# Overview

Characteristics of IT Solutions

IT solutions live long: App lives longer than hardware, longer than system software => separation of application and technical environment

Complexity not through difficult algorithms but through a variety of (simple) combinations **Net Code**=Lines of code that solve the problem

**Gross Code**= Net Code +Error handling for data+ Error handling for system software + Help functionality+ Validation of user rights+ Logging+ Message exchange+ …

———————————————————————————————————————————

# Mobile Solution

**Consumerization** describes the change of how new technologies are introduced into the IT environment of a company.

Formerly: New technologies has been introduced in companies first and later on in the consumer sector.

Nowadays: New technologies will be introduced in the consumer sector first and later on in companies

**differences** **btw mobile clients**: Operating system, Display,Battery life, Processor, RAM and flash memory, Interfaces for data communication, Keyboard, Fall protection,IP protection class, Type of data collection, Dimensions and weight, Operating temperature, Pricing

**Development**: Cross platform(Similar lang. same IDE, Single code base, Saving time and $)

**Apache CORDOVA**(PhoneGap[distribution of Apache Cordova]):engine that powers PhoneGap, similar to how WebKit is the engine that powers Chrome or Safari

+A lot of supported platforms +Nearly only web skills necessary -Hardware access is slow -Not all features could be used on some devices -Risk of exclusion from apple store

**Xamarin** +Native speed an functionality +Native UI +Up to 100& shared code with xamarin.forms

- $999 / year and developer and platform -Bigger apps (Mono) -Still need of knowledge in different OS -Risk of exclusion from apple store

**CORDOVA**: Content-Driven / Design, Less or no hardware support, Almost every platform, Rapid Prototype / web site as an app

**VS. XAMARIN**: Performance / function, Hardware and platform support, The most important platforms, Software development

**Major apps for mobile device**:

**1-Data collection**:(RFID, barcode, Manual)

*Optimization*: No redundant data entry, No media breaks, No transmission errors , Actual data basis, Increasing efficiency, Savings

*Automation*: Automation of downstreamed processes, Interface issues must be considered

*Acceleration*: Quick availability of data, Overall Process acceleration

*Quality*: Plausibility check directly at the site of data generation, Depending on the detection method, intrinsic testing of data (such as checksums), Interactive guidance is possible, No downstream scanning and OCR processes necessary

**2-Data representation**

Characteristics of business information systems —— Goals of architecture

large and complex: Find manageable sub-structures, Development with multiple teams, Development in stages

long lifetime: Extremely clear,System must be changeable, Maintainability

constant change: Recognizable solutions, Uniform consistent solutions, Rules&conventions

build to run: Consider infrastructure, Replacement / upgrade of sub-systems possible (Middleware, hardware, data base)

**Software categories**: (Separation of concerns)

**0-Software**:Independent of application and technology; Ideally reusable;

**A-Software**:Determined by the professional application; Independent of technique;  
Usually the largest part of the system;

**T-Software**:Independent of the professional application; Expert for a technical component - reusable

**R-Software**:Pure transformation (representation); tolerable mixture of A and T  
**AT-Software**: Dealing with mixed techniques and applications; Basically to avoid: hard to maintain; opposes changes; Reuse nearly impossible!

**Component tailoring**: define the responsibilities and to minimize dependencies

Goals——Guidance

*Separation of technical services*: Separation of application logic (A) and technology (T), Application core (A software) is technology independent, Encapsulate technical dependencies

*Separation of concerns*: More than one concern is bad, Avoid distributed concerns, Document concerns, Components belong to exactly one software category

*Parallel development:* can be developed separately, Minimize the interfaces between components

*Encapsulation of neighbour systems*: Encapsulate neighbour systems, Preferably no close coupling

*Usability challenges*: Small Screen, No wholesome keyboard, Different operating concept (e.g., right click), Perhaps other style guides, On the way (in the car, noises,sun),Intuitive

*Prototyping*: Quick, Easy, Cheap, Changeable +Install ad hoc changes in the meeting, Executable, Interactive

**What are prototyping tools able to achieve?** Development in team,Selection of ready-made templates, Interactive operability of the prototype

**And what not?**They do not replace a concept or a design • No technological stick through, No source code=>Discard it!!!

1.Paper or 2.software Or 3.Coding?

1.Quick and intuitive,Existing knowledge, No restrictions Collaborative, Dynamic addition by post-it Stencils

2.Construction kit, Reusing items by copy & paste, Digitized, Could be e-mailed, Desktop sharing  
Click dummy

3.Implementation in target technology, All the capabilities of the device, Live demonstration  
(… but expensive and slow), and architecture trap

**Analog prototype**: Only writing materials required , Easy to learn , Changeable every time without further equipment , Stencils and templates for standard elements , Collaboration and team work with pinboard or whiteboard

**Storyboard**: only gather requirements — At least: understanding the requirements by watching closely the users — *Even better:* challenge the behaviour if the user (ask every time „Why?“) —Ideally*:* taking the role of the user and walk through the use cases yourself

———————————————————————————————————————————

# Profitability of IT

IT related benefits/ Cost of IT

**IT Investments**: IT portfolios and IT projects are often huge investment for companies. Benefits are not clear, costs are high. Limited IT budgets require objective prioritization of investments.

**IT Organisation**: Profitability is a central topic of each company. IT organization is seen as cost driver. In contrast to ‟classical‟ business functions IT is often a ‟black box‟ for decision makers – relation to profitability is not clear, benefits are often neglected.   
Goal: Measurable, complete and sustainable criteria as baseline for managerial decisions. Ongoing control of target achievements  
**IT Investment**:

*Planning*: Case study / Business case,Concept / requirements analysis, Make or Buy decision, Business concept / technical concept, design

*Implementation*: Implementation / Customizing, Test / Integration, Hardware, Software, SLA design, Go-Live / Roll out, Change Management

*Operations*: Adaptions,Maintenance, Means of production, Enhancements, Deployment of releases

*Replacement*: Removal, Backup, Data migration

Cost drivers often neglected

Distribution of costs:Implementation 20-40%, Operation 60-80%

TCO: Total cost of ownership

Net Benefit: Accounting all cashflow ( Cost, benefit) during application lifecycle

A **Business Case** summarizes all crucial aspects of an intended investment with the objective of: identify economical advantages and strategic conformity and enabling final management decision regarding execution. (runs in parallel to all phases of a project) Focus on initial analysis of profitability during analysis an initialization of the IT project. longterm achievement of the objectives is measured by the controlling function.

**classification**: a BC is either a separate project or part of a pre-study.a BC has its own objectives, planning (milestones and deliverables) and is staffed with a project team. finally a BC will provide results – if required as part of the project’s pre-study: reproducible documentation and management summary.   
**Frame conditions:** a sponsor for the BC is required (‟Probably the sponsor of the main project?”).

a stake holder analysis shall be done (‟What will support the project?‟, ‟Do we expect any opposition?‟). company rules have to be considered (e.g. involvement of controlling department).

**1-Definition of goals and objectives & 2-Planning**: Scope:  (goals,methods, dependencies);

Goals+Planning of BC & IT project should be aligned; a skilled team is available

**3-definition of scenarios**: a typical BC examines and compares several scenarios:

−  *baseline scenario*: expected evolution without change („Doing nothing‟).

−  *solution scenario(s)*: expected evolution when the planned investment is done.

Describe the delta between solution and baseline scenario by terms like increasing / decreasing costs or improved benefits

alternative solutions may exist (e.g. solution based on custom or package software).

variants of a solution scenario may exist (e.g. different assumptions on further development of turnover).

Number of scenarios / variants low to reduce effort to be spent for the BC analysis

**4-Analysis**: analysis is main part of the BC and contains:−  cost analysis −  benefit analysis − risk assessment   
steps for cost and benefit analysis: identify cost / benefit „items‟ and assign them to typical cost / benefit categories; check if an item is relevant for profitability calculation; quantify item; valuate item monetary

*Risk* assessment is focused on risks that may impact profitability

**Dimensions of cost analysis**: Cost Categories(Investment cost[Mainly IT costs for make or buy of IT applications], operation cost, process cost[related to personnel costs, e.g. caused by re-organisation or process changes], increase of floating capital:[outstanding debits, inventory, liquid assets]), Origin of Costs, Internal/External costs.

there is no explicit correlation between cost items and cost categories

cost items can occur in more than one category (e.g. SW license costs)

important: differentiate between singular an repeated cost items

regularly singular costs before go-live will be accounted as investment.

maintenance costs and process costs are often repeated costs. They have to be considered normally once per year

**Typical benefits categories**: increasing turnover,increasing earning, higher productivity,reduction of floating capital, reduction of costs (comparison of solution and baseline scenario).

**Quantification of benefits**: have to separate between   
• benefits that can be expressed monetary *directly* ,e.g. reduction of costs   
• benefits that can be expressed monetary *indirectly*, e.g. higher turnover due to higher customer satisfaction   
• Benefits that *cannot be expressed* monetary e.g. better image of a company

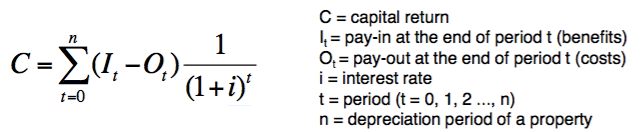
BC should concentrate on main benefits

**5-Calculation of profitability**: mathematical methods to find ratio of benefits and costs. it provides key indicators to compare the planned IT investment with a (imaginary) financial investment. Pc is the most important result of a BC analysis and will provide the ‟hard facts‟ for a decision. Pcs have to be done for each scenario.

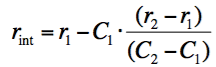
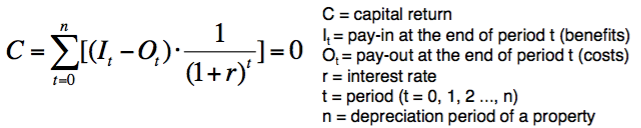
**Methods of PC:**

Static methods: Based on simple assumptions, e.g. average values for investments and spending / earning, Not much relevant for IT projects

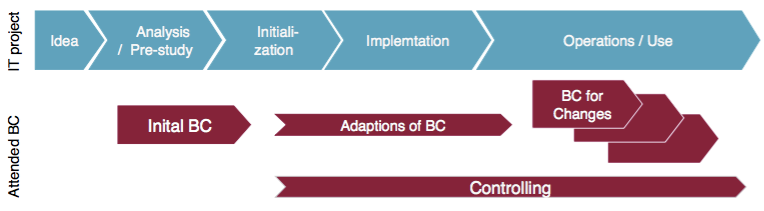
Dynamic methods:imputed interests are considered, e.g. (net present value method, internal rate of return method, methods to consider risks)

**1)Net present value method**: Will the earning expected for the IT investment be larger than capital return based on a given interest rate?

Future cash-flow will be discounted to date of investment. Motivation: Capital available today is more valuable than future capital. Interest rate *i* defines capital return. Motivation: Capital can be invested alternatively on capital market with interest of i%.

Internal rate of return method: At which interest rate will capital return become zero?

Two interest rates r1 and r2 have to defined, where the related capital return C1 and C2 will be positive and negative respectively.Internal interest rate can only be calculated approximately. The internal interest rate is the required interest rate for a similar investment on capital market.

**6-Documentation:** overall documentation of results containing: clear and comprehensive description of scenarios, well benefit / cost items and quantification of these items.−  applied methods for calculation.−  All hard and soft facts that are relevant for the investment. in real live most decisions will not only be based on key figures but also on soft facts.

**1-Idea(Vision)**: TO-DO in project: define general goals, align project scope with company’s business strategy, detail underlying business ideas

TO-DO in BC: collect cost drivers and benefits, analyze stakeholders

**2-Analysis(Vision)**: TO-DO in project: detail goals, analyze gaps, perform feasibility study, define alternative solutions, provide decision paper

TO-DO in BC: identify categories of costs and benefits, rate and quantify categories of costs and benefits, analyze alternative scenarios vs. Reference scenario, perform profitability calculation   
**3-Initialization**: determine and prioritize decision criteria,provide requirements and tender,provide functional specification,vendor selection,conclude contract,initialize project

rate decision criteria economically,detail out solution scenarios

**4-Implementation:** implement project based on the selected methodology; apply principles of project management; adapt BC if required; provide economic measures to support decisions on changes of frame conditions (budget, change requests etc.); start controlling

**5-Operation/Use**: operate and maintain the application,implementation of new requirements and enhancements

adapt BC if required, provide economic measures to support decisions on enhancements and new functionality, perform controlling based on given indexes / figures, provide measure for improvements and optimisation.

———————————————————————————————————————————

# Effort and cost estimate

Estimation based on real experience: Quantify effort, Calibrate the experience&estimate again, perform task.

Expert estimation: experience of expert. expert: has executed task at least 3 times.

small project (9months): P1, P2, P1, P3, P1 => 3.75 years

Large project (3Years): P1, P2, P1, P3, P1=>15 years

Estimation DB: solve the limits of intuition in large scale projects. Standardization required for(size (FSM functional size management), complexity, environmental factors)

**Cost Model**: defines an obligatory structure for effort estimates, effort documentation and recalculation. structure is defined by abstract task categories covering any task.defines a common language. task categories define both the effort categories and furthermore the account system for booking of project efforts.

build the preconditions for: making projects comparable[it is a precondition for systematical learnings and achieving empirical metrics and estimation ratios.], enhance QA and proof of completion for estimates and recalculations.

**Tools of cost model**

Consistent effort categories and the corresponding account system.

Effort estimation template for documentation of the cost estimation.

Post calculation template along account systems to record actual effort by project closing

Estimation ratios based on empirical effort data to validate estimates.

Fixed price charge: Agio for guaranteed fixed price to account for business risks (false assumptions, contractual penalties, forgotten contract demands by estimation

Warranty charge: Agio for warranty claims after acceptance (e.g. bug fixing)

Net effort: immediate effort for development of project artefacts (PI), without cross functional effort (PQ) or indirect project efforts (PN)

Gross effort:Total effort of project delivery:without fixed price charge,without warranty charge=PI(Specification, realisation, …)+PQ(project management)+PN(Job training)

**Estimation methods:**

**Top-down**: Total estimation of project effort by mathematical algorithms based on functional requirements(used for validation of bottom-up estimates): estimation by formula, in general empirically proofed, based on measurable features, i.e. LoC, requirements or specifications, partial costly but good results

algorithmic methods(COCOMO Function Points Use Case Point): Links to development projects realized in the past, No measurable quantities such as product LoC needed, Necessary recalculations of completed projects, comparison methods(analogy method)

ratio methods (multiplier methods percentage method): Similar to analogy method, but you need data from completed projects

**Buttom-Up**: Effort of each task of the project is calculated separately and summarized to gather total project effort.

estimation by experts(single estimate[A single expert determines the estimated values for a particular task],Delphi method[Several experts conduct their estimate anonymously and separately], estimation workshop[Several experts estimate in a joint estimation workshop ]):if possible draw on to analogy method, first-time estimate of new requirements by expertise

**Steps to create a cost estimation:**

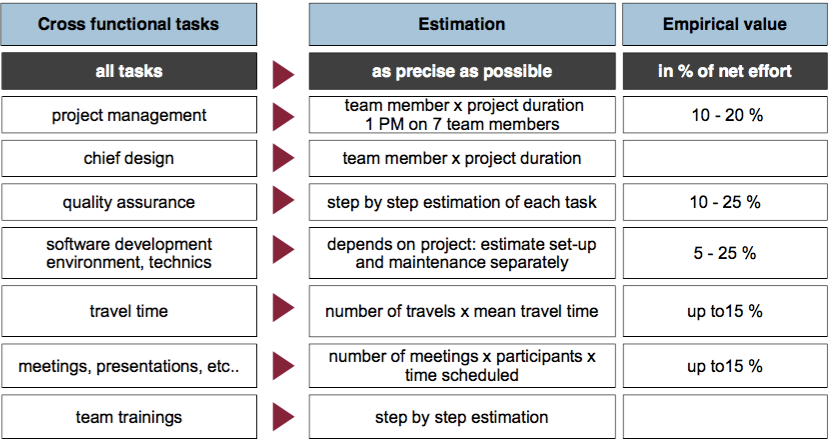
1-work breakdown structure (wbs) list of all necessary tasks, estimate each task =>net effort

wbs-items are not necessarily consistent with the later planning units

result: functional specifications, screen design, system documentation

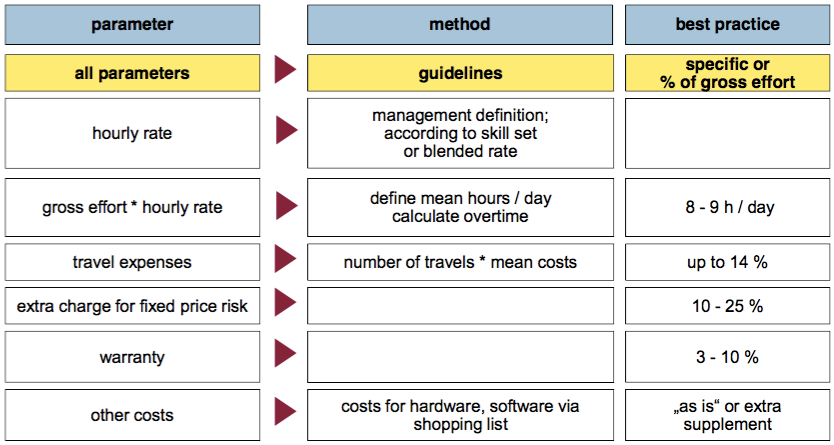
other activities: perform review, project management, meeting, Kick-off-workshop

total effort := estimate + estimation risk

2-cross-functional efforts as empirical values (percent values) => gross effort

3-Rating by calculated hourly rates+ fixed price charge + warranty charge => total budget

4-Validate by:(project plan and staffing curve, project phase ratios, comparison to similar projects) =>feasible budget

5-Target-performance comparison =>budget projection

Optimal team size: Radical(effort in PM)

Summary:

Tangibility/Traceability: Estimate step by step as many effort items as possible; avoid percentage markups

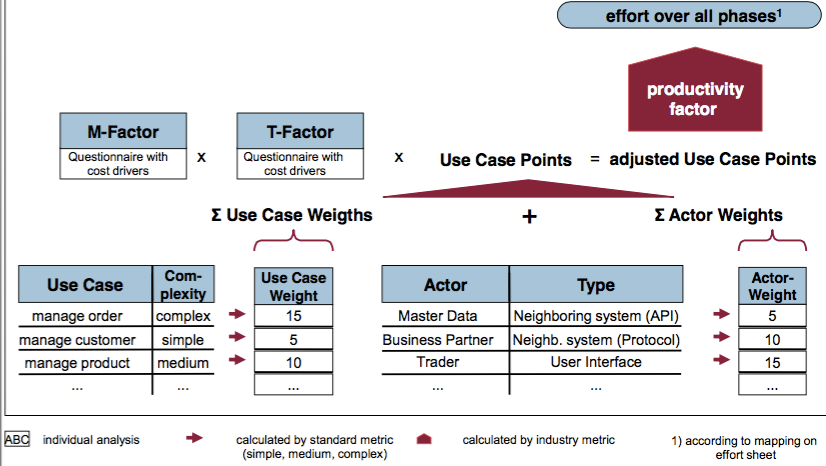
Estimation uncertainty: Record the estimation uncertainty for all items. Note that for each item only one calculated result is used for later project planning and calculation.

Effort estimation template: The result of the estimation is documented in the so-called effort estimation template.

Completeness: effort estimation template ensures the completeness and plausibility of the figures.

Premises: limits are reached (because sthng is not specified, 100% correct, unclear, forgotten).  
In this case it is necessary to formulate assumptions and to integrate those into the bid.

**Top-Down**: Assumption: comparability of project efforts for the same functional scope.Functional size of the requirements is measured in “points”

**Use Case Point (UCP):** directly builds on a use case-based specification and is very easy to use

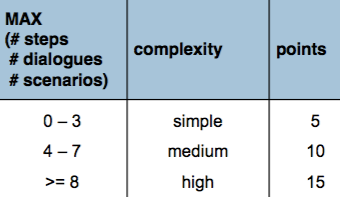
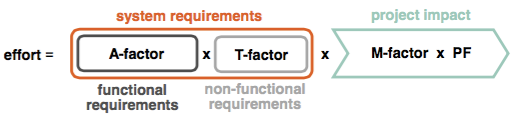
Precondition: rough specification is available

1-Classify Use cases: Use case: describe the behavior and interaction of a system in response to the distinct request or action of an actor (human or technical user of the system).

2-Classify actors: Use cases and actors define the functional respectively business scope of the project (= A-factor). This is recorded as Use Case Points

3-Determine T-Factor: The T-Factor takes into account the non-functional requirements and technological constraints of the project. It is determined by a questionnaire.

4-Determine M-Factor: The M-factor takes into account the organizational complexity of the project and the environment. It is determined by a questionnaire.

5-Calculate effort: The total project effort is calculated with the help of the productivity factor (PF). This factor was determined in the recalculations and is predetermined. The effort includes both business and technical components and is proportional to the identified use case points.

**A step in a use case**: a self-contained business part of the use case, which is:

clearly separated from the next and previous step(eg. by the change of the actor) or the processing “layer" (e.g. input dialog by the user-> processing input on the server-> display result)

generating a defined (intermediate) result (i.e. generating prints)

splitting up a new scenario

A distinct step is counted only once even it is included in several scenarios.

Typical examples of steps: (Enter one or more values in a dialogue (without an intervening server round trip), Call of application functions, Server transactions)

**UCP for**: custom software development,building new applications,implementation of new business processes, master data maintenance system

**UCP NOT for**: product customizing, maintenance, i.e. slight adaptation of running systems,version upgrades, control systems

method is inappropriate if scope of system adjustments is poorly described by use cases(where the functional size (A-factor) varies only slightly)

———————————————————————————————————————————

# Client technologies

interfaces with the user

**Characteristics**: (Presents information to user on a screen,Provides controls for interaction, Provides data validation feedback, Integrates with backend services for logic, Provides access to remote storage for information)

**Devices characteristics**: (OS, Screen size, input method, extras[GPS, scanner, …])

Evaluating client techs. : Systematic analysis (e.g. ISO 250xx)

User centric: quality in use model(Centred on perception of the software by the end user)

Effectiveness, Efficiency, Satisfaction, Freedom of risk, context coverage

Product centric: product quality model

Functional sustainability, performance efficiency, Compatibility, usability, reliability, security, maintainability, portability

**Native app**: can implement all functionality on the client (Visualization, presentation logic, business login, data layer). can use all functionality of OS

functional completeness: covers all the specified tasks and user objectives [Functional sustainability]

**user interface aesthetics**: pleasing and satisfying interaction [usability]

**time behaviour**: response and processing times [performance efficiency]

**Learnability**:user learning to use the product

Downsides: Installability depends on local (admin) rights, Installations needs to be managed by IT operations, Regular updates need to be rolled out, Low portability to another operating system, Technology for front end development might not match technology for back end development (.NET / Objective C / Java)

**Portable Applications** run on more than on Operating System

portability: can be transferred from one hardware, software to another

reusability: asset can be used in more than one system

maintainability:effectiveness and efficiency system can be modified

Desktop: JAVA/Eclipse RCP: Look and Feel is different from a native application, Access to some native features like tray icons, Interaction with native application possible, i.e. using OLE

Mobile: Phone gap, Apache Cordova: Use HTML + JavaScript

Look and Feel is (slightly) different from a native application (you can try to adopt you layout via CSS) , Access to common features of devices, Slower than a native application

**Thin client**: Using a browser with server side logic

installability: effectiveness and efficiency [to be] installed/uninstalled [portability]

replaceability: new version ... different product

operability:easy to operate and control

maintainability: effectiveness and efficiency ... system can be modified …

HTML based server side frameworks: MVC Style (Spring web MVC, ASP.NET MVS), Component Style (JSF, ASP.NET) [Minimal use of JavaScript, Using common browser standards, Building abstractions on top of HTTP and HTML, No access to OS specific functionality, Each user interaction requires feedback from the server]

Rich internet applications (RIA): AJAX w/ component style(RichFaces, IceFaces, PrimeFaces ), Component Based(Google Web Toolkit (GWT) ), Browser Plugins(Adobe Flash Microsoft Silverlight) [Heavy use of JavaScript (or browser plugin), Building lot’s of abstractions on top of HTTP and HTML (GWT cross compiles Java to JavaScript), Some user interaction requires feedback from the server]

Downside: Reduced user experience due to less interactive technology

Server side state requires resources per client connected to the system

Server side state needs to be shared between nodes in a cluster Scalability becomes difficult for a high number of users

Functional completeness problematic when too many levels of abstraction exist

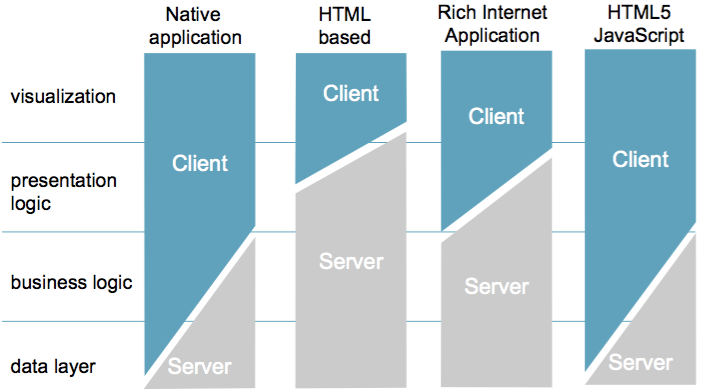
Browser plugins need to be installed separately

Browser plugins are famous for security problems

HTML5 + JavaScript as client technology: All features of a browser technology plus: [Maintainability, portability]

Capacity:maximum limits of a product …

resource utilization: resource used to meet requirements [Performance efficiency]

functional completeness: covers all the specified tasks and user objective [Functional Sustainability]

**Evolution of Client Technology**

maturity: applied to reliability, but also other quality characteristics

maintainability: corrections, improvements, adoption of the software to changes in the environment

adaptability: can be adapted for different or evolving hardware, software ...

**Interoperability of Clients:** can exchange information and use the information

Compatibility: can exchange information ... sharing the same ... environment

Functional completeness: covers all specified tasks and user objectives ...

Interoperability by Operating System and Client Technology

Android: Intents: publish/subscribe mechanism to receive data and events

Windows: OLE: specific interactions with applications, Contracts (since Windows 8): publish/subscribe mechanism to receive data and events

HTML / HTML5: Links: to forward to another app, optionally passing data JavaScript/DOM mash up: “pull” functionality into an app at runtime

Eclipse RCP / .NET WPF: Plugins that can be installed at runtime

**Testing Client Technology**

functional correctness: correct results with the needed degree of precision

Usability: achieve specified goals

maintainability:system can be modified by the intended maintainers

testability: test criteria can be established for the system; tests can be performed for [the] criteria

**Elements of automatic testing:** Continuous Integration Build; Unit Testing; Automatic Provisioning; Smoke Tests; Integration Tests  
Graphical User Interface (GUI) Tests; Load Tests;

Continuous Integration is available for all technologies

Automatic Provisioning is the next challenge  
GUI tests are difficult and brittle

**Difficulties of GUI testing**

.NET / WPF : One OS (few different versions) ,Can be virtualized on very few virtual machines (VMs)   
Eclipse RCP : Several OS (few different versions) , Can be virtualized on few VMs   
Web Client / HTML5 :Several OS, many browsers ,Can be virtualized utilizing multiple VMs   
Android: Many devices, many screen sizes, many versions, Can be virtualized – but what about customized Android?

iOS :Few devices, few screen sizes, few versions Difficult to virtualize

**Choosing Client Technology**

Look at the requirements and prioritize scenarios

Estimate the life time of the application   
Double check if the life time of the client is different from the server   
Evaluate using a catalogue like ISO 250xx   
Find out what criteria are relevant in the given situation   
Talk to people who have used the technology   
Build a prototype   
Don’t forget test automation   
Write down why you have chosen the client technology \*\*   
(Maybe) prepare the client technology to be exchangeable

Sweet Spot for a new project

For a new project a technology stack might look like this:

Choose HTML5/JS as client side technology(but don’t get lost in too many Java Script frameworks)   
Rendering HTML5 on the server and use HTML links to integrate with other components to keep things simple   
Keep the logic and storage of information on the server (probably with a REST service API)   
Consider a mobile first approach(would that affect only the layout, or would it require offline capabilities as well?)

———————————————————————————————————————————

# Automated Testing

**Constructive quality measures**: Development techniques+Guidance based on experience and planning=> it ensure specific typical error do not occur. => higher software quality

Technical measures: methods(e.g. use structure approach, define intermediate products like models, graphs, documents), languages, tools

Organizational measures: guidelines, standard, checklist

**Analytic quality measures**: Diagnostic measures, do not directly ensure high quality, measurement of quality of intermediate and final products => execute data to compare the actual state of software with desired target state. (identify quality of software after development)

Static checks(review documents): (code) inspection&review, static code analysis, Formal verification, symbolic execution

Dynamic checks: 1-Tests: blackbox(Equivalence partitioning, Boundary value analysis) whitebox(instruction covering, Path covering)2-dynamic analysis: memory leak analysis, performance

Equivalence partitioning

divides the input domain of a software unit into different equivalence classes

test cases are designed to cover each partition at least once

technique tries to define test cases that cover classes of errors, thereby reducing the   
 total number of test cases that must be developed  
 Reduces the time required for testing a software due to lesser number of test cases.

Boundary value analysis

tests are designed to include representatives of boundary values  
 special form of equivalence partitioning  
 Use values that are located on and around the borders of the equivalence classes

often used as a part of stress and negative testing

**Characteristics of software quality(ISO9126):**

Functionality: the needed functionality of a system, by pre- and post-conditions,Tests check the pre- and post-conditions (Suitability, Accuracy, Interoperability, Security)

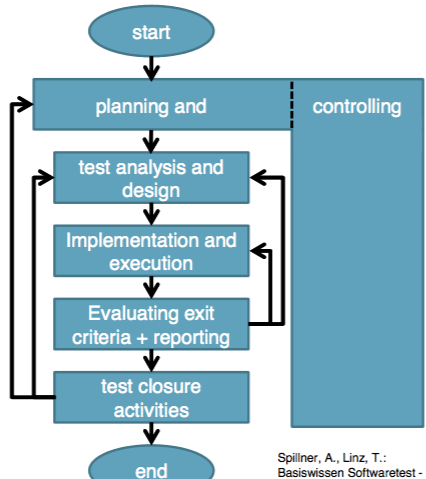
Reliability:Ability of a systems to ensure its performance within a defined period and defined constraints(Maturity, fault Tolerance, Recoverability)

Usability: Important for the acceptance of the system, depends on the group of users, checked in the context of non-functional tests (Understandability, Learnability, Attractiveness)

Efficiency: time and resources needed to execute a specific task, Characteristic is measured with performance tests (non-functional tests) (Time Behaviour, resource Utilization)

Maintainability: set of attributes that bear on the effort needed to make specified modifications. important! because software systems are often used for a long period (Analyzability, Changeability, Stability, Testability)

Portability: set of attributes that bear on the ability of software to be transferred from one environment to another. (Adaptability, Installability, co-Existence, Replaceability)

**Basic Test process(ISTQB):**

1-Planning and controlling

Planning:

1- determine scope,risks,objectives 2- determine test approach.

3-implement the test policy and/or the test strategy. (Test strategy: an outline that describes the testing portion of the software development cycle. It is created to inform PM, testers and developers about some key issues of the testing process. This includes the testing objectives, method of testing, total time and resources required for the project and the testing environments.).

4-determine the required test resources like people, test environments, PCs, etc.

5-Schedule test analysis and design tasks, test implementation, execution and evaluation.

6-determine the Exit criteria we need to set criteria such as Coverage criteria. (Coverage criteria are the percentage of statements in software that must be executed during testing. This will help us track whether we are completing test activities correctly. They will show us which tasks and checks we must complete for a particular level of testing before we can say that testing is finished.)

Test control has the following major tasks:

1-measure and analyze the results of reviews and testing.

2-monitor and document progress, test coverage and exit criteria.

3-provide information on testing. 4-initiate corrective actions. 5-make decisions.

2-Analysis and design: has the following major tasks:

To review the **test basis**. (The test basis is the information we need in order to start the test analysis and create our own test cases. Basically it’s a documentation on which test cases are based, such as requirements, design specifications, product risk analysis, architecture and interfaces. We can use the test basis documents to understand what system should do once built.)

To identify test conditions.; To design the tests. ; evaluate testability of requirements and system.  
To design the test environment set-up and identify and required infrastructure and tools.

3-Implementation and execution: During test implementation and execution, we take the test conditions into test cases and procedures and other testware such as scripts for automation, the test environment and any other test infrastructure.

(Test cases is a set of conditions under which a tester will determine whether an application is working correctly or not.)  
(Testware is a term for all utilities that serve in combination for testing a software like scripts, the test environment and any other test infrastructure for later reuse.)

**Test implementation** has the following major task:  
1-To develop and prioritize our test cases by using techniques and create test data for those tests. (In order to test

a software application you need to enter some data for testing most of the features. Any such specifically identified data which is used in tests is known as test data.)  
We also write some instructions for carrying out the tests which is known as test procedures.  
We may also need to automate some tests using test harness and automated tests scripts. (A test harness is a collection of software and test data for testing a program unit by running it under different conditions and monitoring its behavior and outputs.)

2-To create test suites from the test cases for efficient test execution.  
(Test suite is a collection of test cases that are used to test a software program to show that it has some specified set of behaviours. A test suite often contains detailed instructions and information for each collection of test cases on the system configuration to be used during testing. Test suites are used to group similar test cases together.)

3-To implement and verify the environment.

**Test execution** has the following major task:

1-To execute test suites and individual test cases following the test procedures.  
2-To re-execute tests that previously failed to confirm a fix.(confirmation testing or retesting)  
3-To log the outcome of the test execution and record the identities and versions of the software under tests. The test log is used for the audit trial. (A test log is nothing but, what are the test cases that we executed, in what order, who executed and what is the status of test

case (pass/fail). These descriptions are documented and called as test log.).  
4-To Compare actual results with expected results.

5-Where there are differences between actual and expected results, it report discrepancies as Incidents.

**4-Evaluating Exit criteria and Reporting:**

Based on the risk assessment of the project we set criteria for each test level against which we will measure the “enough testing”(criteria). Exit criteria come into picture, when:

−  Maximum test cases are executed with certain pass percentage.

−  Bug rate falls below certain level. −  When achieved the deadlines.

Evaluating exit criteria has the following major tasks:  
 1-To check the test logs against the exit criteria specified in test planning.

2-To assess if more test are needed or if the exit criteria specified should be changed.

3-To write a test summary report for stakeholders.

**5) Test Closure activities**: Test closure activities are done when software is delivered. The testing can be closed for the other reasons also like:

When all the information has been gathered which are needed for the testing. ;When project is cancelled; or some target is achieved; or a maintenance release or update is done.

**Test closure activities:**

To check which planned deliverables are actually delivered and to ensure that all incident reports have been resolved.  
To finalize and archive testware such as scripts, test environments, etc. for later reuse.  
To handover the testware to the maintenance organization. They will give support to software.

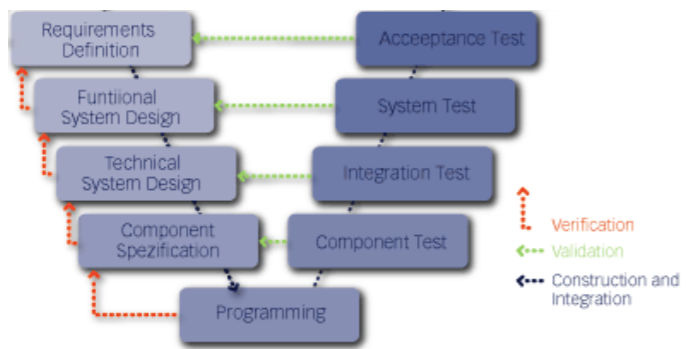
To evaluate how the testing went and learn lessons for future releases and projects

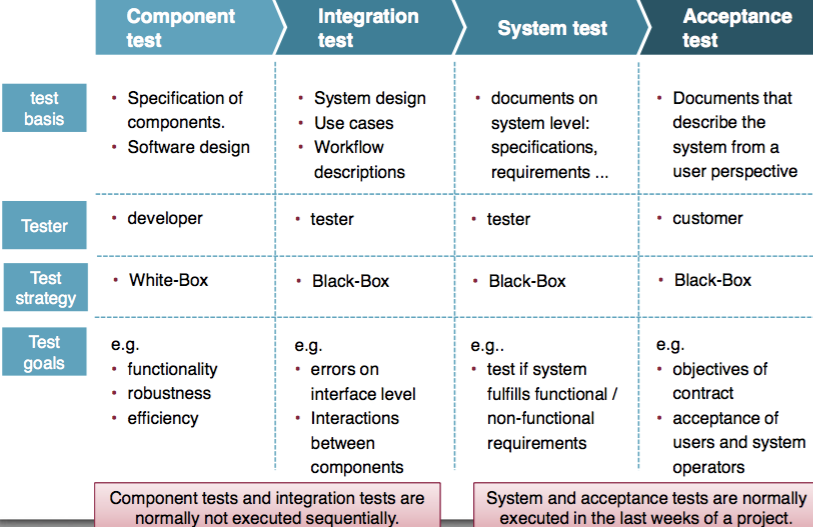
**Test level:**

-Component level: Component test verify the correct functionality of the individual components of a system. Focus on individual components. performed by the developer prior to integration.

-Integration level: Integration test is to check the flawless interaction of the system components. The individually tested components are gradually integrated and the interaction is tested. Focus on interaction of the components. So interfaces need to be tested in different combinations.

-System level: System test is the final test in a realistic environment (without customer). Functional testing against the specification, Mass testing, performance testing, load testing, Usability test. Focus of the examination is the behavior of the overall system

-Acceptance test: Acceptance test is performed by the customer in the operating environment of the customer with real data. Generation and installation of the system, Test cases for real use cases, Random test cases, Audit Documentation. Acceptance testing is the basis for the acceptance by the customer.



Early tests reduce project risks.Early tests show the quality of the product during development.

**Basic principles of software testing (ISTQB):**

1-Testing shows presence of defects: Testing can show that defects are present, but cannot prove that there are no defects. 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 (all combinations of inputs and preconditions) is not feasible except for trivial cases. Instead of exhaustive testing, we use risk and priorities to focus testing efforts.

3. Early testing: Testing activities should start as early as possible in the software or system development life cycle, and should be focused on defined objectives.

4. Defect clustering: A small number of modules contain most of the defects discovered during pre-release testing, or show the most operational failures.

5. Pesticide paradox: If the same tests are repeated over and over again, eventually the same set of test cases will no longer find any new bugs. To overcome this “pesticide paradox”, the test cases need to be regularly reviewed and revised, and new and different tests need to be written to exercise different parts of the software or system to potentially find more defects.

6. Testing is context dependent: Testing is done differently in different contexts. For example, safety-critical software is tested differently from an e-commerce site.

7. Absence-of-errors fallacy: Finding and fixing defects does not help if the system built is unusable and does not fulfil the users’ needs and expectations.

**Test tools classification:**

Tool Support for management of testing and tests

storage, administration and monitoring of test cases;administration of incidents; administration of requirements;configuration management;

Tool support for test case specification: administration of pre-/post-conditions of test cases; test data preparation tool (e.g. based on database or specification)

Tool support for static tests: support of review processes; static analysis (e.g. cyclomatic complexity);model checker

Tool support for execution and logging: test harness/unit framework tools; test comparators; coverage management tools; security testing tools; simulation; capture-and-replay-tools

Tool support for non-functional tests: performance testing, load testing, stress testing tools; monitoring tools

**jUnit**: Framework for testing Java programs , Especially used for the automation of unit tests (test of methods and classes)

Simple creation of test drivers,Easy automation of tests, Integration in IDEs, Periodical execution of +tests is possible (nightly builds), Projects specific adaptions are possible, Refactoring of code affect test drivers

-No direct separation of test code and test data,Dependent components must be initialized before test execution, No GUI-Tests possible

**Selenium**:Test framework for web applications,Capture manual executed operating steps.

Operating steps are stored as test scripts. can repeat the stored test scripts. can store test scripts as HTML-tables. based on HTML&JavaScript.Selenium-IDE available as Firefox Add on

+Simple creation of repeatable GUI-test scripts, Suitable for regression tests, Contains a reasonable reporting of executed test cases, Generation of test scripts possible.

- No direct separation of test code and test data. Very sensitive according to GUI-changes.Test cases are not influenced by refactoring.Manual post-processing of scripts is necessary. For a long term use of Selenium you need a good test (framework) architecture (modularization of test cases, ...)

**Automated testing:**

+Time pressure and risks are reduced in later project phases, It is possible to test the application quickly after a change, especially important after the go-live, Protection of side effects, Cost saving, Less stupid test executions

-Team is too inexperienced regarding testing, • wrong tools, • Different tools are not compatible, Test case not maintainable or it is not possible to automate test cases, High costs for maintenance of test cases

Order to introduce tool support for testing: 1.defect management,2.configuration management,3. test planning,4.test execution, 5.test specification

Test automation pays of after round about **5** repetitions of test cases

**Elements of test concept**: unit test at the level of components(developer test), Continuous integration with nightly builds and reports, Creation of test case specifications, automated integration tests, central creation of test data, manual integration tests

Developer tests

−  Manual developer tests for dialogs −  JUnit tests

−  Developer was only allowed to check-in code changes, if all automated tests of the corresponding component have been successful.

−  Concrete guidelines about kind and coverage of tests

−  Refactoring of Junit tests has been working well

Continuous Integration

−  Continuous Integration with Hudson

−  After a developer checked-in new code, Hudson has built the application and executed all Junit tests

−  Detailed reports and automatic delivery of reports to responsible persons

Nightly-Build

-Nighly-Build was executed daily  
 -Build and deployment of the application for manual tests on the next day.

-Execution of automated developer and integration tests during nightly-build

-History of reports(Sometimes we needed to identify at which time an error occurred the first time)

•Test case specification

-Concrete guidelines how to extract test goals from the system specifications(ensured test coverage e.g. test pre/post condition)

-Test strategy on the level of test case specification(dialogs: manual, use case: automate) -Decision about used tool (Proven! or JUnit)

-Review of test case specification (Early quality assurance, Check of completeness)

-shared test case specifications with our customer

———————————————————————————————————————————

# SAP

**ERP**: Integrated package of different business applications, usually bundled within one system. "real-time" as promoted key feature. Often with a module based architecture.Integration of business processes. possibilities for analysis and controlling

Benefits: Allows the integration along the value-added chain, Allows decision making along all hierarchies.

———————————————————————————————————————————

# Scalability and performance

performance :Response time(time user wait), Throughput(task per time), Resource consumption(CPU, ..)

Scalability: How does the system work under higher load, How do faster hardware or more hardware improve the performance

Low performance=>Cost=> (Customer satisfaction / image damage, productivity, Buy new hardware=>(often does not solve)

**Performance within the Activities**

Requirement

Gather, document and cross-check the non functional requirements

Document for each requirement; Frequency of function calls + number of parallel users

Volume of data to work on; Volume of data to transport; Maximal response time

Design

Cross-check the non functional requirements; Write guidelines for the developers;

Design an architecture to meet the non functional requirements

Implementation

Follow the guidelines; Keep typical performance killers in mind;

Inform the architect / designer if any (performance) problems occurring

Test: Plan performance and load test; test in realistic env.(same hardware, data volume, load)

**Strategies for High Performance**

1)Transfer spare data: low data volume, few remote call, low additional latency(protocol low overhead), use caching(+reduce access to server, database, storage. –reduce access to simple bus. Logic, reduce access to ORM, have to be refreshed manually)=> to much caching=>reduce performance & use more memory

2)Process data fast: no unnecessary work, efficient alg. => less maintainable, efficient mechanisms of PL, Efficient library.

*Architecture*: stateful

Stateless: does not hold state, less memory, good for load balancing, high scalable(Additional servers can be used to serve more users), hard to impl.

*Redundancy*: problem parallelizable, session handling, increased failure probability, additional complexity

3) prepare for redundancy: take into account from beginning, avoid bottleneck: design with no bottleneck

**Availability**: The ratio of (a) the total time a functional unit is capable of being used during a given interval to (b) the length of the interval.

Cold standby: If the first server fails the second one takes over. The standby server is started when the failure occurred.

Hot standby: The standby server is always online

Cluster: Second server runs as described for scalability

Rolling update: With additional redundancy even during the update a failure doesn’t bring the system down(group of cluster)

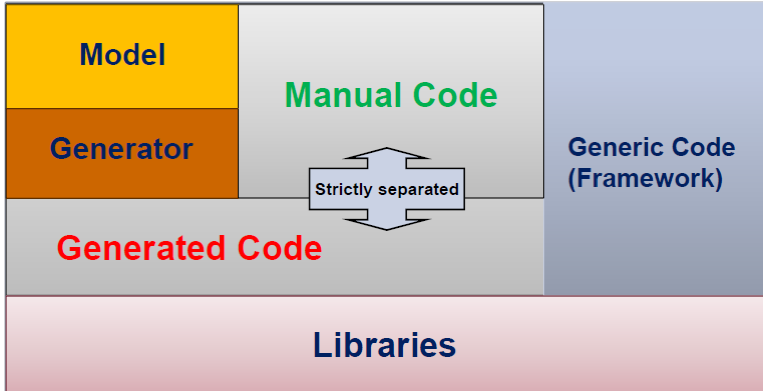
———————————————————————————————————————————

# MDD

+Abstraction, Composition/Decomposition; Automation; Business Domain; Communication

Modes: Communication media btw Bus. & IT; For bus. Domain orientation; dealing with complexity

**DSL**: better&quicker, reduce learn overhead&complexity, stay in context of problem



**MDD Adv:**

1-Increased productivity: developer is concentrating on creative things and not on the „boilerplate“ code

2- Resolving the complexity: Only architects and generator experts have to understand the framework and the generator• Business Logic (A-Software) and Technology (T-Software) are separated • Developers focus on the business logic

3-Better quality: Architectural Patterns are implemented only ONCE.

Architectural Changes are controlled and handled centrally in the Generator

• Number of possible design failures decreases

• Principal changes have to be discussed with the architecture team

• Developers tend to produce less code, but better code

4-Flexibilty: Technology Change( Model remains, Generator changes) Product Lines( Models change , Generator remains)

5-Documantation: The model is the core of the development; The model has to be always up to date (since the code is generated from it); The business concepts are documented in the model; Different documents can be generated from the model

No heavy model, graphical representation, DSL for GUI,

**Practical hints:**

1. Development and Maintenance of Generators takes time and costs money.

Consider the costs for Modeling, Export and Generation in your projects.

2. Good real Reference Examples (Application Prototypes) must be created and tested before the creation of Generators.

3. Separation between generated and manual code is necessary. „Protected Regions“ (Mix) is an Anti-Pattern.

4. Generated Code should be well structured and well readable, since the developers deal with source code.

5. Versioning (Releases, Branches) for Generators, Models and Source Code is extremely important. A consistent and reasonable Release-Concept is absolutely necessary!

———————————————————————————————————————————

# BPM

**BMP**: Describe business objective in the same language the computer needs to automate it

• Automatically guiding the stuff through the process

• Create documentation anytime,not up front

• Business and IT working parallel, continuous changes

• Bridges the gaps between Requirements, Specifications and Code

* + - Focus is not just on code generation,but on a process to guide you and allow you build for change

**BP**:consists of a collection of *activities* that are executed in some enterprise or administration according to certain rules and with respect to certain goals.

WF: A **Workflow** is the realization of a business process by some information system.

Activity: An **activity** / **task** of a business process is an atomic work step that, on the given level of abstraction, cannot be split into more detailed steps.

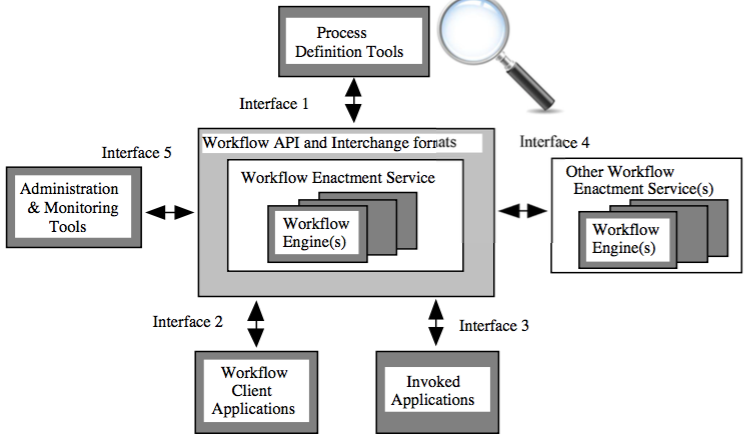
**Business Process Management:** is a discipline, which deals with discovery, organization, documentation and improvement of business processes. Business Process Management synchronizes such business areas as Planning, Design, Construction, Production, Maintenance, Tracking, and Adjustment in an organization.

**Information system**: A system for storing, retrieving, combining and evaluating information.

**documents** are created, used, and changed. These documents help to exchange information among different activities of the same business process and among different business processes.

**resource** is a means necessary for executing an activity. When the resource is a person, we call the resource an **agent**.

**WFMS**: is an information system for the development,planning,control, execution, and monitoring,documentation and evaluationof workflows (business processes).



**BP Modeling**: representation of a business process in a standardized notation. A model is a network of activities and their relations (control flow). Activities can contain information about associated data, business services and resources.

**Business Modeling**: Business experts and analysts model processes in order to illustrate(who is doing what?who needs what?who communicates with whom? where the data flows? where is the standard flow and where are the exceptions? where is the critical path?)

**Technical Modeling**: The goal is the automation and the process and its support by IT, Extensions of business processes with technical details,The processes are executable by the process engine.

**BPEL** (BP Execution language): Supports web services, Is not understandable by business people  
No graphical notation, BPMN 1.X was mapped to BPEL in order to execute it

**EPC**(Event driven process chains): EPCs were used by SAP for documenting processes EPCs can be transformed to BPMN 2.0, Comparison to BPMN: (No separation between data and event flows.Start, End and other events can no be distinguished. No event types (e.g. time, message).  
Can not be executed by the process engine)

BEST PRACTICE:

**Separation of concerns:**

Business Processes represent the real business cases and have a domain-specific canonical business data model. The orchestration of services is done on this level. Business Services or Subprocesses are used within the processes.

Business Services represent the reusable and self- contained services, which utilize the technical services. They are responsible for transforming the technical data models of technical services into canonical business data model. Business Services can call other Business Services.

Technical Services are responsible for encapsulating the technical binding of applications and data. They know the data structures and functions of specific applications. The use specific interfaces to Files, Databases, Queues, etc. directly or with the help of adaters. They reference physical resources. They provide different interfaces (e.g. Webservices, REST, Java RMI, etc...).

**Structure the process**: enables better reuse and separation of concerns.

**Develop processes incrementally and iteratively**: Rapid development of the first version leads to the *quick business feedback* and initiates a detailed discussion of exceptions and rules. During the further iterations the process is stepwise refined and exceptions are handled. The core of the business process automation is „incremental“, since the implementation is growing incrementally over multiple iterations.

**Focus on agility and changes**: The result of every iteration is a working application, not a paperwork

Focus on people and on the organisation: „Bring the decision to the place where it will be carried out“

———————————————————————————————————————————

# Agile

+individuals and interactions, working software, customer collaboration , responding to change

vs: processes and tools, comprehensive documentation, contract negotiation, following a plan

**Scrum**: concepts: small team (5-7 people),development of a software product,well-defined roles,aligned (small) set of key practices,built around the iteration cycle

**XP**: Values: Communication, Simplicity, Feedback, and Courage

12 Practices: Pair Programming, Collective Code Ownership, Continuous Integration, Automated Tests, Whole Team, Refactoring, Sustainable Pace,  
Short Iterations, System Metaphor, Coding Standards, Simple Design, Planning Game

**Kanban**: values: Continuous Flow, Minimize Work in Progress Pull principle, Stop the line, Kaizen

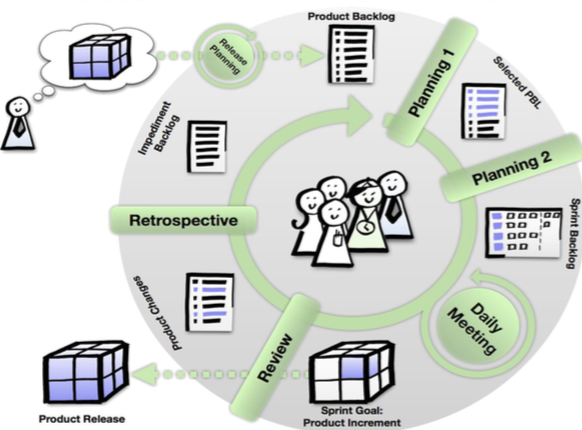
**SCRUM**

**ROLES**:

Product Owner:Knows the requirements,Responsibility, that needed functionality is delivered at the right time

(Has a vision about how the final product should look like, Secures the budget, Slices the vision down smaller packages (user stories or epics),Knows when she/he needs it,Can explain the requirements, Can prioritize requirements, Maximizes return on investment)

Scrum Team: Creates final product (includes developers, testers, database admins, etc.)

Scrum Master: Ensures that the working methods run smoothly, Ensures that the methods are followed,Tackles obstacles that arise outside the team

**Release planning**: Done by the Scrum Team together with Product Owner; All known user stories / epics of the software are estimated; Items that will be developed in the next few iterations will be broken down to smaller bits to fit an iteration

**Estimation of effort**: Planning Poker: Estimation by all implementing team members (comparable to Delphi Method);Evaluation is done in story points; Each participant has a set of cads.

**Steps**: Moderated by Scrum Master  
• For each user story to be estimated:Product Owner presents user story, Team asks Product Owner questions, about it until it feels confident to estimate the user story, Each team member estimates the task and puts a card upside down on the desk, All cards are turned to show the estimations at the same time, Highest and lowest estimation are discussed in the group, Estimation is repeated up to two times or when the estimations match  
• The meeting lasts for maximum of one hour

**Product Backlog**: Product Owner orders the user stories and epics by their importance / return on investment, Identification of possible release milestones

**Planning 1**: Product owner and team meet at the beginning of the sprint, Team selects as many items from the top of the backlog as they think they can finish within the sprint , Team gives commitment to deliver at the end of the sprint

**Planning 2**: Team divides user stories to tasks (bites that one person can chew in one day) , Developers pick their first tasks on the task board

Daily Meeting / Daily Stand-up: 15 minutes for the whole team;Three Questions answered by each team member: What you have done during the last day? What you plan to do today?What is your current obstacle?; Move the tasks together on the board

**Review & Product Increment**: Product increment finished at the end of the sprint;At least the most important stories are now complete; Delivered as fully working product;Review together with the product owner and maybe other stake holders; Possible product release

**Product Changes**: Product changes are added to the product backlog

**Retrospective and Impediment Backlog**: Reflection as a team to find out things that worked well and that need adaption; Result: prioritized Impediment Backlog; Scrum Master will handle Impediments outside the team, impediments inside the team are handled by the team

**Add a feature:**

1) new req: user story=> (roles, functionality and motivation are now clear) (INVEST / independent, negotiable, valuable, estimable, specific, testable)

2)Priorities of requirements

3)Detailing Requirements: Work of a business analyst (Product Owner?)

(Use Case Documentation,Mock-up Screens,Activity Diagram)=>understood by business and developers (costly)

Estimation with Planning Poker: =>(Effort Points per User Story, Acceptance Criteria)

**Definition of Ready**: Quality gate before actual development starts , Ensure that enough information is available to be able to complete the story in one sprint

**Sprint Planning I**: Vision for the next Sprint (2 weeks) ; Presentation by Product Owner of prioritized Back Log; Team selects the stories that can be implemented in the Sprint (taking into consideration skills) ; Commitment of the team to deliver these Stories according to the „Definition of Done“ => (docs: prioritized and estimated User Stories, res:Selected Product Backlog)

**Sprint Planning II**: Dividing a Story into Tasks (max. one developer day) Documents: User Stories in Selected Product Backlog Result: Tasks

Daily Meeting / Daily Stand-up: 15 minutes meeting;What you have been working on, what is ready, problems, next steps (Documents: User Stories in Selected Product Backlog, Tasks Result: Sprint Burn Down Chart, Impediments)

**Definition of Done**: Specified by business w. help of business analyst (checked by team member)

•Analysed by developer (checked by second team member); Developed, automated test, documented and committed by developer (reviewed and test by second team member); Automated deployment to test environment; Module user acceptance test (approval by business)   
(second quality gate)   
**Documents**: User Stories in Selected Product Backlog, Tasks, Mock-Ups, Use Cases, Activity Diagram, Acceptance Criteria; Result: Application, Developer Documentation

**Sprint Review**: Implemented Stories presented to Product Owner and Stakeholders

Documents: User Stories in Selected Product Backlog, Mock-ups, Use Cases, Activity Diagram, Acceptance Criteria

**Sprint Retrospective**:Team reviews development process and discusses and decides about changes Documents: (Impediment Backlog)

Obstacles for agile teams:

No access to product owner or similar person;No single sponsor

Missing cross-functional and skilled team; Team members not full time on the project

Technology doesn’t support continuous integration and automated tests

Transparency and inspect & adopt are not compatible with organisation’s culture

You’re not allowed to fail; Not enough urgency / complexity / novelty

Large scope, very few releases; Large teams, multiple geographies, different time zones

High Visibility already in early phases of the project

———————————————————————————————————————————

# Project management

**Project**: Undertaking that is characterized as being unique through its circumstances. Uniqueness regarding: goal; time, budget, resources; organization;difference from other undertakings.

in reality: A Project is an undertaking to reach a defined goal in a given time with limited resources whereby the way to the solution is not given or known.

**Project Management** is the practical, goal-oriented solution of a task in a defined time and cost frame on the basis of theoretical insights.

**Project Management Institute (PMI)**: Project Management is the application of knowledge, skills, tools and methods to project tasks to fulfill project requirements.

Different sizes of projects behave quite differently and therefore have to be managed differently. n\*(n-1)/2

Break down team into subteams:

Coordination stays within subteams.  
 Synchronization of results throughout lifetime of project.

Separation of concerns minimizes coordination between subteams.

Dependencies between subteams have to be managed

**Types** of IT Projects: Custom build software; software product; roll out of a software product; run&maintain; IT Consulting; answer to RFP

Responsibility of a project manager=>time, budget, quality.

**Project planning**:

**First level**: Phase(period of time, has goal and main activites)( Phases are separated by milestones(special event in project, it has: specific result, deadline, status(planned, in progress,achieved))

The **critical path** is the path through all work packages without any buffer. Any delay on the critical path jeopardizes the deadline.

**project management phases**: 1.preparation, 2.initialization, 3.execution(iterative processes: planning, execution, controlling, steering) ,4.finalization.

1. Mediator between Business and IT (In the direction of business:(ew opportunities offers by IT; processes be optimized); In the direction of IT:(Implications for systems, Feasibility, synergies)

Projects have to be covered by a business case[A business case makes a project controllable and steerable when times get rough]  
 (Total cost of ownership ,Benefits should be quantified)

Priorities: should reflect importance, not urgency. It is never difficult to decide what is important. It is difficult to decide what is not important (and therefore will probably not be done at all).

2-decide on process model: Depends on circumstances and Process Models have to be enhanced project specifically

Team building: Find the right persons, Kick-off meeting at the beginning. Goals: Static organization (rolls, tasks, team, ...)Dynamic organization (plan, reports, meetings, …)

3.1. Project planning: A plan has to be realistic. When your expectations are too high you will never be successful. Project planning is constructive! If you don’t plan you don’t have a plan. But you also don’t know what to do. When planning you do constructive thinking about what to do. Using a tool to document the plan is not much effort.

3.2. Control: When controlling progress the view has to be focussed on the future, not the past.