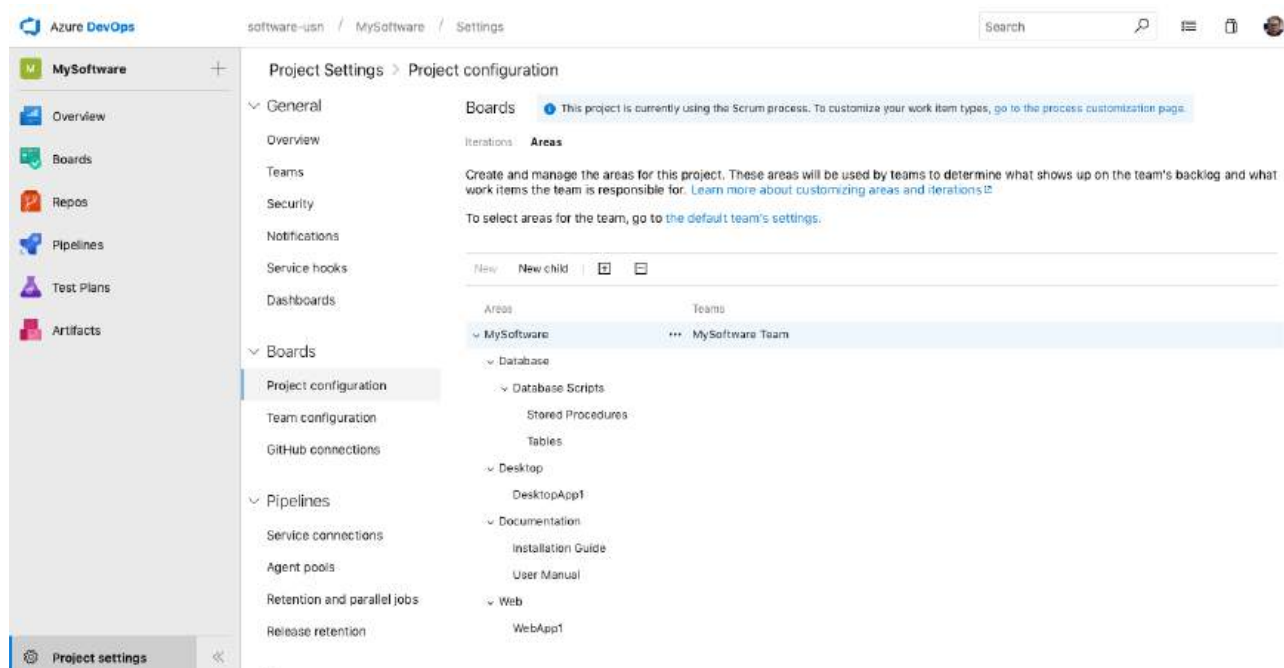
Figure 24-3: Azure DevOps Areas

Figure 24-4 shows an example of different Iterations in Azure DevOps. We can create Iterations for the different releases/milestones, such as Alpha, Beta1, 2, 3, RC, RTM.
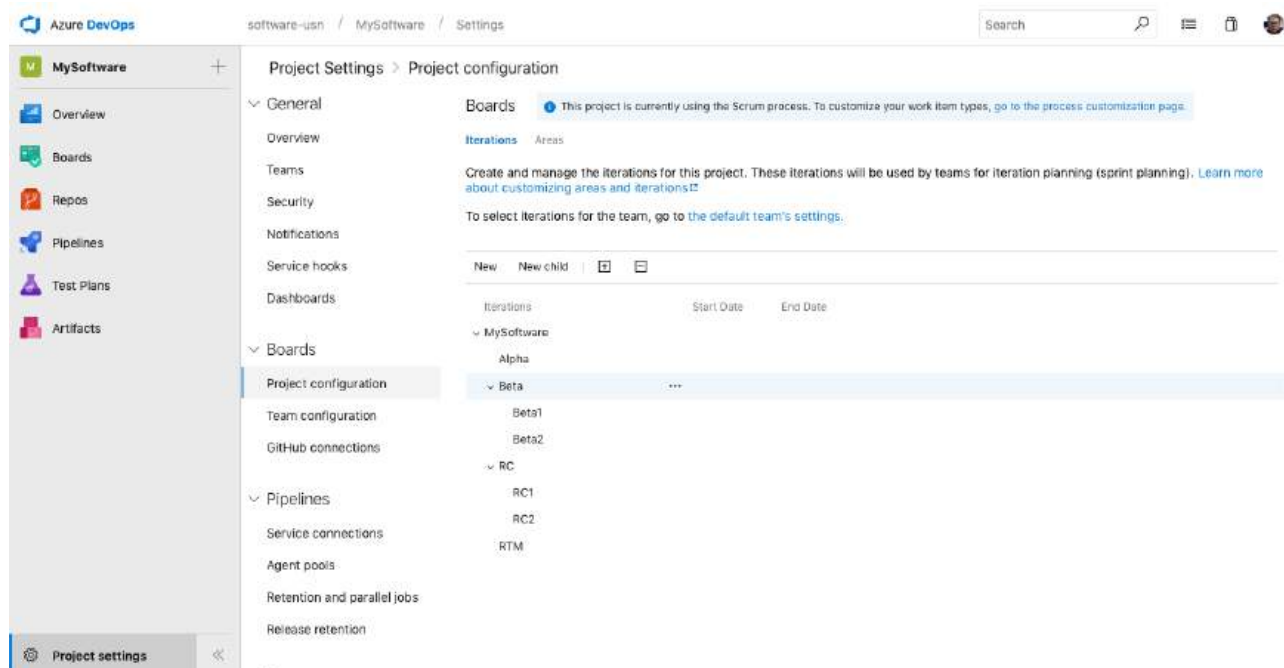
Figure 24-4: Azure DevOps Iterations

24.3   Work Items

Work Items (Figure 24-5) are an important part of Azure DevOps. You may use Work Items to register your requirements, user stories, bugs, tasks, etc.
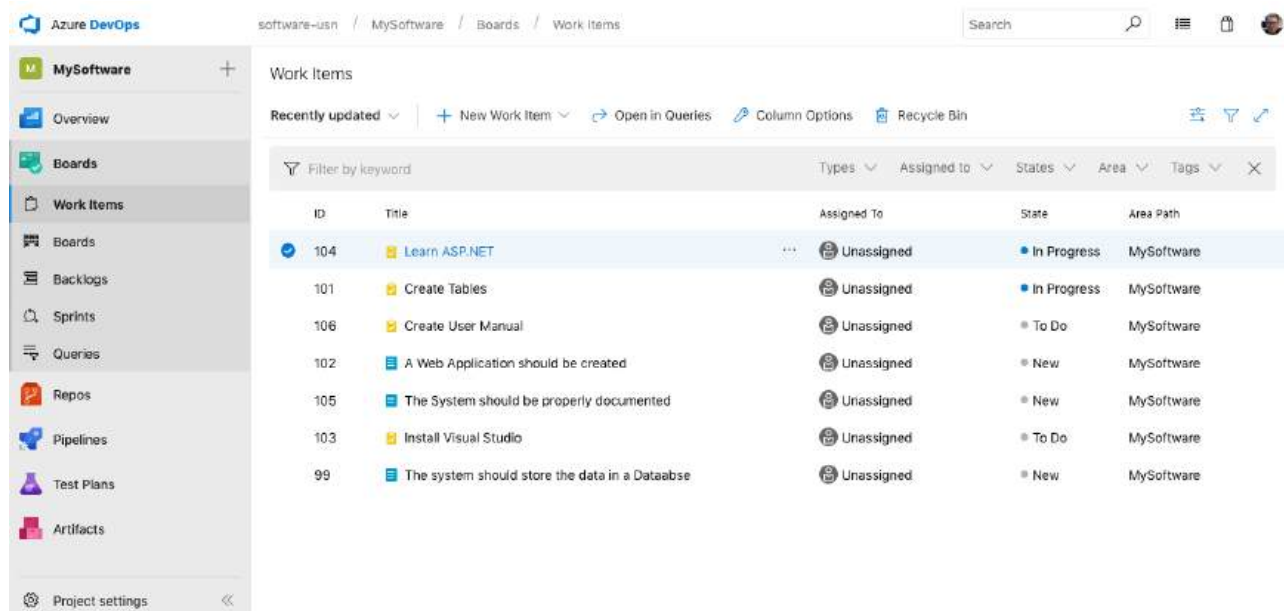
Figure 24-5: Work Items in Azure DevOps

In Azure DevOps you can create different Work Items, such as:

- Task

- Bug
- Feature
- Scenario
- Issue
- User Story
- Test Case
- etc.

These Work Items will be used at different level in your development cycle. When the Testers reports bugs, they will, e.g., use the "Bug" Work Item, etc.

In Figure 24-6 we see how we can enter new Work Items using the Work Item Editor.
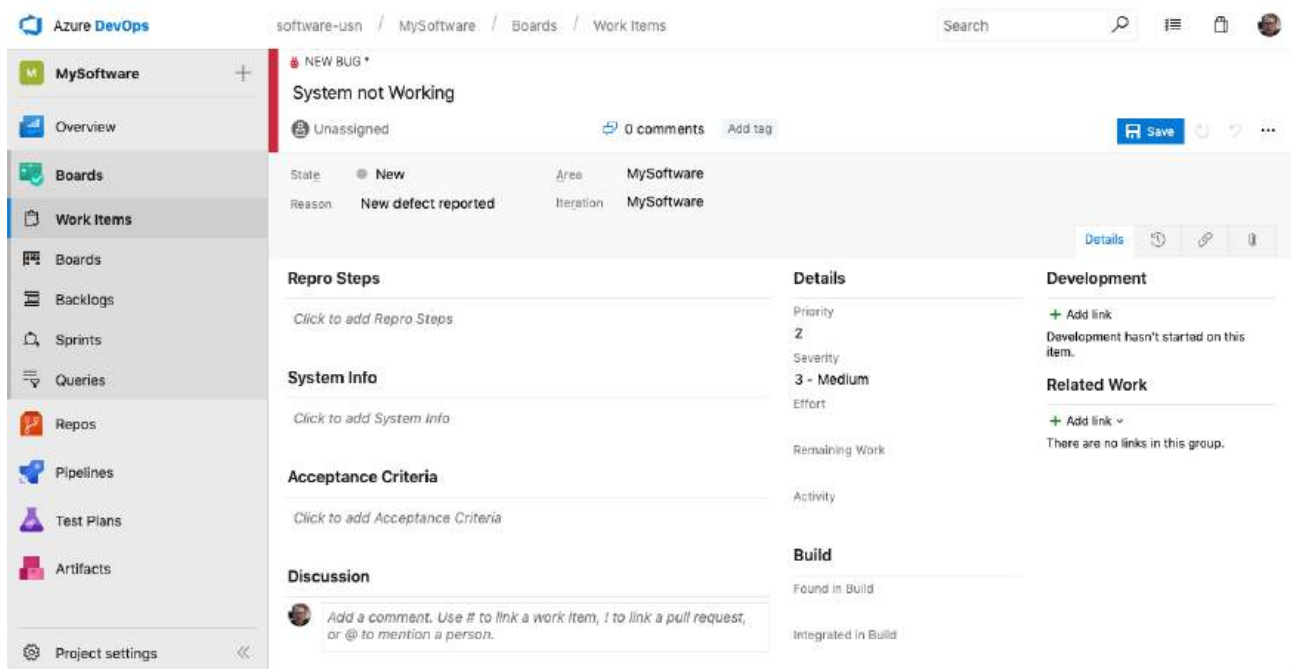
Figure 24-6: Work Item – New Bug

24.3.1   Queries

Queries are 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

You can use the Query Editor (Figure 24-7) to tailoring your own queries.
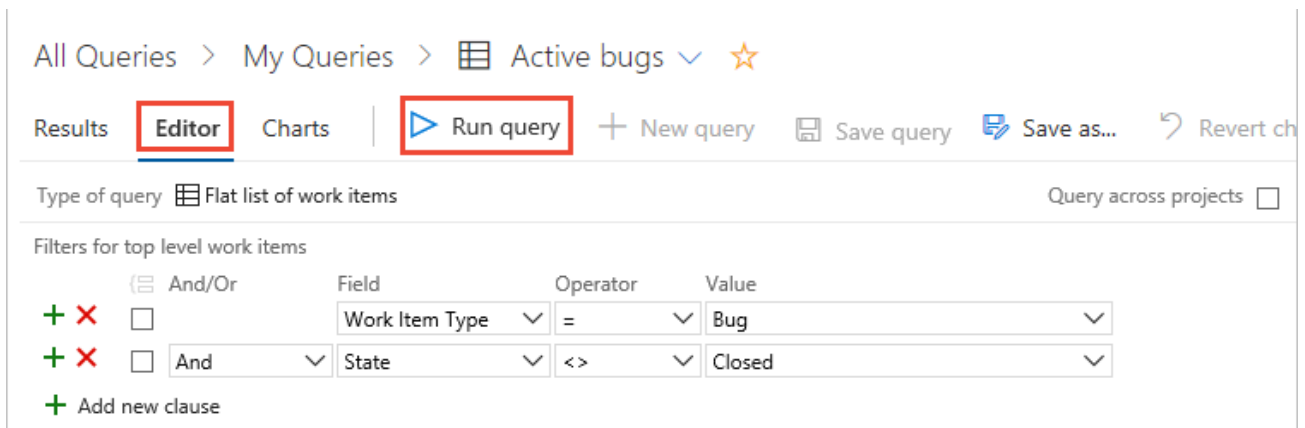
Figure 24-7: Query Editor

24.4   Taskboard

In Agile development and Scrum the Taskboard feature (Figure 24-8) is very useful.
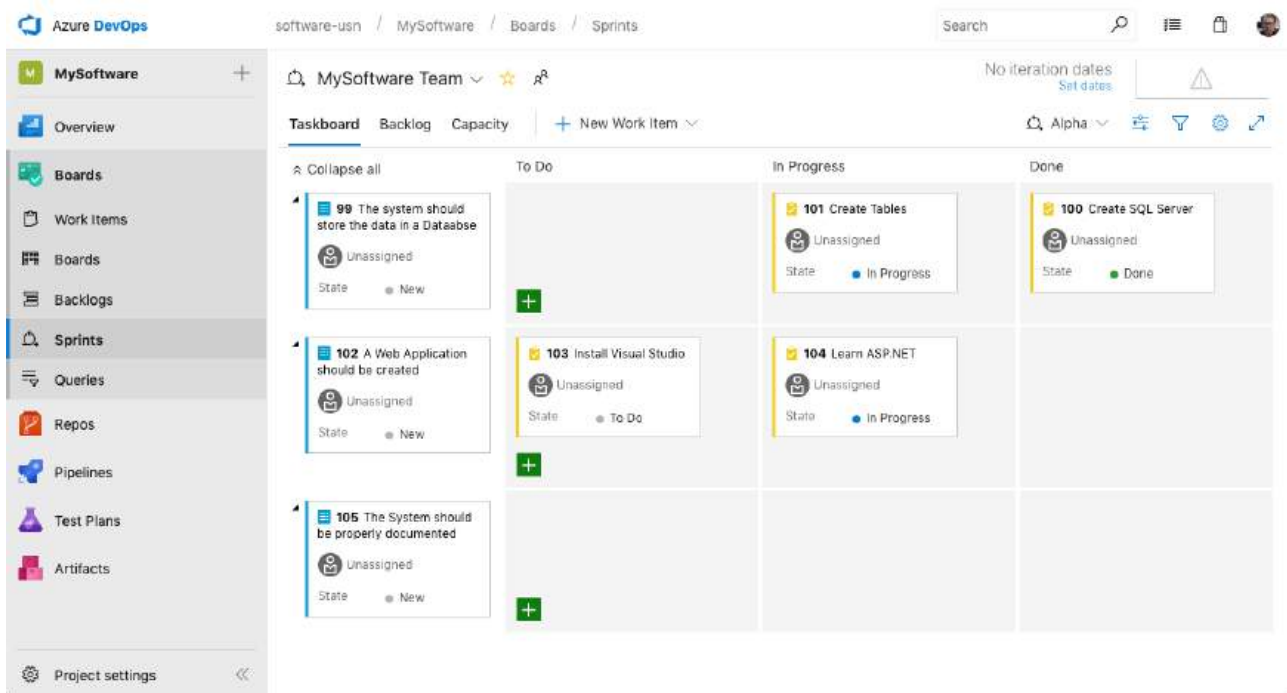
Figure 24-8: Taskboard

24.5   Agile (Scrum) Development in Azure DevOps

Azure DevOps has built-in Templates for Agile Development and Scrum.

Azure DevOps with Scrum: https://youtu.be/-QmfMhtrxp0

We have the Product Backlog, and the Sprint Backlog features (see Figure 24-9).
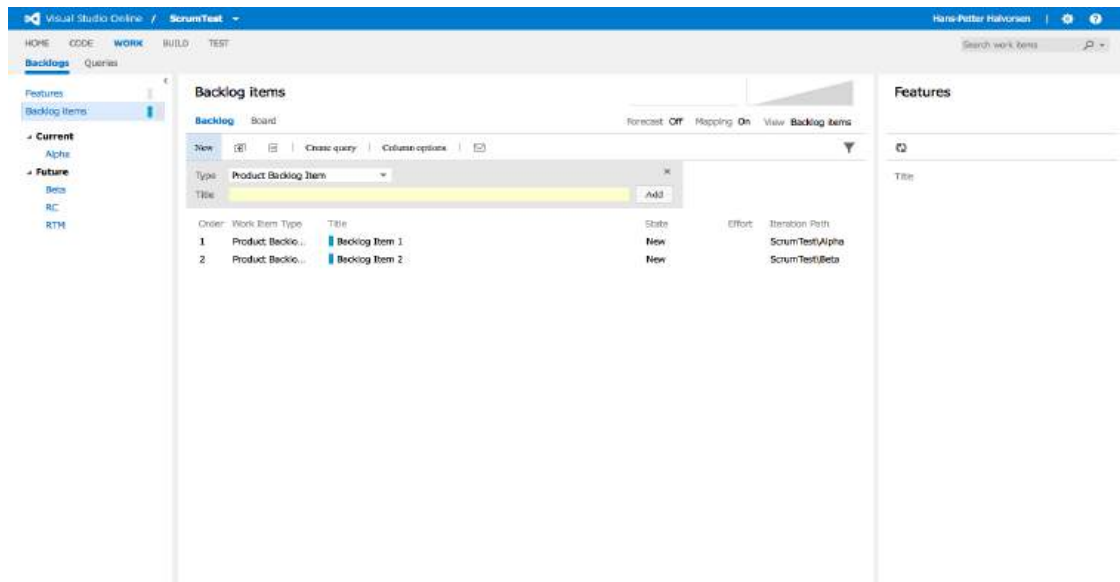
Figure 24-9: Product Backlog and Sprint Backlog in Azure DevOps

We also have a digital Taskboard available, see Figure 24-10.
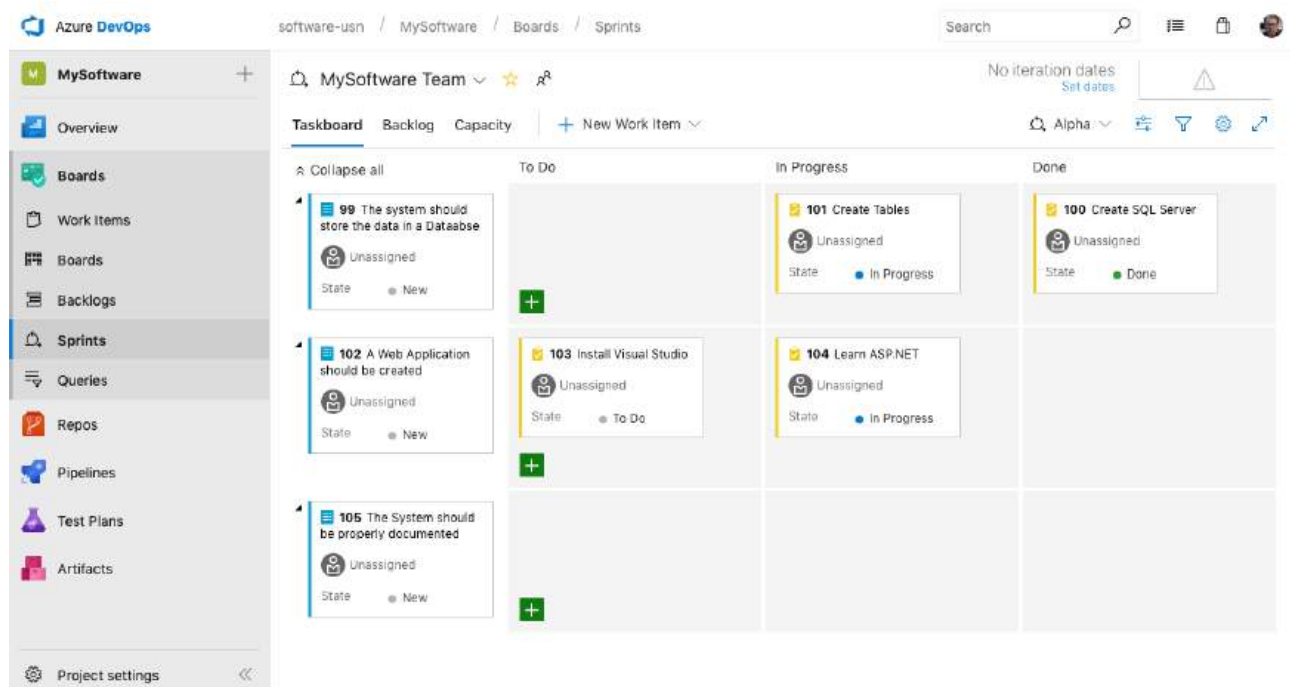
Figure 24-10: Using a Taskboard in Azure DevOps

24.5.1   Product Backlog Items in Azure DevOps

To create the Product Backlog, we can add them in Azure DevOps as so-called Work Items. If you use the Agile/Scrum templates a predefined Work Item Type "Product Backlog Item" is used for this purpose.
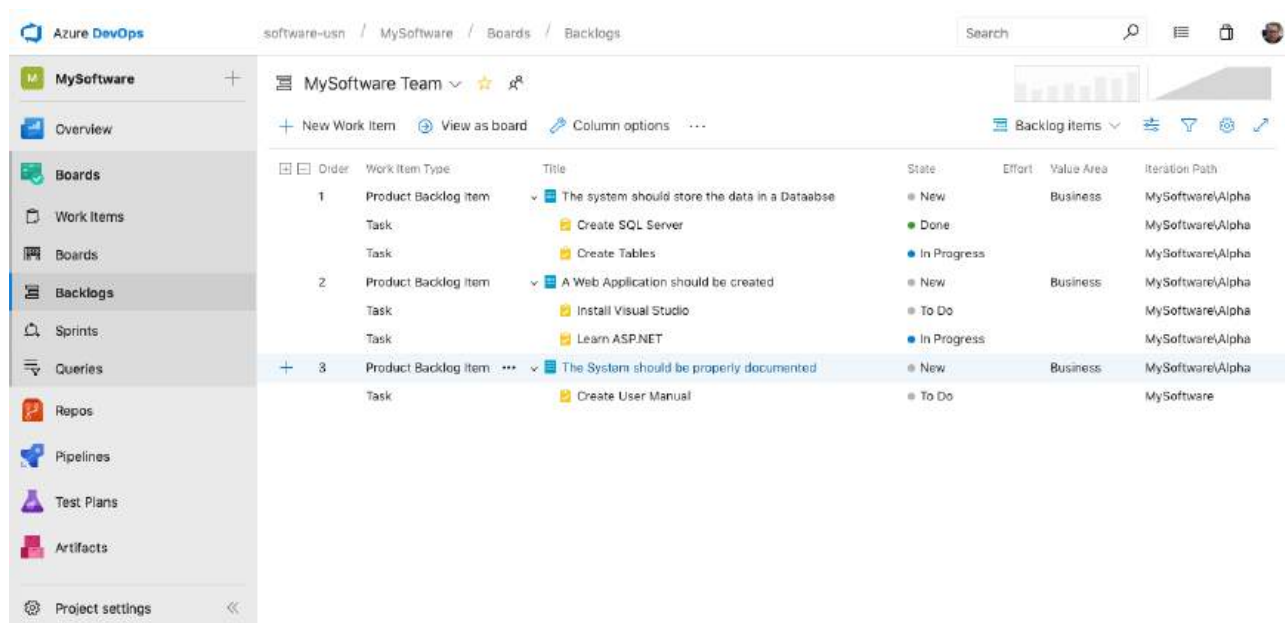
Figure 24-11: Create the Product Backlog in Azure DevOps

You may also group the items in the Product Backlog into "Features".

24.5.2   Sprint Backlog Items in Azure DevOps

To create the Sprint Backlog, you just drag them to the proper Iteration.
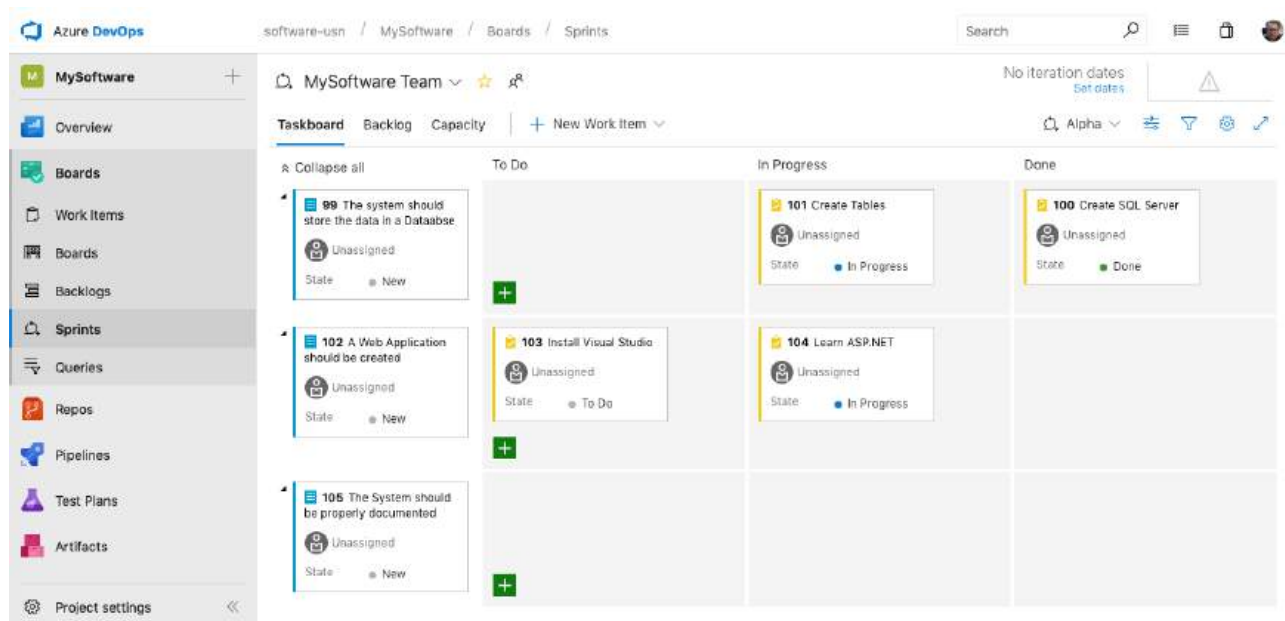
Figure 24-12: Create the Sprint Backlog in Azure DevOps

To make that happen, you need to configure the different sprints as "Iterations" in Azure DevOps (Figure 24-13). You also need to right-click and select "Set as Teams Backlog iteration".
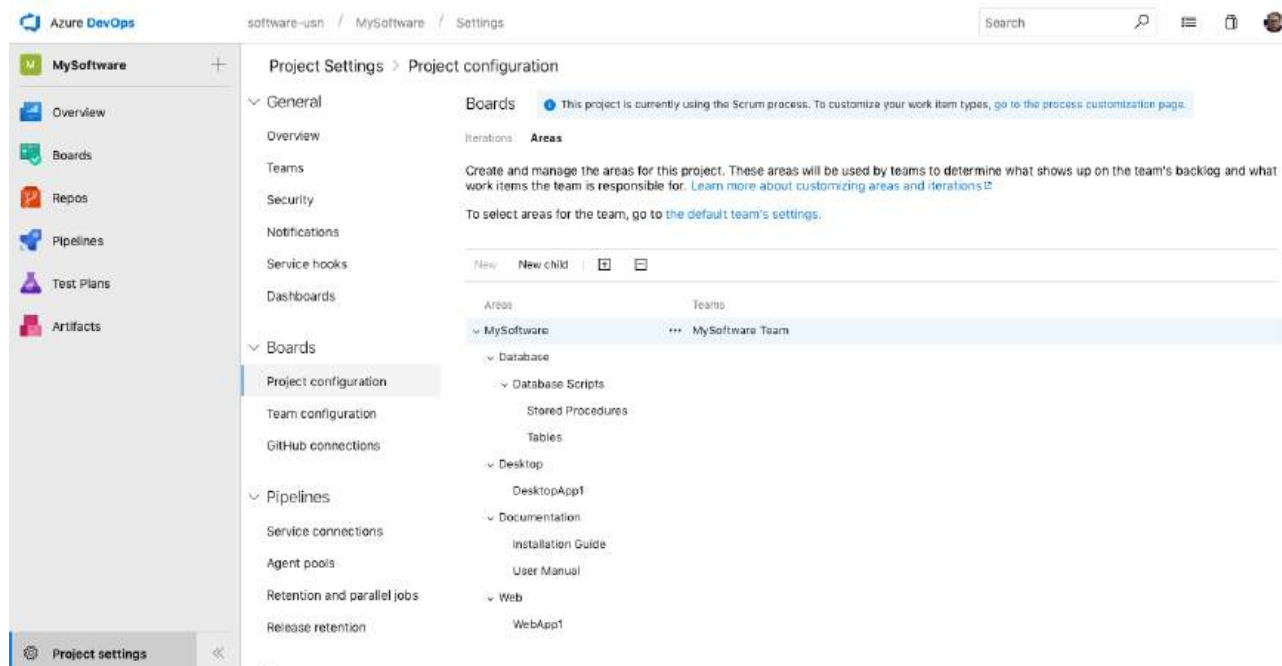
Figure 24-13: Configure Iterations and Sprints in Azure DevOps

Break Sprint Backlog Items down into Tasks:

Finally, you break the Sprint Backlog Items into Tasks. In the Sprint Backlog click the ✚ sign to add Tasks to the specific Backlog Item (see Figure 24-14).
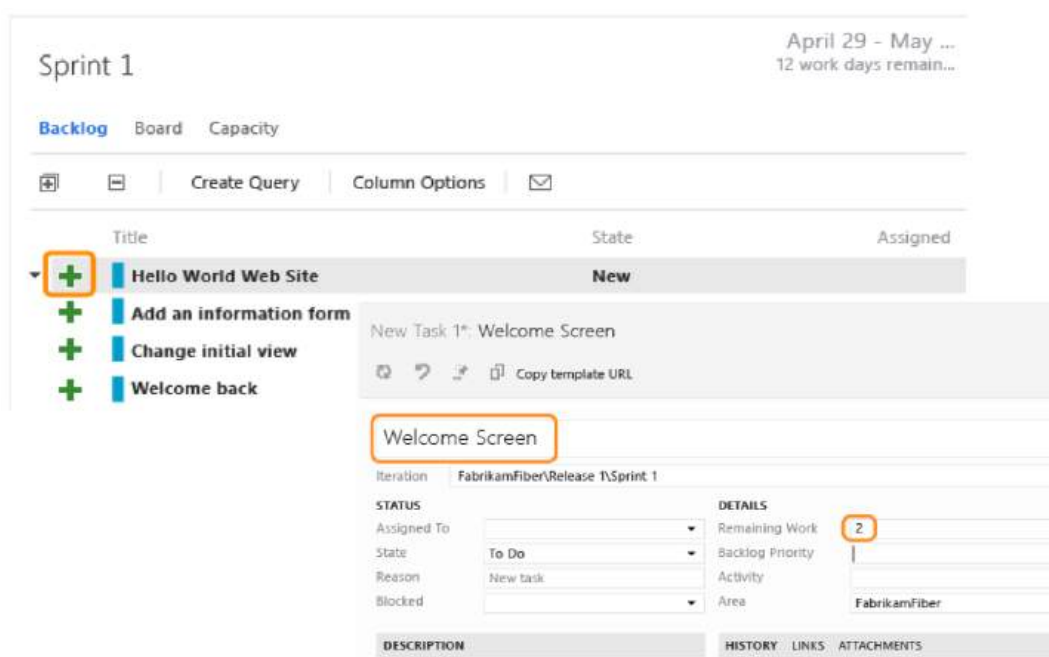
Figure 24-14: Break Sprint Backlog Items down into Tasks

Then give the Task a name and estimate the work it will take (Remaining Work).

The results may look something like this (Figure 24-15):
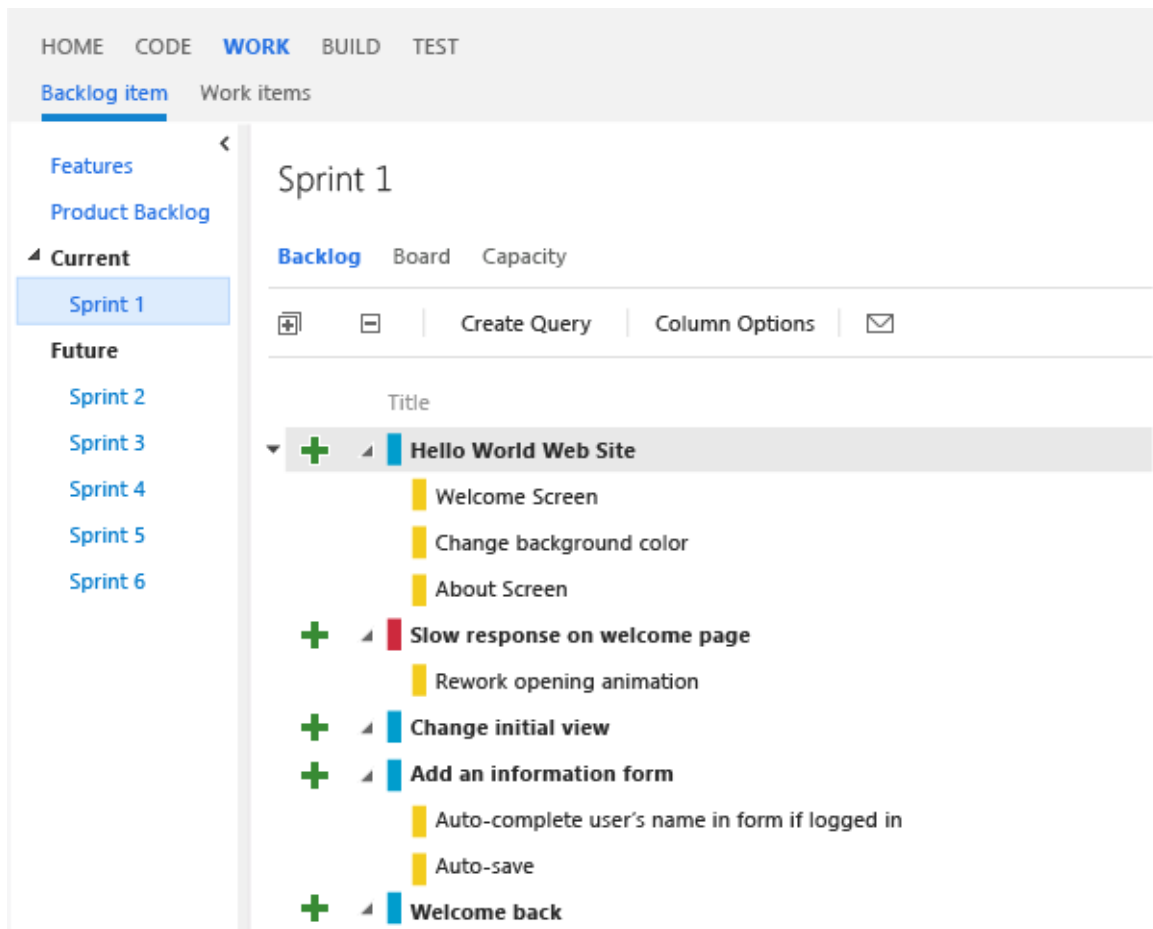
Figure 24-15: The Sprint Backlog divided into a hierarchical structure with Features, Backlog Items and Tasks

24.5.3   Taskboard

Based on the Sprint Backlog Items and the Tasks created for each Sprint Backlog Items we can start using the Taskboard features inside Azure DevOps.

The Taskboard is the heart of Daily Scrum Meetings. We can easily move tasks (drag and drop with the mouse) on the task board to reflect their current state.
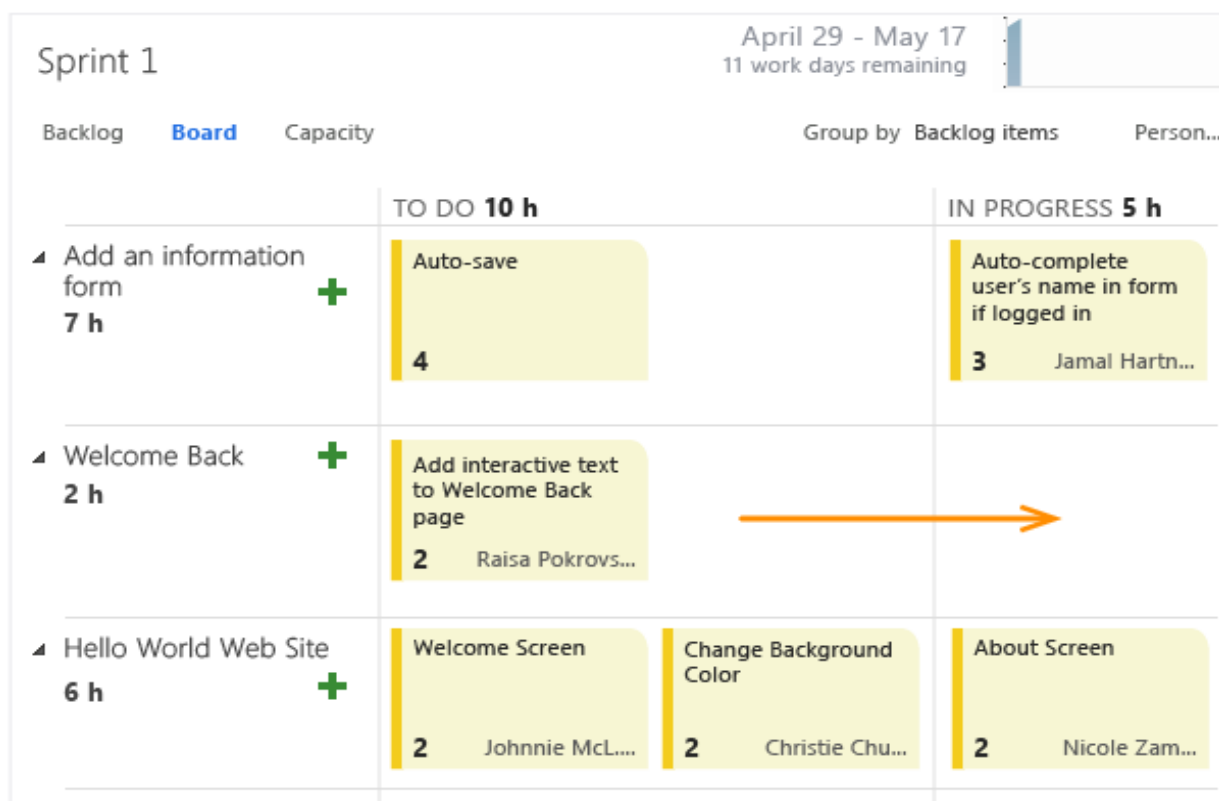
Figure 24-16: Updating the Sprint Taskboard in Azure DevOps

We can also easily Assign/reassign people to the different tasks in addition to updating the "Remaining Work" field before the Daily Scrum Meeting, see Figure 24-17.
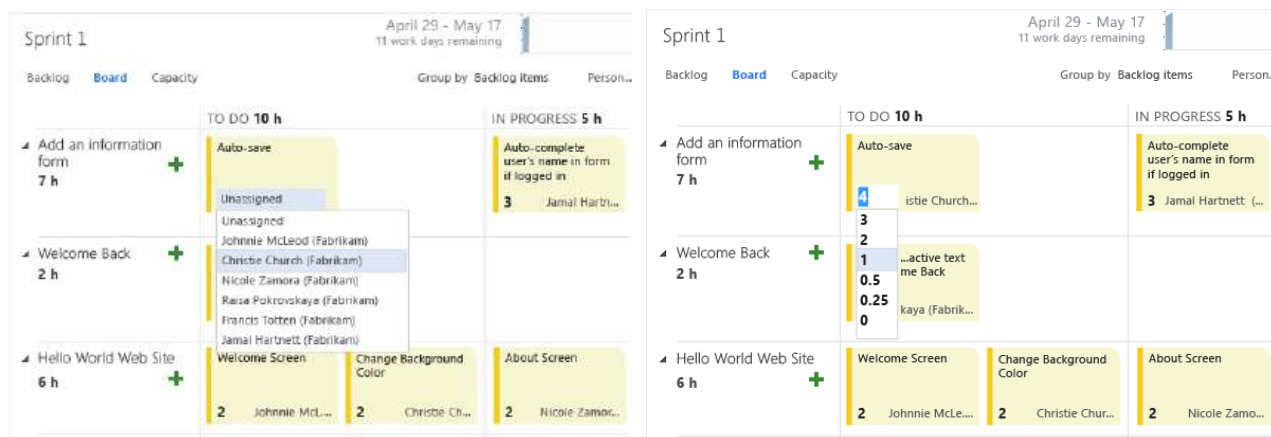
Figure 24-17: Using the Taskboard to update the Sprint Tasks before/under the Daily Scrum Meeting

24.6   Software Testing in Azure DevOps

Azure DevOps is a great tool for testing, for planning tests and it has advanced features for bug reporting and bug tracking (see Figure 24-18).
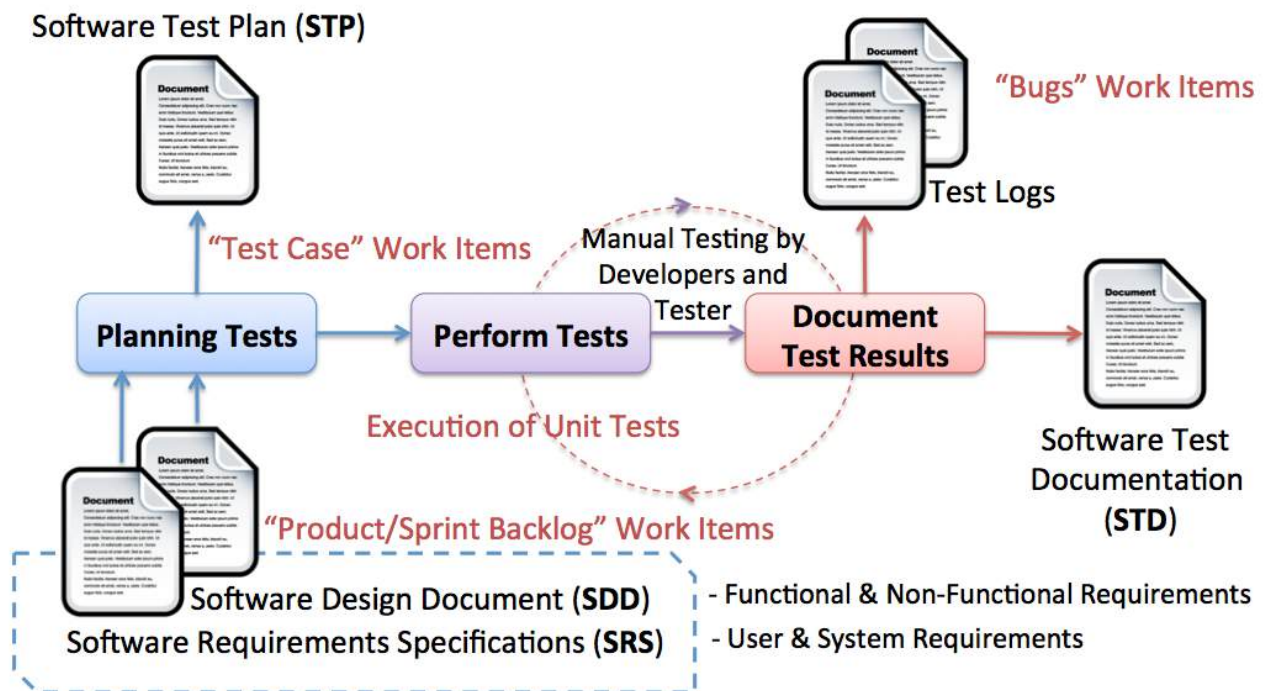
Figure 24-18: Software Testing in Azure DevOps/Visual Studio Team Services

25 Databases

Almost any kind of software program uses a database for back-end storage, e.g., Facebook, etc.

Popular Database Systems:

- Microsoft SQL Server
- Oracle
- MySQL
- SQLite
- MS Access
- MariaDB
- Etc.

The focus in this chapter will be Microsoft SQL Server. For more information about database systems, please see [25].

25.1 SQL Server

SQL Server consists of a Database Engine and a Management Studio. The Database Engine has no graphical interface - it is just a service running in the background of your computer (preferable on the server). The Management Studio is graphical tool for configuring and viewing the information in the database. It can be installed on the server or on the client (or both).

Introduction to SQL Server: https://youtu.be/SlR4KOhAG1U

SQL Server Express Installation: https://youtu.be/hhhggAlUYo8

SQL Server and Structured Query Language (SQL): https://youtu.be/sI6skicZse0

SQL Server with C# Windows Forms App: https://youtu.be/rXugzELsQl0

ASP.NET Core - Get Data from Database: https://youtu.be/pChkbx9BFVA
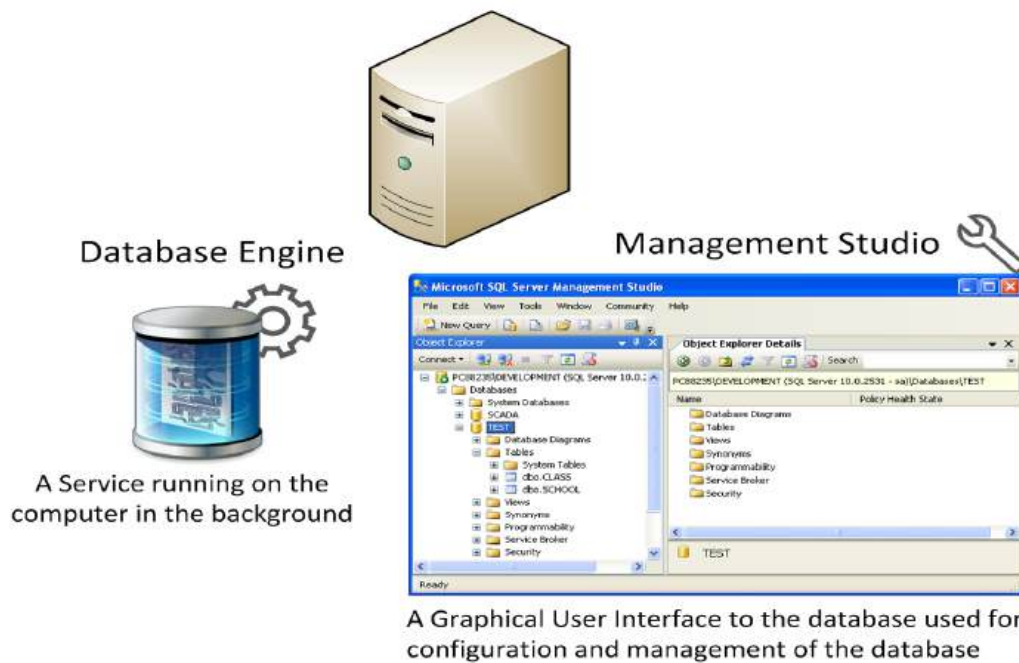
Figure 25-1: SQL Server

In Figure 25-2 we see the SQL Server Management Studio.
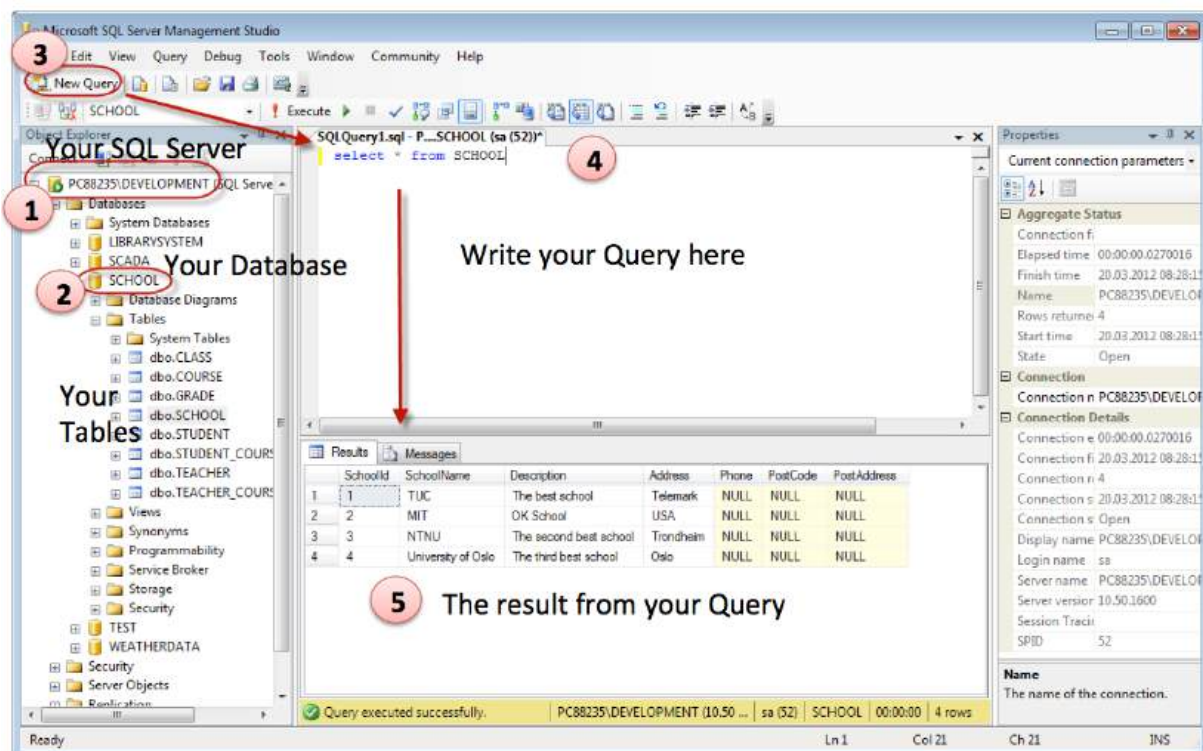
Figure 25-2: SQL Server Management Studio

25.2   ER Diagram

ER Diagram (Entity-Relationship Diagram) is used for design and modeling of databases. It specifies tables and relationship between them (Primary Keys and Foreign Keys), see Figure 25-3.
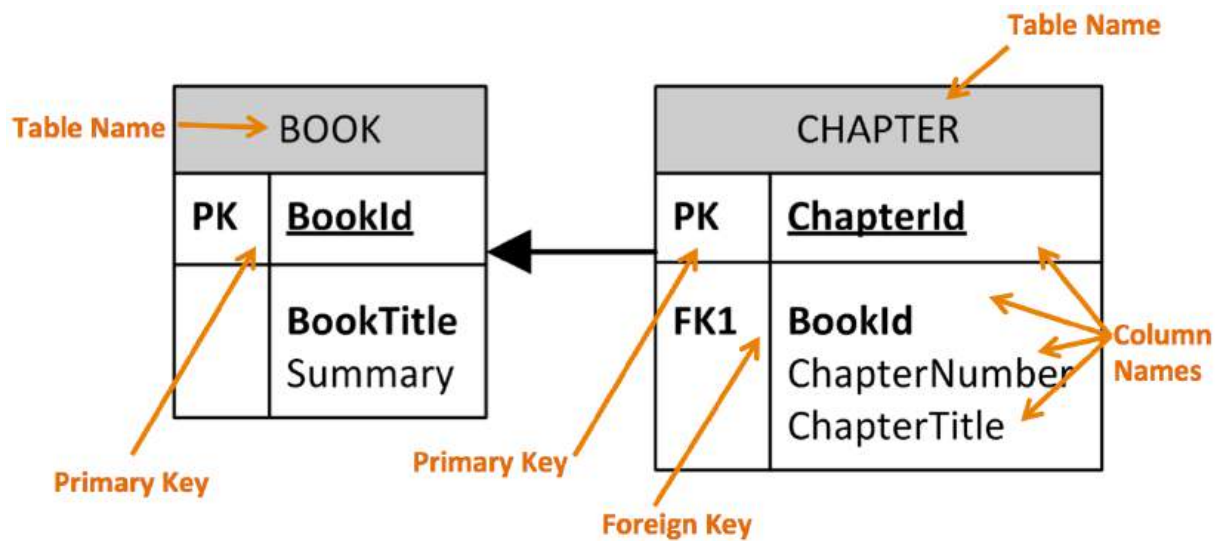
Figure 25-3: ER diagram with Primary Keys and Foreign Keys relationships

In Figure 25-4 we see a typical ER diagram.
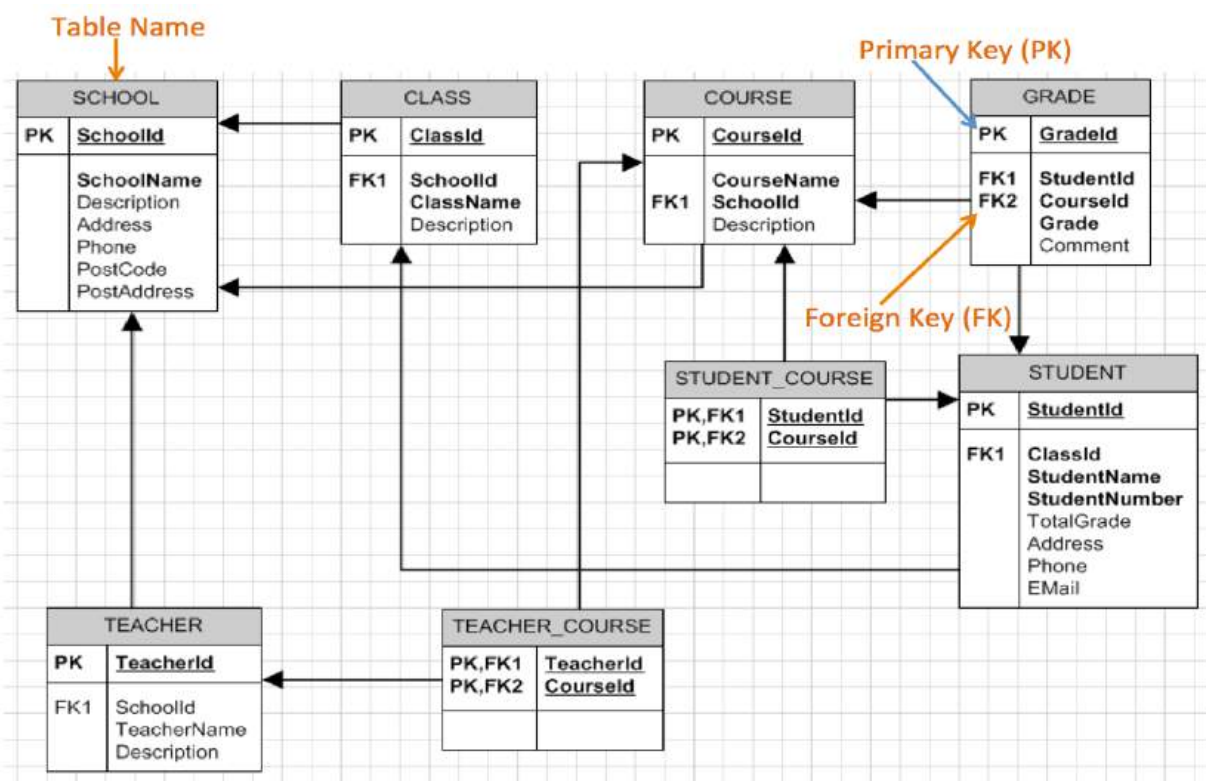
Figure 25-4: ER Diagram Example

25.3   ERD Tools

We can use a lot of different tools to create such ER diagrams. Some examples are DB Designer, Lucidchart and erwin Data Modeler.

Database Design and Modelling using DB Designer: https://youtu.be/jcBIzOfIxPs

Database Design and Modelling using Lucidchart: https://youtu.be/pztyVxBn8HM

erwin Data Modeler: https://youtu.be/jLKoluiJJ_k

25.4   Structured Query Language

Here we will only give a short introduction to Structured Query Language. For more information about SQL, please see [19].

SQL Server and Structured Query Language (SQL): https://youtu.be/sI6skicZse0

SQL is a database computer language designed for managing data in Relational Database Management Systems (RDBMS). In SQL, we have 4 different types of queries:

- INSERT
- SELECT
- UPDATE
- DELETE

Below we see some examples of typical SQL queries:

```
insert into STUDENT (Name, Number, SchoolId)
values ('John Smith', '100005', 1)

select SchoolId, Name from SCHOOL

select * from SCHOOL where SchoolId > 100

update STUDENT set Name='John Wayne' where StudentId=2

delete from STUDENT where SchoolId=3
```

These are referred to as **CRUD** – Create (Insert), Read (Select), Update and Delete.

25.4.1   Best Practice

Here are some "Best practice" recommendations for creating tables in a database system:

- **Tables**: Use upper case and singular form in table names – not plural, e.g., "STUDENT" (not students)
- **Columns**: Use Pascal notation, e.g., "StudentId"
- **Primary Keys**:
    - If the table name is "COURSE", name the Primary Key column "CourseId", etc.
    - "Always" use Integer and Identity(1,1) for Primary Keys
- Specify Required Columns (NOT NULL) – i.e., which columns that need to have data or not
- **Data Types**: Standardize on these Data Types: int, float, varchar(x), datetime, bit
- Use English for table and column names
- Avoid abbreviations! (Use RoomNumber – not RoomNo, RoomNr, ...)

26 Unit Testing

Unit Testing (or component testing) refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class and methods level. Unit Tests are written by the developers as part of the programming. Unit tests are automatically executed (e.g., Visual Studio and Azure DevOps have built-in functionality for Unit Testing).

Unit Testing in Visual Studio: https://youtu.be/QIfNViZkqEc

ASP.NET Core - Unit Testing: https://youtu.be/EzeDCEQ2qMs

Here are some "best practice" rules regarding Unit Testing:

- A Unit Test must only do one thing
- Unit Test must run independently
- Unit Tests must not be dependent on the environment
- Test Functionality rather than implementation
- Test public behavior: private behavior relates to implementation details
- Avoid testing UI components
- Unit Tests must be easy to read and understand
- Create rules that make sure you need to run Unit Tests (and they need to pass) before you can Check-in your Code in the Source Code Control System

26.1 Unit Tests Frameworks

Unit Tests are built into Visual Studio. Some other Unit Tests Frameworks are:

1. JUnit is a unit testing framework for the Java programming language.

2. Nunit: NUnit is an open-source unit testing framework for Microsoft .NET. It serves the same purpose as JUnit does in the Java world

3. LabVIEW Unit Test Framework Toolkit

4. etc.

26.2   Unit Testing in Visual Studio

Visual Studio has integrated possibilities for Unit Testing. In Figure 26-1 we see the Unit Test Project that is built into Visual Studio.

Unit Testing in Visual Studio: https://youtu.be/QIfNViZkqEc

ASP.NET Core - Unit Testing: https://youtu.be/EzeDCEQ2qMs

Note! Some of the more advanced test features in Visual Studio are only available in the Enterprise edition.
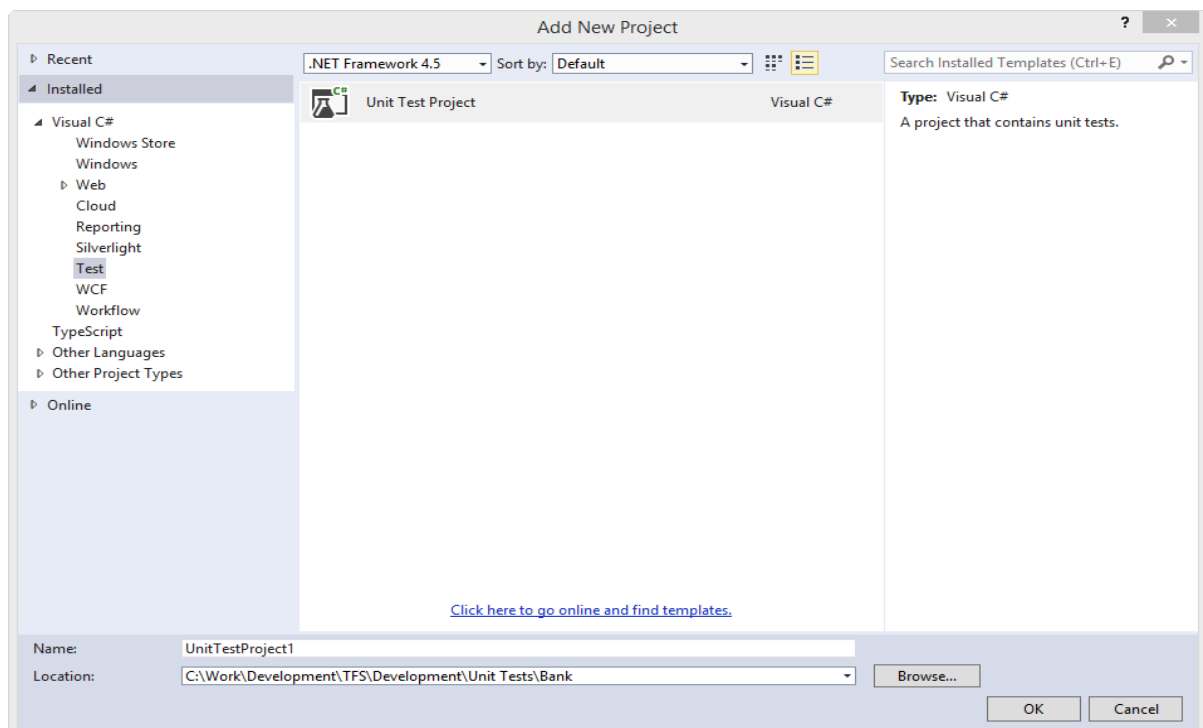
Figure 26-1: Unit Test Project used in Visual Studio

A typical Test Class in Visual Studio looks like this:

```
using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using BankAccount; //The Code that is going to be tested

namespace BankTest
{
    [TestClass]
    public class BankAccountTests
    {
        [TestMethod]
```

```
        public void TestMethod1()
        {
           …
        }
      …
    }
}
```

A test method must meet the following requirements:

- The method must be decorated with the [TestMethod] attribute.

- The method must return void.

- The method cannot have parameters.

Example of a Unit Test written in C#:

```
[TestMethod]
public void Debit_WithValidAmount_UpdatesBalance()
{
    // arrange
    double beginningBalance = 11.99;
    double debitAmount = 4.55;
    double expected = 7.44;
    BankAccount account = new BankAccount("Bryan Walton",  beginningBalance);
    // act
    account.Debit(debitAmount);
    // assert
    double actual = account.Balance;
    Assert.AreEqual(expected, actual, 0.001, "Account incorrectly");
}
```

We get an overview of all the Tests in the Test Explorer (Figure 26-2):
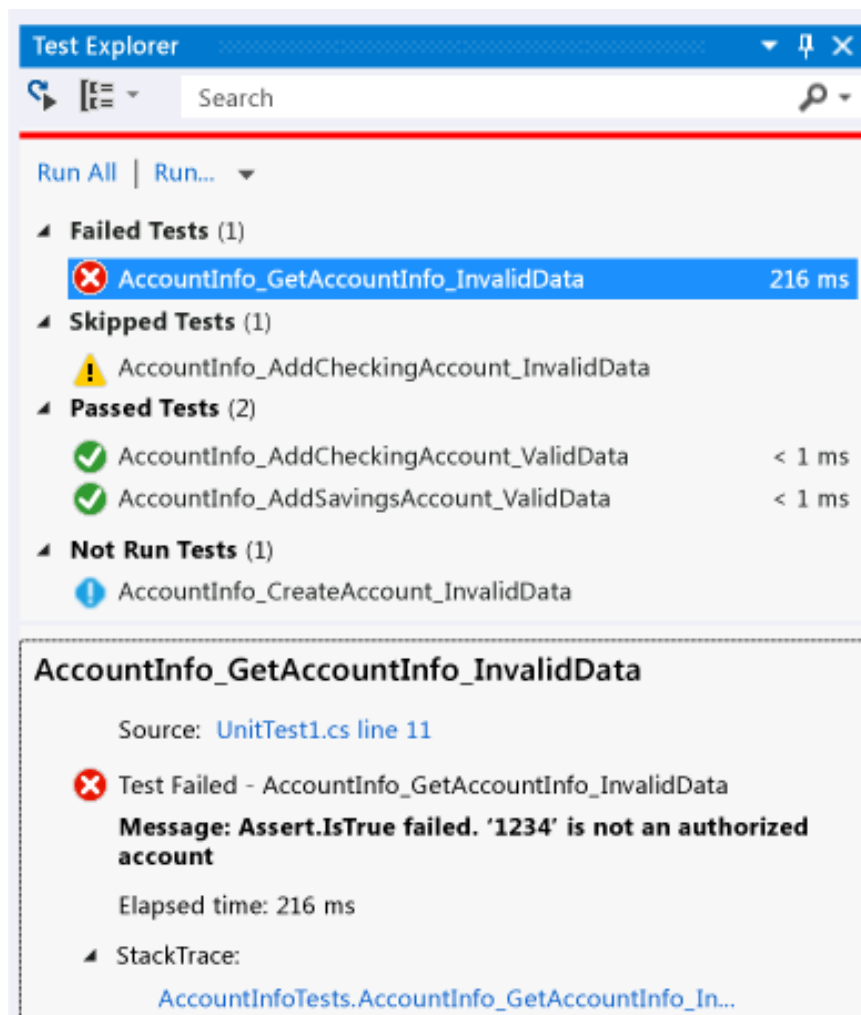
Figure 26-2: Test Explorer inside Visual Studio

When you build the test project, the tests appear in the Test Explorer. If the Test Explorer is not visible, choose Test on the Visual Studio menu, choose Windows, and then choose Test Explorer.

As you run, write, and rerun your tests, Test Explorer displays the results in default groups of Failed Tests, Passed Tests, Skipped Tests and Not Run Tests.

You can choose to run the tests manually or automatically, e.g., every time you build the code, etc. If you use Azure DevOps together with Unit Testing in Visual Studio you can also choose to force Unit Testing before you can check-in the code.

26.3   Code Coverage

Code coverage is a measure used in software testing. It describes the degree to which the source code of a program has been tested.

Example:

```
int foo (int x, int y)
{
    int z = 0;
    if ((x>0) && (y>0))
    {
        z = x;
    }
    return z;
}
```

When we test this function, it depends on the input arguments which parts of the code will be executed. Unit Tests should be written to cover all parts of the code.

In Visual Studio (Ultimate or Premium) these features are built-in (see Figure 26-3).
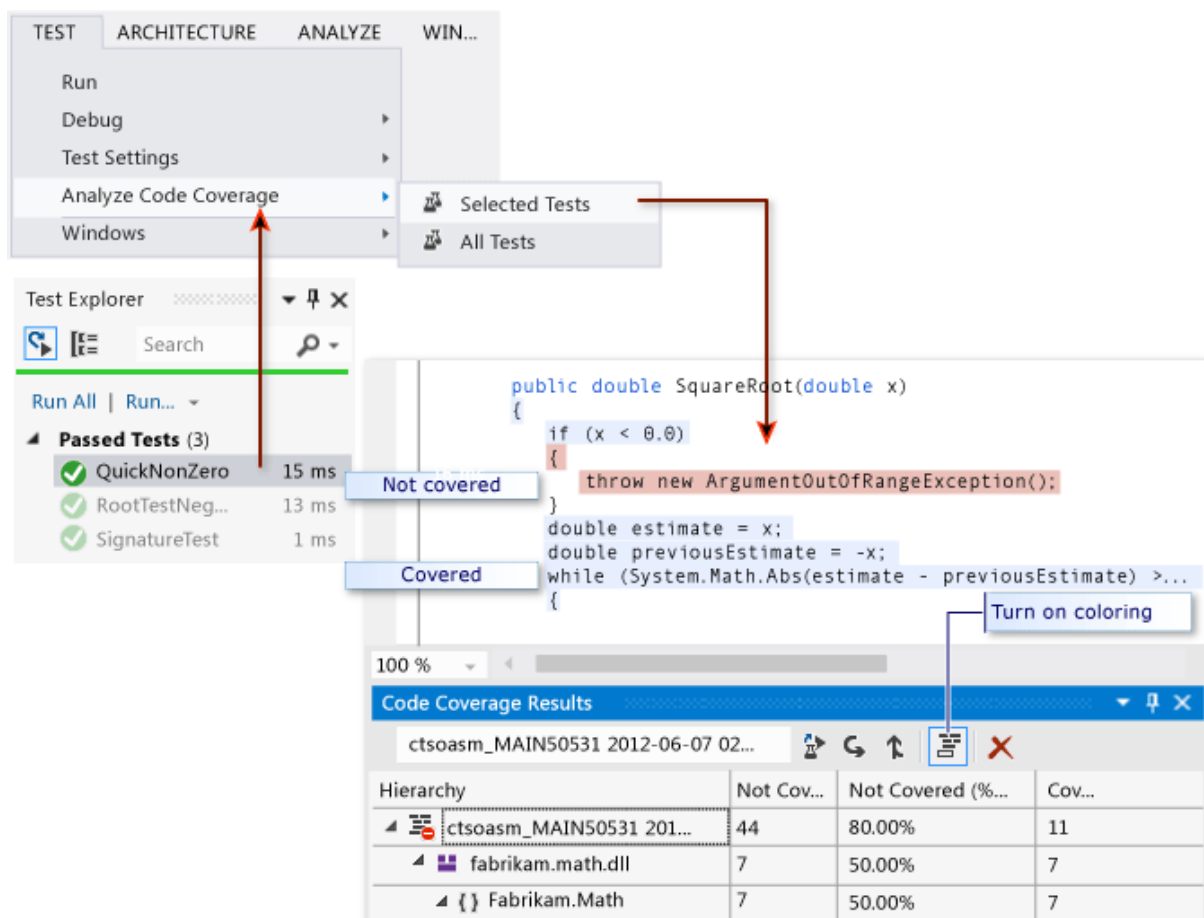
Figure 26-3: Code Coverage in Visual Studio

26.4   Exercises

Make sure to discuss and reflect over the following:

1. Explain the difference between unit testing and integration testing (or interface testing in Sommerville)

2.   Suggest "test cases" for the use case "Take out Money" from an ATM, i.e., give examples of
      Unit Tests and Integration Tests

27 Deployment in Visual Studio

Getting software out of the hands of the developers into the hands of the users. More than 50% of commissioned software is not used, mostly because it fails at the deployment stage. 80% of the cost of (commissioned) software comes at and after deployment. Visual Studio has many built-in features for deployment.

Deploy a Windows Forms App using Visual Studio: https://youtu.be/qXS9ie3KZFE

Virtualization Deployment:

Install WinForm Desktop App in Virtual Test Environment using VirtualBox: https://youtu.be/g7CPEVFT8AA

Install ASP.NET Core Web App in Virtual Test Environment using VirtualBox: https://youtu.be/7XrRd7voasI

Microsoft Azure Deployment:

Microsoft Azure - SQL Databases and App Services: https://youtu.be/ca6Q6LdshIs

Is it a Generic Software Product or a Tailor-made Software Solution? Different Deployment/Installation preparations required!

We have

- Generic Software:
    - Many Customers
    - The Customers install the Software itself
- Tailor-made:
    - Typically, only one Customer
    - The Developer Company typically installs the software (at least server-side components)
    - If many Desktop Clients: A Setup is required

Different Deployment/Installation preparations required depending on what type of Apps you are Developing.

Desktop Apps

– **You need to create an .exe file and a Setup Package**

– Setup packages can then be distributed on CDs/DVDs or downloaded from a Web Page

– You use Setup to install the software on all the clients

– Time consuming, cumbersome, depends on local components that might not be installed, version conflicts, etc. This makes it difficult (and a lot of work and testing) to create robust setup packages

– Mac: You can deploy to Mac App Store, Windows 8: You can deploy to Windows Store

Web Apps

• No <u>client</u> installation is needed!

• Installed on a **Web Server** (IIS, Apache)

• Accessed on the Clients using only a Web Browser

• Easy, simple to deploy new versions, bugfixes, etc. (Customers don't need to do anything)

• But make sure your App supports all major Web Browsers (Internet Explorer, Chrome, Firefox, Opera, Safari)

Mobile Apps

• Deployed to "**App Stores**" like Apple App Store, Google Play, Windows Store (Windows 8)

Server-side (Database, Web Services, etc.)

• Typically, a setup package that installs this, or manually if it is a tailor-made solution

27.1   Setup Creation Software

Here are some examples:

• **InstallShield** Professional/Premium

- InstallShield is professional software for creating installers, Price €2000+

- **WiX Toolset** (Windows Installer XML)

  - Used to create Windows Installer packages ("MSI files)

  - The WiX toolset builds Windows installation packages from XML source code.

  - Free and Open Source

  - Used by e.g., Microsoft to create Setup packages for Office, SQL Server, Visual Studio, etc. Apple also uses it.

- **Inno Setup**

  - Free of charge Installer for Windows programs

- **NSIS** (Nullsoft Scriptable Install System)

  - Professional open-source system to create Windows installers.

- etc.

27.2   ASP.NET Core Deployment

ASP.NET Core is a framework for creating professional web applications and it is fully integrated with Visual Studio.

Microsoft Azure - SQL Databases and App Services: https://youtu.be/ca6Q6LdshIs

Part 5 : Cyber Security

In this part, Cyber Security will be discussed in the context of Software Engineering and Software Development.

28 Cyber Security

28.1 Introduction

Cybersecurity is the practice of protecting systems, networks, and programs from digital attacks.

These cyberattacks are usually aimed at accessing, changing, or destroying sensitive information, extorting money from users, or interrupting normal business processes.

With the good comes the bad. Since the Internet connected all devices together a new era of our life was a fact.

The Internet is great for many things, but it is also a great place for criminals.

Figure 28-1 we see the evolution of computers and internet from the early begging and until today.
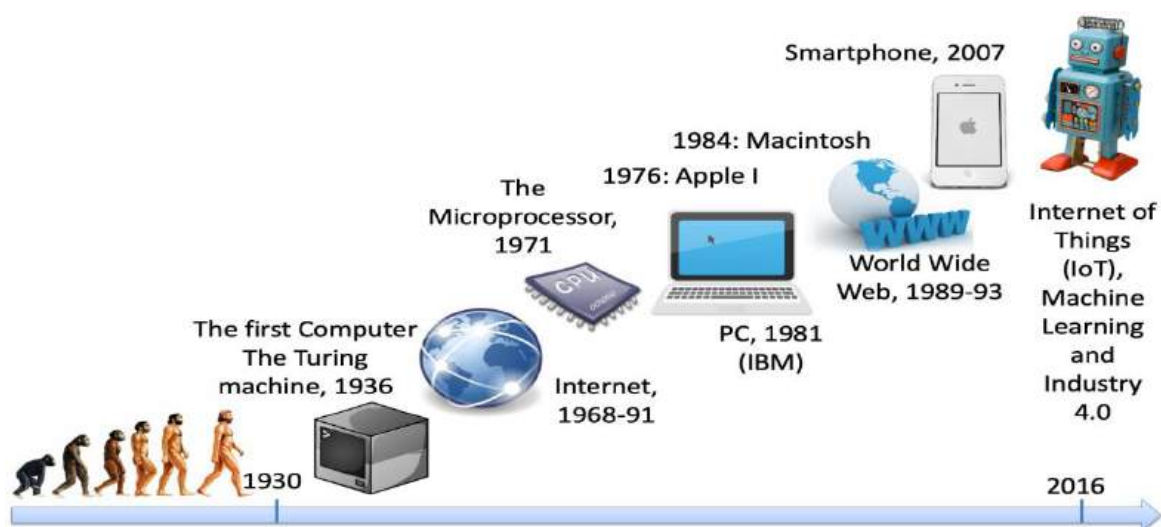
Figure 28-1: The Evolution of Computers and Internet

You probably use hundreds of different Internet services. Is your personal data safe within these companies?

- Is the data well protected (from hackers)?
- Is the data sold to other companies (advertising purposes)?
- Can you get an overview of the information stored on you?
- Is it possible to delete it?

Some terms:

Data Security: Protect digital data (e.g., data in a database) from destructive forces and from the unwanted actions of unauthorized users (e.g., hackers, etc.).

Data Privacy: Issues regarding your personal data stored.

Cyber Security is the practice of protecting systems, networks, and programs from digital attacks.

Cybersecurity, or computer security, is a catchall term for any strategy for protecting one's system from malicious attacks aimed at stealing money, personal information, system resources (cryptojacking, botnets), and a whole host of other bad things. The attack might occur on your hardware or software, or through social engineering.

GDPR: General Data Protection Regulation. EU directive. Purpose: Protect privacy and the data stored, i.e., protection of your digital life.

Figure 28-2 shows an overview of Cyber Security Issues.

Figure 28-2: An overview of Cyber Security Issues

28.2   Types of Cyber Security Attacks

Different types of Cyber Security threats:

- Ransomware
- Malware
- Social engineering
- Phishing
- SQL Injection
- Etc.

These will be discussed in more detail below.

28.2.1   Ransomware

Ransomware is a type of malicious software. It is designed to extort money by blocking access to files or the computer system until the ransom is paid.

These include, e.g., email phishing and malvertising (malicious advertising). After it is distributed, the ransomware encrypts selected files and notifies the victim of the required payment.

Paying the ransom does not guarantee that the files will be recovered, or the system will be restored.

The most "famous" Ransomware is the WannaCry Ransomware.

28.2.2   Malware

Malware is a type of software designed to gain unauthorized access or to cause damage to a computer. Malware is short for "malicious software".

Examples of common malware include viruses, worms, Trojan viruses, spyware, adware, and ransomware.

28.2.3   Social Engineering

Social engineering is a tactic that adversaries use to trick you into revealing sensitive information. They can solicit monetary payment or gain access to your confidential data. Social engineering can be combined with any of the threats listed above to make you more likely to click on links, download malware, or trust a malicious source.

28.2.4   Phishing

Phishing is the practice of sending fraudulent emails that resemble emails from reputable sources.

The aim is to steal sensitive data like credit card numbers and login information, or to install malware on the victim's machine.

Phishing is the most common type of cyber-attack.

You can help protect yourself through education (teach them not to click on links, etc. from untrusted sources) or a technology solution that filters malicious emails.

Types of phishing attacks:

- **Deceptive phishing** - Deceptive phishing is the most common type of phishing. In this case, an attacker attempts to obtain confidential information from the victims. Attackers use the information to steal money or to launch other attacks. A fake email from a bank asking you to click a link and verify your account details is an example of deceptive phishing.
- **Spear phishing** - Spear phishing targets specific individuals instead of a wide group of people. Attackers often research their victims on social media and other sites. That way, they can customize their communications and appear more authentic. Spear phishing is often the first step used to penetrate a company's defenses and carry out a targeted attack.
- **Whaling** - When attackers go after a "big fish" like a CEO, it's called whaling.
- **Pharming** - pharming sends users to a fraudulent website that appears to be legitimate. However, in this case, victims do not even have to click a malicious link to be taken to the bogus site.

28.2.5   Spam

Spam is unsolicited and unwanted junk email sent out in bulk to an indiscriminate recipient list. Typically, spam is sent for commercial purposes. It can be sent in massive volume by botnets and networks of infected computers.

Often, spam email is sent for commercial purposes. While some people view it as unethical, many businesses still use spam. The cost per email is incredibly low, and businesses can send out mass quantities consistently. Spam email can also be a malicious attempt to gain access to your computer.

Spam email can be difficult to stop, as it can be sent from botnets. **Botnets** are a network of previously infected computers. As a result, the original spammer can be difficult to trace and stop.

Spam email can be dangerous. It can include malicious links that can infect your computer with malware (see What is malware?). Do not click links in spam. Dangerous spam emails often sound urgent, so you feel the need to act.

28.2.6   SQL Injection

A Structured Query Language (SQL) injection occurs when an attacker inserts malicious code into a server that uses SQL and forces the server to reveal information it normally would not.

An attacker could carry out a SQL injection simply by submitting malicious code into a vulnerable website search box.

28.3   How to be Secure?

How can you avoid cyber-attacks in general?

What can you do as a company or a private person?

Here are some examples:

- Passwords
- Firewall
- Antivirus and antimalware software
- Access control
- VPN
- Wi-Fi Network
- Education

These will be discussed in more detail below.

28.3.1   Passwords

Make sure to use secure passwords, don't use the same password for all your services and software systems.

28.3.2   Firewall

A firewall is a network security device that monitors incoming and outgoing network traffic and decides whether to allow or block specific traffic based on a defined set of security rules.

Firewalls are the first line of defense in network security.

A firewall can be hardware, software, or both.

28.3.3   Web Application Firewall (WAF)

A web application firewall (WAF) is an application firewall for HTTP applications. It applies a set of rules to HTTP conversation. Generally, these rules cover common attacks such as cross-site scripting (XSS) and SQL injections.

The WAF software (or hardware) is installed on the server that runs the web applications or web sites.

A WAF is used for protection of a specific web application or set of web applications.

A WAF creates a shield between the web application and the Internet, which can avoid many common attacks.

A WAF helps protect web applications by filtering and monitoring HTTP traffic between a web application and the Internet. It typically protects web applications from attacks such as cross-site forgery, cross-site-scripting (XSS), file inclusion, and SQL injection, among others.

Most professional hosting platforms offer WAP solutions, like Amazon Web Services.

In addition, there are many types of WAF software you can buy and use, e.g., Imunify360.

References:

https://www.cloudflare.com/learning/ddos/glossary/web-application-firewall-waf/

https://www.owasp.org/index.php/Web_Application_Firewall

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

28.3.4   Antivirus and Antimalware Software

"Malware", which is short for "malicious software", includes viruses, worms, Trojans, ransomware, and spyware. Sometimes malware will infect a network but lie dormant for days or even weeks. The best antimalware programs not only scan for malware upon entry, but also continuously track files afterward to find anomalies, remove malware, and fix damage.

The Windows 10 operating system has built in antivirus software called "Windows Defender".

The name "Antivirus" software is a little old, because viruses are just one kind of malware in today's world of cyber threats. Though viruses still exist, there are other forms of malware that are more common these days, as mentioned in an earlier chapter.

28.3.5   Access Control

Not every user should have access to your network. To keep out potential attackers, you need to recognize each user and each device.

28.3.6   Two-factor Authentication

Two-factor authentication (also known as 2FA) is a type, or subset, of multi-factor authentication.

It is a method of confirming users' claimed identities by using a combination of two different factors: 1) something they know, 2) something they have, or 3) something they are.

A good example of two-factor authentication is the withdrawing of money from an ATM; only the correct combination of a bank card (something the user possesses) and a PIN (something the user knows) allows the transaction to be carried out.

All the large providers of internet services, like Facebook, Google, Apple, etc. offers Two-factor authentication, you just need to turn it on.

References:

https://en.wikipedia.org/wiki/Multi-factor_authentication

28.3.7   VPN

A virtual private network encrypts the connection from an endpoint to a network, often over the Internet.

28.3.8   Web Hosting Providers

Make sure to use professional Web hosting companies for your web sites.

You have large Cloud platform providers like Amazon Web Services (AWS), Microsoft Azure, Oracle Cloud and Google Cloud Platform.

These large providers have a high level of security, they continuously update the systems with new security patches, etc.

28.3.9   Wi-Fi Network

Use only secure Wi-Fi networks, not open Wi-Fi network that do not need password, etc.

28.3.10   Operating System

Make sure your PC and the operating system is up to date.

28.3.11   Education

Learning more about cyber security threats and how you can protect yourself as a private person or a company is crucial.

29 SQL Injection

A Structured Query Language (SQL) injection occurs when an attacker inserts malicious code into a server that uses SQL and forces the server to reveal information it normally would not.

An attacker could carry out a SQL injection simply by submitting malicious code into a vulnerable website search box.

SQL Injection - with Practical Examples using ASP.NET Core and C#:
https://youtu.be/AWHXQGaJXe

29.1 SQL Injection Examples

Below you find some basic SQL Injection examples.

Web Site:

```
<form action="/cgi-bin/login" method=post>
      Username: <input type=text name=username>
      Password: <input type=password name=password>
<input type=submit value=Login>
```

SQL query executed by the web site for getting the user information:

```
select * from Users where (username = 'submittedUser' and password =
'submittedPassword');
```

SQL Injection Example #1:

For example, if an application accepts and processes user-supplied data without any validation, an attacker could submit a maliciously crafted username and password. Consider the following string sent by an attacker:

```
username=admin%27%29+--+&password=+
```

The SQL query executed by the web site for getting the user information will be:

```
select * from Users where (username = 'admin') -- and password = '  ');
```

In this example, an attacker could successfully log in to the application using the admin account without knowledge of the password to that account.

Note that the string of two dash characters (--) indicates to the database server that the remaining characters in the SQL statement are a comment and should be ignored.

SQL Injection Example #2:

A hacker might get access to user names and passwords in a database by simply inserting " OR ""=" into the user name or password text box:

User Name:

```
" or ""="
```

Password:

```
" or ""="
```

The SQL query executed by the web site for getting the user information will be:

```
select * from Users where Name ="" or ""="" AND Pass ="" or ""=""
```

The SQL above is valid and will return all rows from the "Users" table, since OR ""="" is always TRUE.

References:

https://www.w3schools.com/sql/sql_injection.asp

29.2   Resources

Here are some other resources regarding SQL injections:

https://www.owasp.org/index.php/SQL_Injection

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

https://www.w3schools.com/sql/sql_injection.asp

30 User Identity and Login

Here we will give an overview of user identity and login features in modern applications.

The concepts will be exemplified using web technology and the ASP.NET Core web framework from Microsoft.

ASP.NET is an open-source web framework, created by Microsoft, for building web apps and services using the .NET Framework or the .NET Core. We have both ASP.NET and ASP.NET Core. ASP.NET Core is the new approach built on .NET Core.

In this chapter we will see how we can create and use login functionality in your ASP.NET Core Web Applications.

Typically, you need to create functionality for User Registration, Login, etc. Here you will see how this can be done from scratch. If you do it from scratch, you will have full control of your code.

If you use something called "ASP.NET Core Identity" (which will be explained and demonstrated in the next chapter) lots of "magic" happens behind the curtains. If something is not working, it may be more complicated to figure out why.

Below you will find some useful ASP.NET resources.

https://www.halvorsen.blog/documents/programming/web/aspnet

30.1 Password Security

Keeping your passwords safe is important and all software systems should take this seriously.

Password security mechanism:

- Encryption and Decrypting

- Hashing

- Salting

- 2 Factor Authentication

- Etc.

These password security mechanisms will be described in more detail below.

30.1.1   Encryption and Decrypting

Encryption is the practice of scrambling information in a way that only someone with a corresponding key can unscramble and read it.

Encryption is a two-way function. When you encrypt something, you are doing so with the intention of decrypting it later.

To encrypt data, you use an algorithm. Many different encryption algorithms do exist

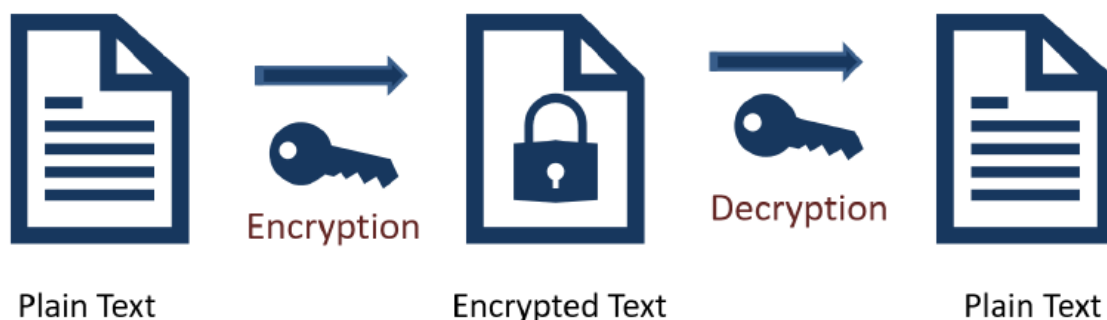Figure 30-1 gives an overview of the concepts of Encryption and Decryption.

Figure 30-1: Encryption and Decryption

When should encryption be used? Here are some examples:

- Encryption is a two-way function.
- You encrypt information with the intention of decrypting it later.
- Examples when to use encryption:
- Protecting Files and Information on your Computer
- Protecting your Cloud data
- Transmitting Data between 2 Computers
- Etc.

The key is that Encryption is reversible. Hashing is not.

30.1.2   Hashing

Hashing is the practice of using an algorithm to map data of any size to a fixed length. Encryption is a two-way function. Hashing is a one-way function.

While it is technically possible to reverse-hash something, the computing power required makes it unfeasible. Hashing is one-way. See Figure 30-2.

Encryption is meant to protect data in transit, hashing is meant to verify that a file or piece of data has not been altered—that it is authentic. In other words, it serves as a checksum. Every hash value is unique.
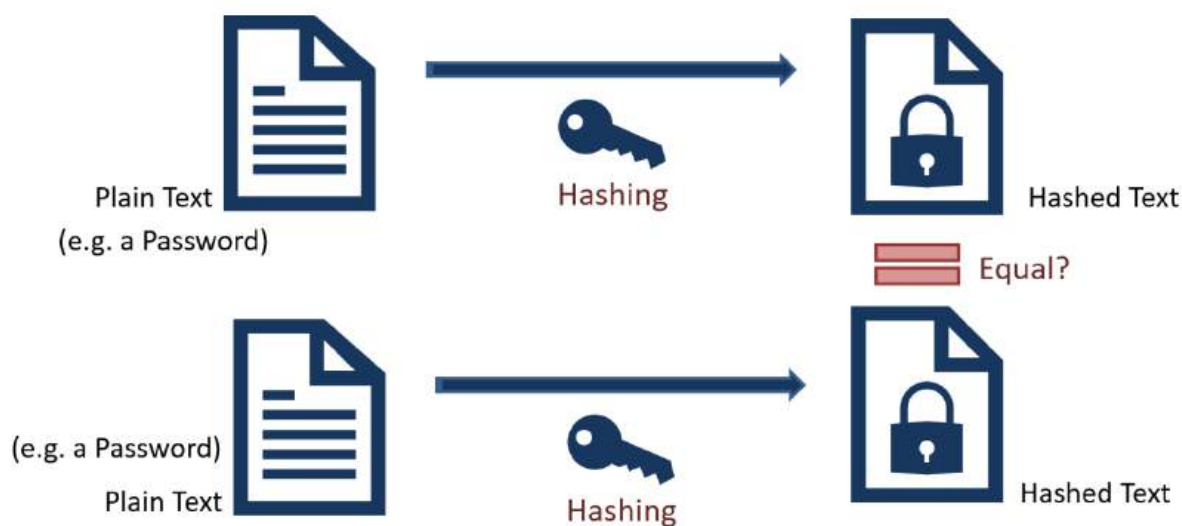

Figure 30-2: Hashing

30.1.3   Rainbow Tables

Is it possible to for a hacking to get access to Hashed Passwords?

By using something called "Rainbow Table" the hacker can get access to your hashed password, see Figure 30-3.

Password Table for System X

UserName	HashedPassword
Mike	4420d1918bbcf7
Bob	73fb51a0c9be7d
Peter	4420d1918bbcf7

Password	HashedPassword
tesla	4420d1918bbcf7
friendship	73fb51a0c9be7d
bicycle	7420e1618abcf6

Rainbow table

If a Hacker gets access to this Database, he can see that Mike and Peter have the same password.
But he does not know the actual password

If the Hacker has access to so-called "Rainbow table" (which is essentially a pre-computed database of hashes), he may also be able to find the Password (as seen here)

If you have a complicated password, it is less likely that your password is in such a Rainbow table

Figure 30-3: Using Rainbow Table for Hacking your Hashed Password

If a hacker gets access to this Database, he can see that Mike and Peter have the same password, but he does not know the actual password. If the Hacker has access to a so-called "Rainbow table" (which is essentially a pre-computed database of hashes), he may also be able to find the Password, as seen in Figure 30-3. If you have a complicated password, it is less likely that your password is in such a Rainbow table.

30.1.4   Salting

Salting is a technique typically used for Password Hashing. It is a unique value that can be added to the end of the password to create a different hash value. The additional value is referred to as a "salt". This is done to make it even more secure. Typically, the Hashing Algorithm uses a Random salt. This prevents an attacker from seeing whether users have the same password. See Figure 30-4.

Figure 30-4: Salting

Is it possible to hack "Hashing with Salt"?

Assume Mike and Peter use the same Password, see Table 30-1. If a hacker gets access to this database, he cannot see that Mike and Peter have the same password. This is because a random Salt has made these 2 Hashed Passwords different!

Table 30-1: Examples of Hashed Passwords with Salt

User Name	Hashed Password with Salt
Mike	4420d1918bbcf7

| Bob | 73fb51a0c9be7d |
| Peter | 4520d1818cbcf7 |

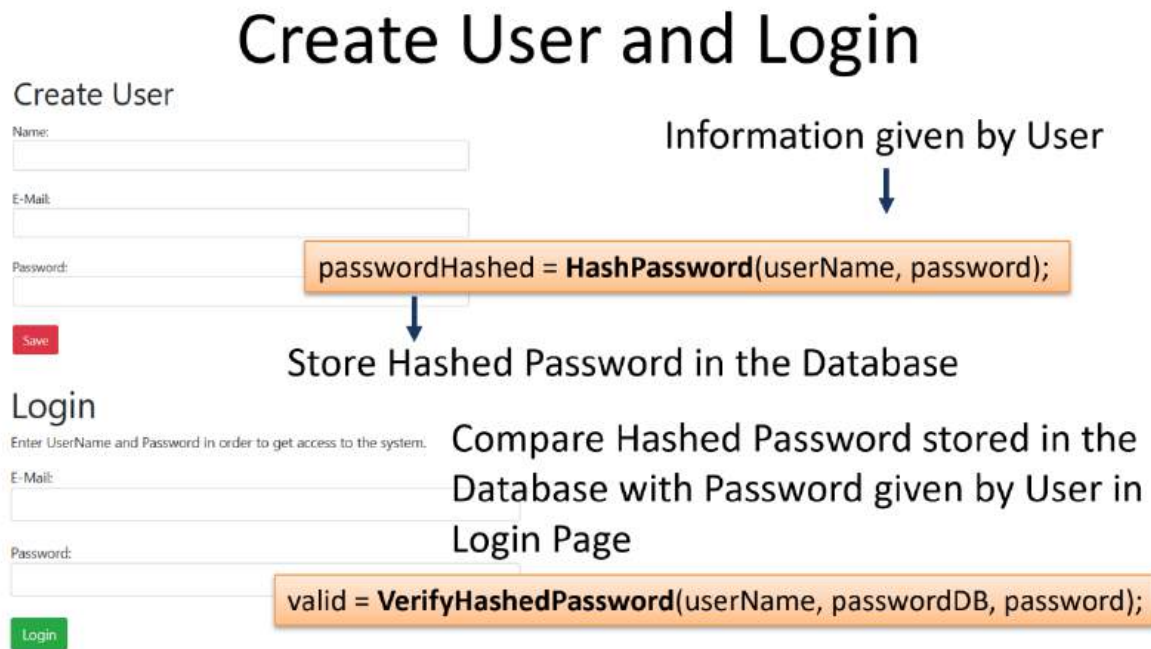Figure 30-5 shows a typical Flow when Creating User and Login.

Figure 30-5: Typical Flow when Creating User and Login

31   SQL Server Authentication

31.1   Introduction

SQL Server is a Database System from Microsoft. SQL Server comes in different editions, for basic, personal use

SQL Server Express is recommended because it is simple to use, and it is free.

Web:

https://www.halvorsen.blog/documents/technology/database/sql_server.php

Videos:

Introduction to SQL Server: https://youtu.be/SlR4KOhAG1U

SQL Server Express Installation: https://youtu.be/hhhggAlUYo8

SQL Server and Structured Query Language (SQL): https://youtu.be/sI6skicZse0

31.2   Authentication

SQL Server has 2 different types of authentications:

- Windows Authentication
- SQL Server Authentication

Using "Windows Authentication" the Connection String looks like this:

```
DATA SOURCE=<SQL Server Name>;DATABASE=<Database Name>;Integrated Security = True;
```

Using "SQL Server Authentication" the Connection String looks like this:

```
DATA SOURCE=<SQL Server Name>;DATABASE=<Database Name>;UID=sa;PWD=<Your Password>;
```

Replace <SQL Server Name> with the name of your SQL Server, typically "<YourComputerName>\SQLEXPRESS" if you are using SQL Server Express.

UID is a SQL Server user, here you can create your own SQL Server user inside SQL Server Management Studio or use the built-in sa user (sa=System Administrator). During the setup of SQL Server, you need to select "Mixed Mode" and enter the password for your sa user.

It may look something like this:

```
DATA SOURCE=HPPCWORK\\SQLEXPRESS;DATABASE=MEASUREMENTS;UID=sa;PWD=Password123;
```

You can also turn on "SQL Server Authentication" in SQL Server Management Studio (SSMS) after installation of SQL Server. See Figure 31-1.
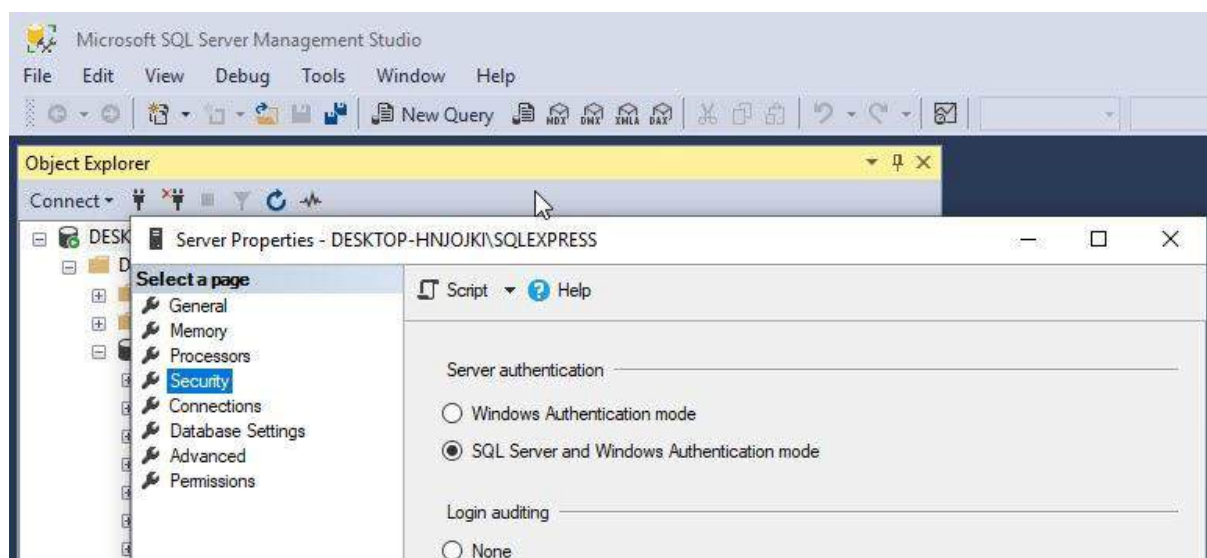

Figure 31-1: SQL Server Authentication

To change security authentication mode, do the following steps:

- In SQL Server Management Studio Object Explorer, right-click the server, and then click Properties.
- On the Security page, under Server authentication, select the new server authentication mode, and then click OK.
- In the SQL Server Management Studio dialog box, click OK to acknowledge the requirement to restart SQL Server.
- In Object Explorer, right-click your server, and then click Restart. If SQL Server Agent is running, it must also be restarted. Or just restart your computer.

31.3   Create Logins in SQL Server

"sa" (short for System Administrator) is a built-in Login in SQL Server. You can also create your own SQL Server Logins. Normally you should do that rather than using the "sa" login. "sa" have access to "everything" and in context of Data Security that is unfortunate. In general, you should make your own Logins that have access to only what is strictly necessary. See Figure 31-2.
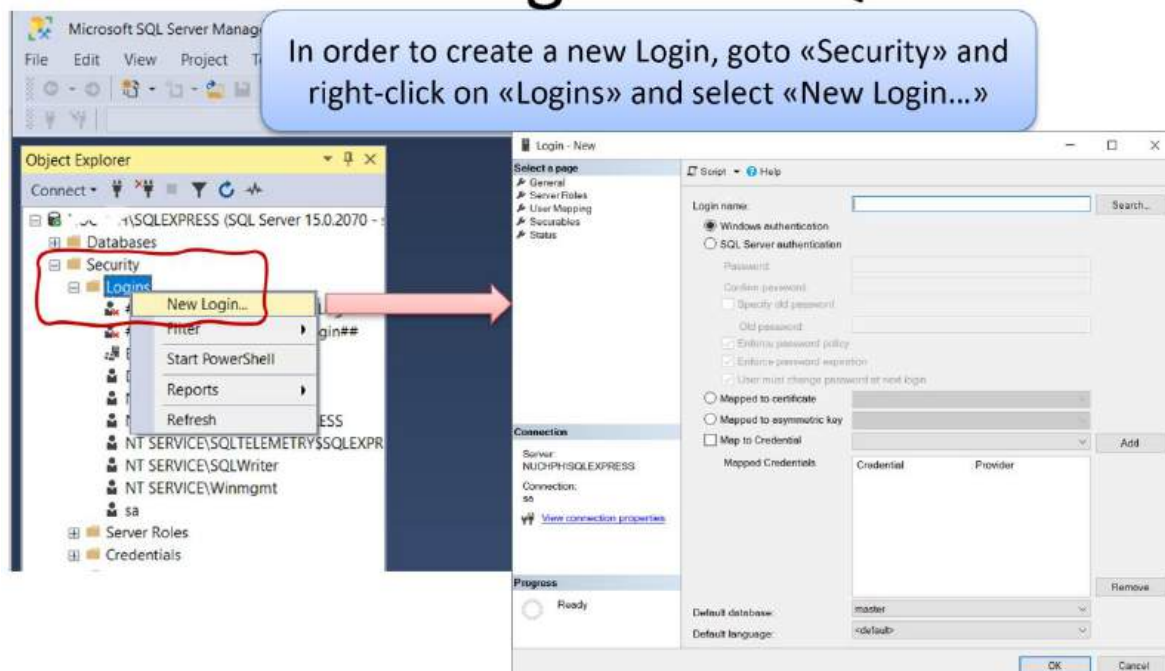
Figure 31-2: Create Logins in SQL Server
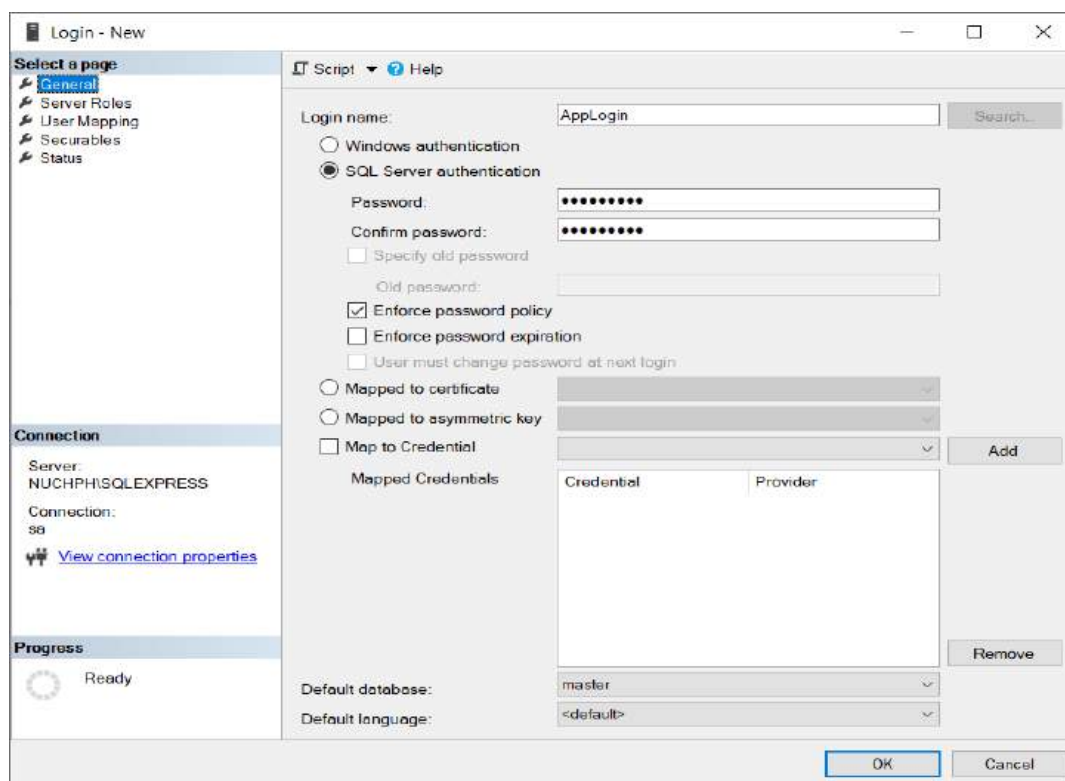
We can select login type, see Figure 31-3.

Figure 31-3: New Login

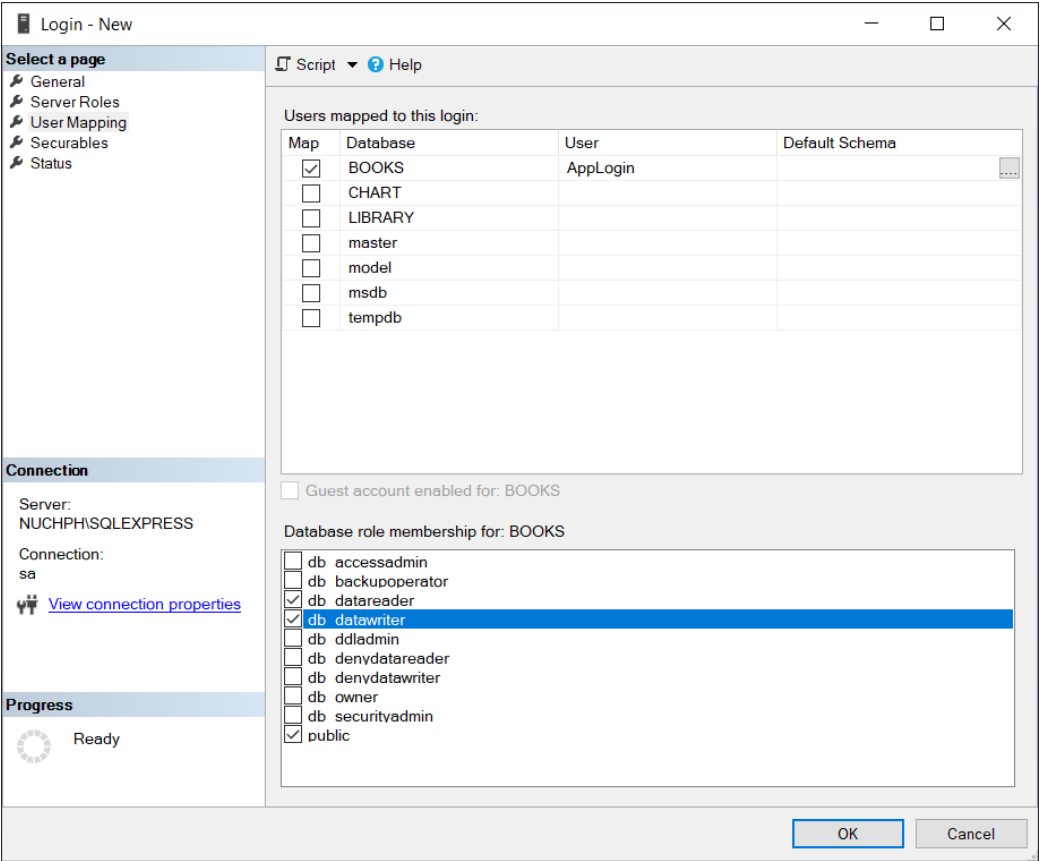Select access levels, see Figure 31-4.

Figure 31-4: Login - Select Access Level

Part 6 : Additional Resources

In this part, additional resources, references, and appendices are available.

32 Glossary

Below we will give a short overview of some important topics in software development.

Term	Description
Agile Development	Agile software development is a group of software development methods based on iterative and incremental development.
Agile Manifesto	The philosophy behind Agile methods is reflected in the Agile Manifesto.
ALM	Application Lifecycle Management. An ALM system takes care of all aspects in software development from planning, requirements, coding, testing, deployment, and maintenance. Azure DevOps is an example of an ALM system.
API	Application Programming Interface. API specifies how some software components should interact with each other. In practice in most of the cases an API is a library of different classes, methods/functions, etc. that a developer can use.
Burn Down Chart	A burn down chart is a graphical representation of work left to do versus time. The outstanding work (or backlog) is often on the vertical axis, with time along the horizontal. That is, it is a run chart of outstanding work. It is useful for predicting when all the work will be completed. It is often used in agile software development methodologies such as Scrum. However, burn down charts can be applied to any project containing measurable progress over time.
Code Freeze/Feature Freeze	No changes in the code can be made, except bugs.
Code Review	Code inspections to check if code is properly written, and useful to avoid bugs in your code.
Daily Scrum	In Scrum Development they have daily 15 minutes meetings within the Software Team. The Meeting is usually a Stand-up meeting.
DDT	Development-Driven Testing

Dog-fooding	Computer software company uses its own product to demonstrate the quality and capabilities of the product.
GUI	Graphical User Interface. The part of the software the user sees and interact with.
HA	High Availability. Some software needs to run 24-7.
HMI	Human Machine Interface. Another term for Graphical User Interface.
IDE	Integrated Development Environment. A software application that helps developers to create, edit, compile, and run their code.
Internal Server Error	Typical errors from web pages. The Web server (running the Web Site) encountered an unexpected condition that prevented it from fulfilling the request by the client.
Pair Programming	2 developers working together.
QA	Quality Assurance. QA refers to the engineering activities implemented in a quality system so that requirements for a product or service will be fulfilled.
Refactoring	Code refactoring is used in software development to improve existing code without changing its functionality. The goal is to make the code more robust and easier to maintain.
SCC	Source Code Control or Version Control. A version control system keeps track of all work and all changes in a set of files. Allows several developers (potentially widely separated in space and time) to collaborate and share code.
SDD	Software Design Document. A document describing the design of a software application.
SDK	Software Development Kit. A SDK is typically a set of software development tools that allows for the creation of applications for a certain software package, software framework, hardware platform, computer system, operating system, or similar development platform.
SDLC	Software Development Life Cycle. The process of creating software.
Software Development Process	Waterfall, Agile, etc.

Software Engineering	The discipline for creating software applications. A systematic approach to the design, development, testing, and maintenance of software.
SRS	Software Requirements Specifications. A document stating what at application must accomplish.
STD	Software Test Documentation. Contents: Introduction, Test Plan, Test Design, Test Cases, Test procedures, Test Log, …, Summary.
STP	Software Test Plan. Documentation stating what parts of an application will be tested, and the schedule of when the testing is to be performed.
TDD	Test-driven development (TDD) is a software development process that relies on the repetition of a very short development cycle: first the developer writes an (initially failing) test case that defines a desired improvement or new function, then produces the minimum amount of code to pass that test, and finally refactors the new code to acceptable standards.
Azure DevOps	Azure DevOps is a Source Code Control (SCC), Bug Tracking, Project Management, and Team Collaboration platform from Microsoft
UML	Unified Modeling Language. A Language used in Software modeling.
XP	eXtreme Programming. XP is an Agile Software Development method. It's based on Unit Testing, Code Reviews and Pair Programming.

References

[1]   I. Sommerville, *Software Engineering*, 10 ed.: Pearson, 2016.

[2]   Wikipedia. (2017). *Software Requirements Specification*. Available:
        http://en.wikipedia.org/wiki/Software_Requirements_Specification

[3]   Wikipedia. (2017). *Software Development Process*. Available:
        http://en.wikipedia.org/wiki/Software_process

[4]   Wikipedia. (2017). *Waterfall Model*. Available: http://en.wikipedia.org/wiki/Waterfall_model

[5]   Wikipedia. (2017). *V-Model (Software Development)*. Available: http://en.wikipedia.org/wiki/V-Model_(software_development)

[6]   Wikipedia. (2017). *Agile Software Development*. Available:
        http://en.wikipedia.org/wiki/Agile_software_development

[7]   Agile. (2017). *Agile Manifesto*. Available: http://agilemanifesto.org

[8]   Wikipedia. (2017). *Pair Programming*. Available:
        http://en.wikipedia.org/wiki/Pair_programming

[9]   Wikipedia. (2017). *Scrum Development*. Available:
        http://en.wikipedia.org/wiki/Scrum_(development)

[10]   Wikipedia, "Unified Process," 2017.

[11]   K. Schwaber and J. Sutherland. 2011, The Scrum Guide. Available: scrum.org

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

[13]   Microsoft. (2017). *Windows 8 UX Guidelines*. Available: http://msdn.microsoft.com/en-US/library/windows/apps/hh465424

[14]   Apple. (2073). *Mac OS X UX Guidelines*. Available: https://developer.apple.com/library/mac/-documentation/userexperience/conceptual/applehiguidelines/UEGuidelines/UEGuidelines.html

[15]   Wikipedia. (2017). *GUI Mockups*. Available: http://en.wikipedia.org/wiki/Mockup

[16]   F. Tsui, O. Karam, and B. Bernal, *Essentials of Software Engineering*, 3 ed.: Jones & Barlett Learning, 2014.

[17]   H.-P. Halvorsen. (2017). *ASP.NET and Web Programming*. Available: https://www.halvorsen.blog

[18]   H.-P. Halvorsen. (2017). *Introduction to Visual Studio and C#*. Available:
        https://www.halvorsen.blog

[19]   H.-P. Halvorsen. (2017). *Structured Query Language*. Available: https://www.halvorsen.blog

[20]   H.-P. Halvorsen. (2017). *So You Think You Can MATLAB?* Available: https://www.halvorsen.blog

[21]   H.-P. Halvorsen. (2017). *LabVIEW Resources*. Available: https://www.halvorsen.blog

[22]   L. Malka. (2013). *API*. Available: http://www.lior.ca/publications/api_design.pdf

[23]   H.-P. Halvorsen. (2017). *Introduction to Web Services*. Available: https://www.halvorsen.blog

[24]   E. Blankenship, M. Woodward, G. Holliday, and B. Keller, *Professional Team Foundation Server 2012*: Wiley, 2013.

[25]   H.-P. Halvorsen. (2017). *Introduction to Database Systems*. Available: https://www.halvorsen.blog

Hans-Petter Halvorsen

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

Web: https://halvorsen.blog/

https://halvorsen.blog/
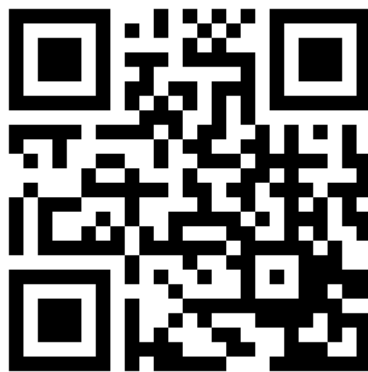
Software Development

A Practical Approach!

Hans-Petter Halvorsen

Copyright ©

Software Development