Software Project Management

Software project management

1:

We can group our activities in **Day-to-day** vs **temporary. Day-to-day** activities are like being a pilot, a singer, or a cook. You do the same activities like a routine. **Temporary** activities are more of a project, and start and end at some clear point.

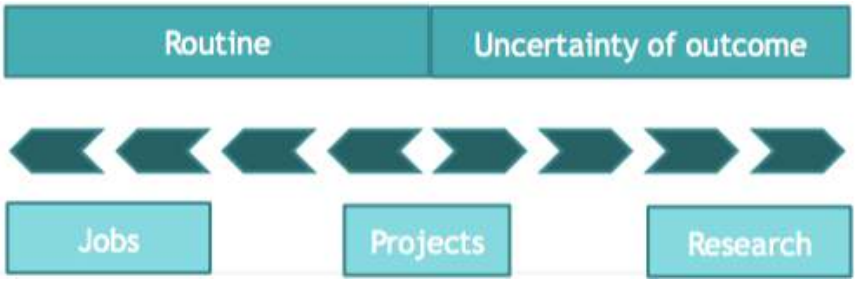
**Operations**: *Frozen stable pattern*.
- ongoing day-to-day activities
- standing roles and responsibilities
- certain objective: doing task right within timebox

**Projects**: *unfreeze – change – refreeze*
- Temporary activities
- Temporary roles and responsibilities
- Uncertain objective: create value for the future

Executing a project is an investment: consuming resources in the expectation of some benefit in the future.
- **Resources**: time, money, goods, machine-time, knowledge, etcetera
- **Expectation** in the future: uncertainty -> outcome can make you rich or poor
- **Benefit**: increase earnings, cost reduction, risk mitigation

A project is a **temporary** endeavour undertaken to create a unique product or service
- **Temporary**:
  - Activity has a defined end goal
  - Activities are non-repetitive
- **Unique purpose**:
  - Outcome changes the environment and/or organization
- **Requires resources**, often from various areas
- A project involves **uncertainty**, often new 'never done before' activities are involved
- A project has a primary **customer or sponsor, and beneficial owner**

**Project characteristics**:
1. Planning is required
2. Non-routine tasks are involved
3. Specific objectives are to be met or a specified product is to be created
4. Work is carried out for someone other than yourself
5. Work involves several specialisms
6. People are formed into a temporary work group to carry out tasks
7. Work is carried out in several phases




**The money needs to come first to get results**

- The investment is 'at risk' during the project
    - **Software project management** helps to control the risk, i.e. time and budget
    - People call a project successful if the expected risk is not exceeded, i.e. no unexpected delays and costs, and if the result is bringing the expected benefits.

Are software projects really different from other projects? Not really, but:

- Invisibility
- Complexity
- Flexibility
- No building phase, only designing
- Cheap, but finding good people is hard


Make software more problematic to create than other engineered artefacts

**Feasibility study**

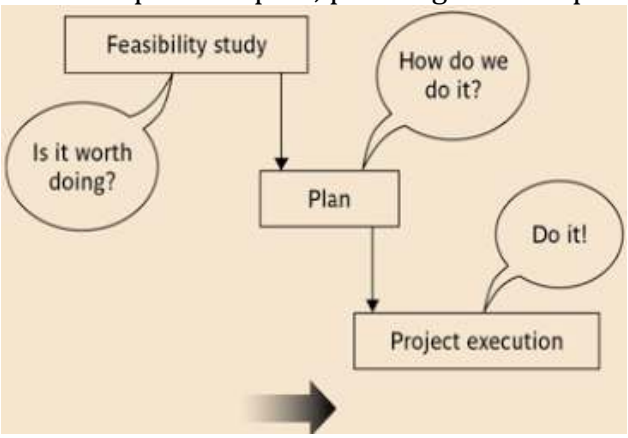
- Is a project technically feasible and worthwhile from a business point of view

**Planning**

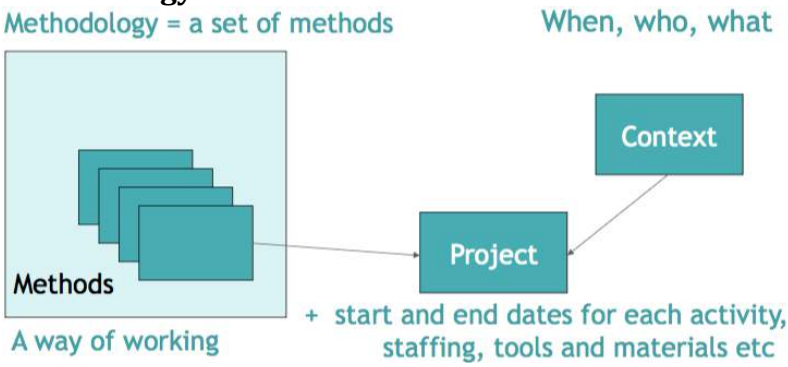
- Only done if project is feasible

**Execution**

- Implement plan; plan might be adapted as we go along (and learn)




**Methodology**




**Two philosophies for project management**

**Authorative:**

Project goal: deliver exactly what has been requested, on time, on budget

Methodologies:

- Prince2
- PMI
- V-model
- 'Waterfall'

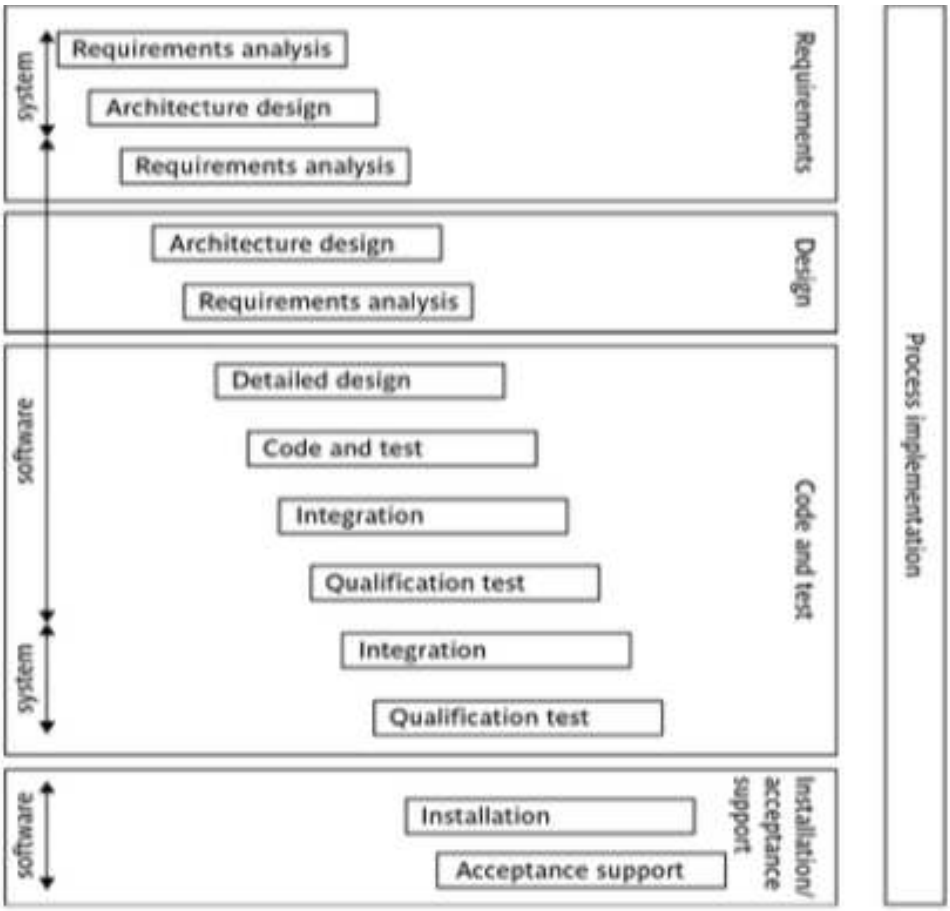
**Liberal:**

Act autonomous (within certain boundaries) to deliver maximum value for customers

Methodologies:

- Agile
- Scrum
- Spotify model

The software development life cycle (waterfall/classic): Great for a deterministic project with low nr of unknowns, i.e. not typical for software




Essence of project management: To ensure a project successfully meets the project goals (objectives)

- Establish project goals
    - Determine what is truly important
- Track progress towards the project goals
    - Provides means to correct along the way
    - Document the project along the way
- Manage risks and expectations

**Setting objectives:**

- Answering the question *'What do we have to do to have a success?'*
- Need for a project authority
    - Sets the project scope
    - Allocates/approves costs
- Could be one person – or a group
    - Project board
    - Project management board
    - Steering committee

Objectives should be SMART
S – specific, that is, concrete and well-defined
M – measurable, that is, satisfaction of the objective can be objectively judged
A – achievable, that is, it is within the power of the individual or group concerned to meet the target
R – relevant, the objective must be relevant to the true purpose of the project
T – time constrained: there is defined point in time by which the objective should be achieved

In short: if you cannot show/buy it, it does not add measurable value

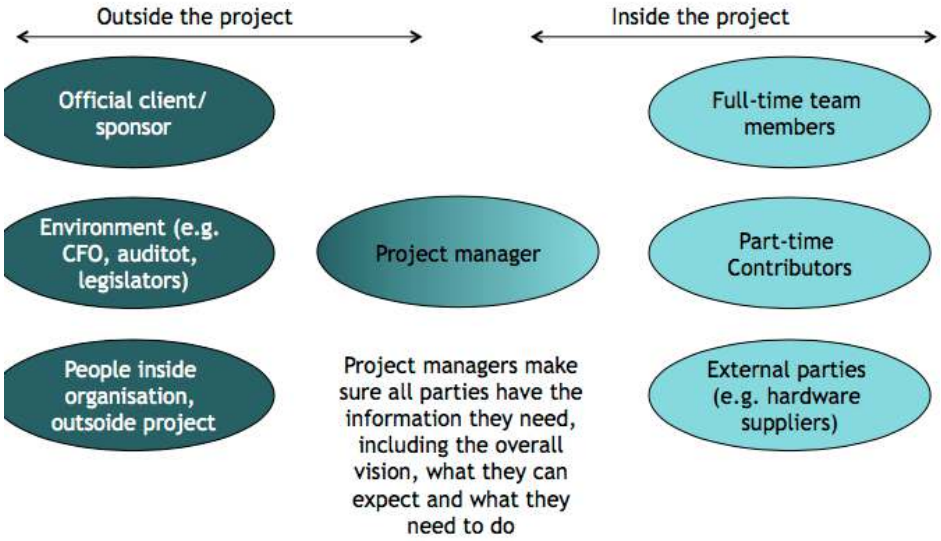
**Measures of effectiveness**
Goals:

- overall goal – user satisfaction with software product
- analyst goal – accurate requirements
- developer goal – reliable software
- user goal - operational task taking less time

How do we know that the goals have been achieved?

- By a practical test, that can be objectively measured and assessed. E.g. for user satisfaction with software product:
    - Number of complaints – if low etc etc
    - Time to execute an operational task using the system

**Project management role**




**What is management?**
This involves the following activities:

- Planning – deciding what is to be done
- Organizing – making arrangements
- Staffing – selecting the right people for the job

- Directing – giving instructions
- Monitoring – checking on progress
- Controlling – taking action to remedy hold-ups
- Innovating – coming up with solutions when problems emerge
- Representing – liasing with clients, users, developers and other stakeholders

**Four dimensions of development**


These are dimensions: so not a trade-off but combination, or better: synergy

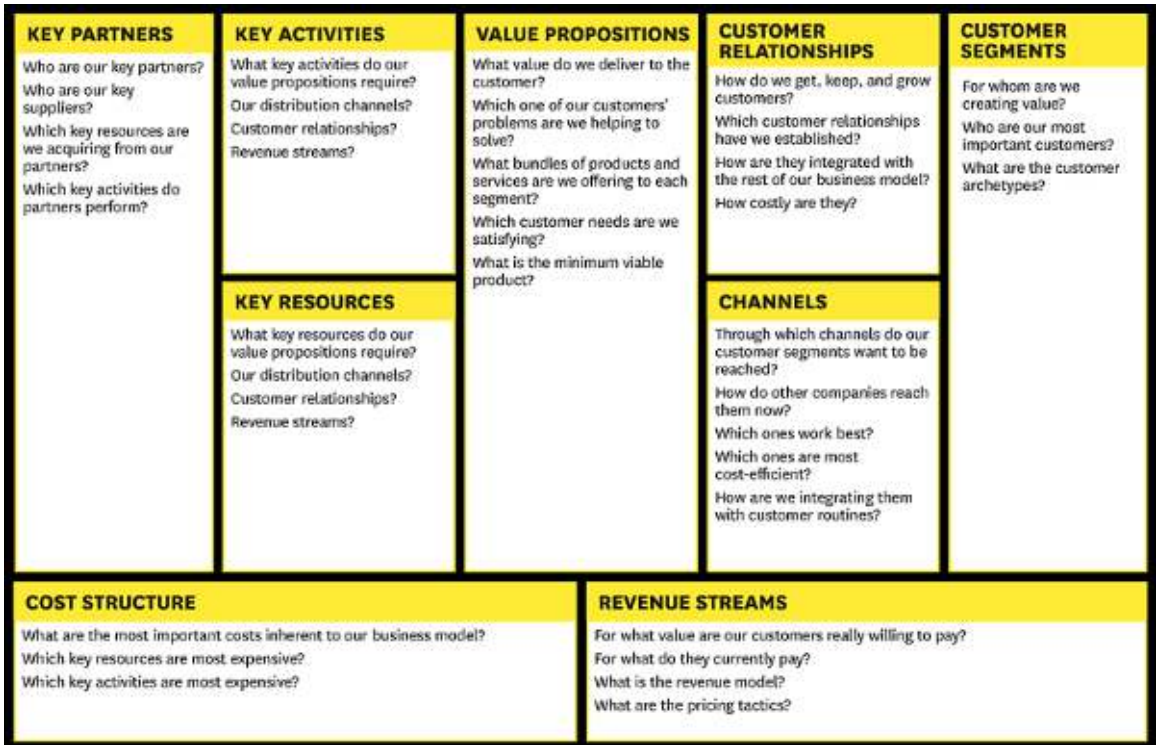
**People dimension**

Improvements:

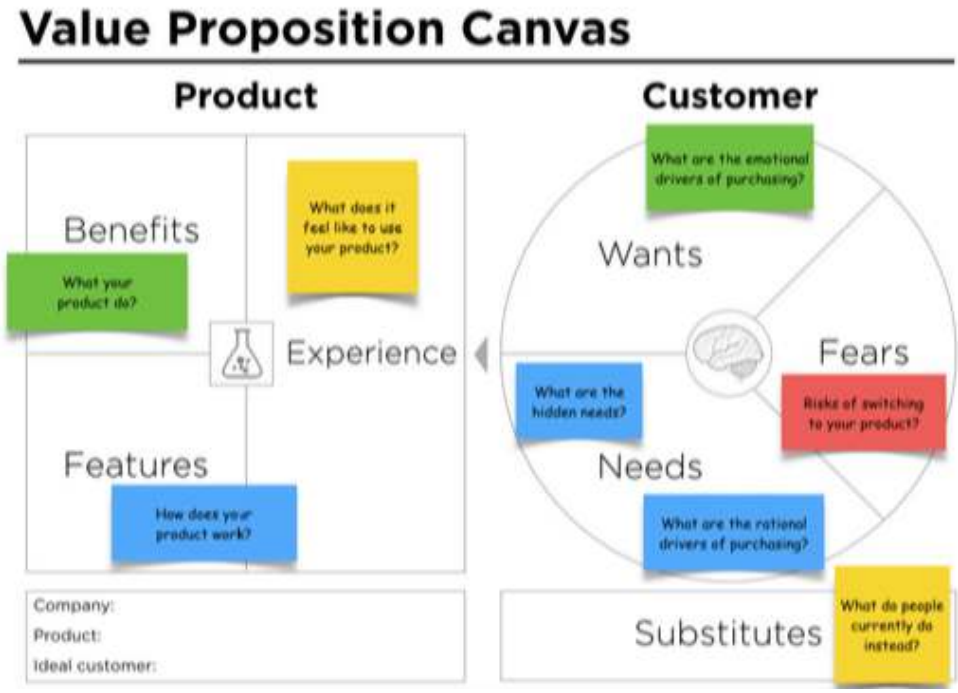
- Team selection
- Team organization: roles and responsibilities (matching people to tasks)
- Motivation
- Career development
- Balance: individual and team
- Clear communication

Conway's law: *"organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations"*

**Business model canvas:**




**Value proposition canvas:**




**Why, how, what**

**Why = the purpose**. What is your cause? What do you believe? Apple: we believe in challenging the status quo and doing this differently

How = the process. Specifications taken to realize the Why. Apple: our products are beautifully designed and easy to use
**What = the result**. What do you do? The result of Why, Proof. Apple: We make computers.
In a circle, why is in the middle, then how, and then what.

2: Scope management

The scope is about:

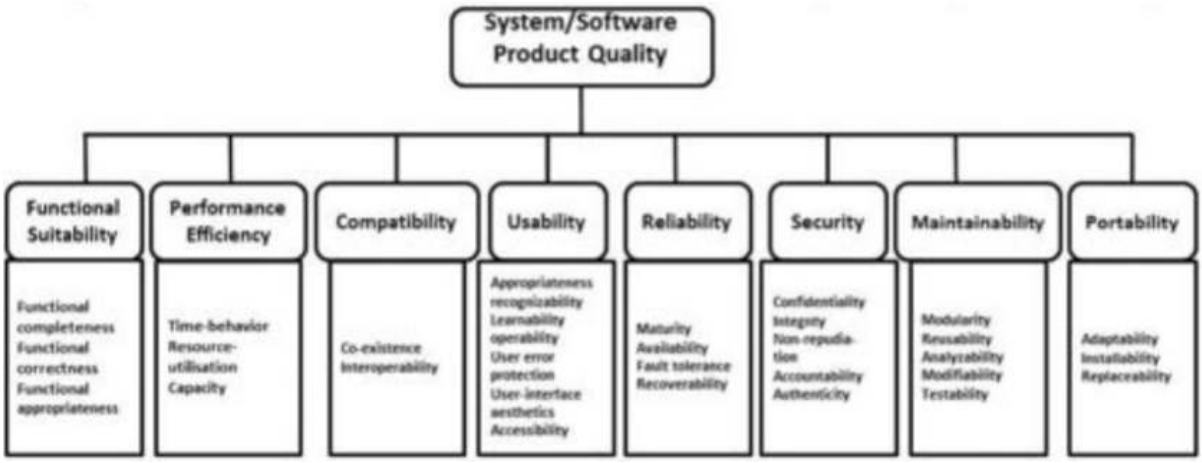
1. What will the project produce in the end
2. The process required to ensure that all the work required to complete the project successful

**Defining project scope**

- What is the work that will be done
    - Describe what is IN scope
    - Describe what is OUT scope
    - Describe what should be decided by stakeholders, and give consequences
- Scope management (plan/manifesto)
    - How will the scope be defined, managed and controlled?
    - How will the scope be communicated to the team and stakeholders/community partners?
- Scope creep
    - Incremental expansion of the project scope
    - Introducing features not originally planned
        - Delay project and add cost, or cut of quality

**Relation time, costs, scope and quality**
Time, costs and scope behave like Boyle's law, where quality is constant for a given time, budget and scope.

Software quality model ISO 25010




**Simplified model for managing scope**

Collect requirements -> define scope -> create WBS -> verify scope -> control scope

What is a requirement?

- A condition or capability needed by a user to solve a problem or achieve an objective
- A condition or capability that must be met or possessed by a system
- Types: functional and quality requirements; constraints on the environment and technology of the system

Make sure all requirements support the realization of the vision

Techniques to collect requirements:

- Interviews, focus groups, facilitated workshops, group creativity techniques, group decision making techniques, questionnaires and surveys, observation, prototypes. All methods are fine, as long as they deliver valuable user stories.

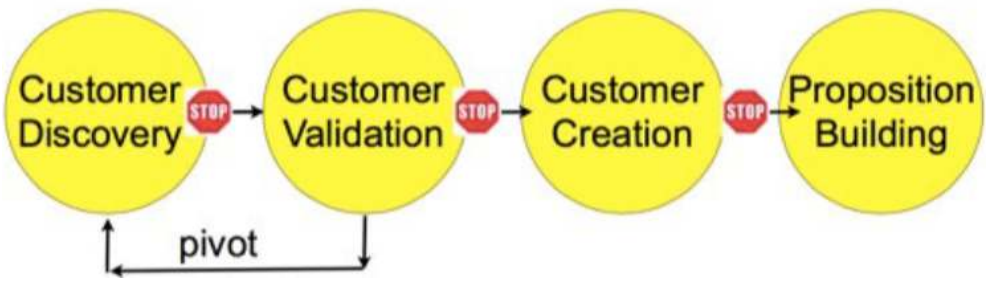
**User stories**

As a <role>, I can <what>, so that <why>. For example: As a driver, I can check my speed so that I do not get speeding tickets

- Risk: customers don't always know what they want and can only provide a vague idea. Remember: a plan is never the goal

- Why interpreting is hard: because it is about semantics. For example, if you need to find a coffee solution for VU students, what would be the best solution? You do not know the causes or reasons for this need.

The concept of **minimum viable product (MVP)** kicks in. Create the smallest possible solution (MVP) to test the projects assumptions




Even if it is an internal project, you want feedback from your stakeholders.

**Scrum** is a project method supporting early feedback. Whatever project method you choose for (or is chosen for): embed feedback

For a shippable product (MVP) you need a Definition of Done (highly influenced by the quality requirements.

- Simple definition: it looks ok and you can click through it without errors.
- Extended definition:
    - Feature implemented
    - Unit tests added
    - Code reviewed by second developer
    - Etc etc

Guidelines for definition of done:

- Potentially shippable
- At least one stakeholder will complain if we leave it out
- Multiple features are doable and testable in one sprint

**Scope risk: gold plating**

This term is given to the practice of exceeding the scope of a project in the belief that a value is being added. These changes inevitable consume time and budget and are not guaranteed to increase customer satisfaction.

- Are your features and requirements meeting real needs

**Mature projects perform A/B testing**

All business expectations are assumptions that can only be tested on the real market.

**Scope risk: lack of change control**

You can expect there to be a degree of scope creep in most projects, therefore it is important to design a process to manage these changes.

**Communicate that software projects are complex, in general**

Many projects run into problems because they are new in an industry and have never been done before. Nobody knows what to expect, there are no lessons learned and no one to ask.

- How many people have made a distributed multiplatform, worldwide, schedule system for planes?
- A webshop is a couple of days work?! (and what about payment provider integration, layout skin, and catalogue transformation..)

And of course, as it is about value/money and external stakes: there are sometimes people who want 'everything' for nothing. Keep in mind you should say NO to an over demanding client, or even considering not to sign the contract. And as client/sponsor, define a number of progress KPI's for lazy suppliers

**Project scope statement/manifesto**

- Product scope description, product acceptance criteria, project deliverables, project exclusions, project constraints, project assumptions. Note: not every organization, boss, sponsor, client is able to deal with an agile way of working. In that case, be clear about the responsibilities.

**Work Breakdown Structure**

- Aims for identifying all the work to be done
- Is ideally a decomposition of the work in atomic (executable in isolation) sub tasks
- A WBS is a foundation of the project that provides the basis for planning and managing project schedules, costs, resources, and changes

The phases in an average software project/sprint: Design -> build -> test -> deploy. There does not exist something such as the perfect WBS. Even in an agile environment, WBS will help to get an overview of major tasks and phases to be done.

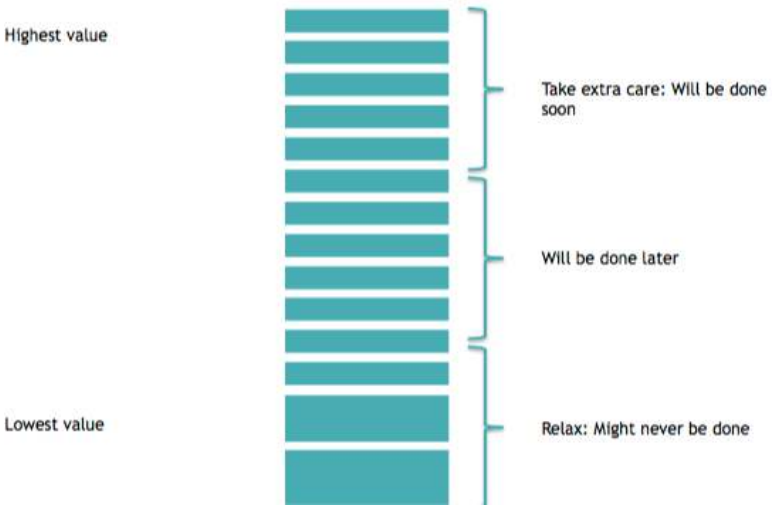
**Approaches to developing WBSs**

- The analogy approach: review WBSs of similar projects and tailor to your project
- The top-down approach: start with the largest items of the project (outputs) and break them down (activities, specific task)
- The bottom-up approach: start with the specific tasks and roll them up

Various forms of representing a WBS:

- Graphical
- Textual (written document, presentation)
- Tabular
- Management system (Jira)

**Product backlog: value comes first in Agile**




**Scope verification/intermediate delivery of valuable results**
Scope verification involves formal acceptance of the completed project scope by the stakeholders. Acceptance is often achieved by a customer inspection and by the sponsor and then sign-off on key deliverables.

**Staffing approach**
Roles:

- Get insight what kind of work needs to be done
- Select a team based on the necessary expert domains
    - UX design, algorithm experts, data analysts, etc.

Next courses are about estimation nr of FTE (Full Time Equivalent). Ideal team size does not exceed seven persons.

People tend to be cynical about projects. They seem to fail, or at least you may read a lot about failing projects for two reasons:

1. Fails are entertaining
2. Fails are useful to make your work relevant

Still projects fail may result in real losses: bankruptcy and/or firing people. So what can you at least do? Keep the scope under control. THINK BIG, act small. Your vision can be as big as the universe, what you do is what you can control now.

**To summarize: techniques to handle scope creep:**

- Be sure you thoroughly understand the project vision. Meet with the project drivers and deliver an overview of the project as a whole for their review and comments.
- Understand your priorities and the priorities of the project drivers. And update them once in a while

- Define your deliverables and have them approved by the project drivers. Show progress wherever it is possible.
- Break the project down into major and minor milestones and complete a generous project schedule to be approved by the project drivers
- Expect that there will be scope creep, have a process for that
    - It is not easy, as stakes are involved. Do you want to lose your client, the tender to a competitor or make a loss on your project.

**Scope control is about the expectations of the stakeholders**

- Goals of scope control are to:
    - Influence the factors that cause scope changes
    - Assure changes are processed according to procedures developed as part of integrated change control
    - Manage changes when they occur
- In case of agile: discuss the prioritization of the backlog with the stakeholders.

3: Lifecycle planning including Agile

**What is a software life cycle?**

Period of time that begins when a software product is conceived and ends when the product is retired from use.

- Main function: to establish the order in which a project specifies, prototypes, designs, implements, reviews, tests, and performs its other activities
- Every software-development effort goes through a 'lifecycle' -> new product development and maintenance updates of existing software

Lifecycle planning

- Three primary lifecycle model components
    - Phases and their order
    - Intermediate products of each phase
    - Reviews used in each phase
- Greatly influences your chance of success
- Not choosing a lifecycle is a bad option

Inappropriate or lack of lifecycle model: slow work, repeated work, unnecessary work, frustration.

**Software life-cycles**

1. Pure waterfall
2. Code and fix
3. Spiral
4. Modified waterfall
5. Evolutionary prototyping
6. Staged delivery
7. Design to schedule

8. Evolutionary
9. Design to tools
10. COTS (commercial off the shelf software)
11. SCRUM

**(Pure) Waterfall model**

- The "grand daddy" of models defining basic phases
- "Pure" model: no phases overlap
- Document-driven

When?

- For well-defined and stable product definition, but complex
- Quality requirements dominates cost and time
- You have a technically weak staff or an inexperienced staff
    - Reduce energy

Disadvantages

- Not flexible: rigid march from start -> finish
- Difficult to fully define and specify requirements at the beginning of the project
- Can produce excessive documentation during the lifecycle
- Few visible signs of progress until the end

**Code and fix**

1. Produce some code
2. Check if it works
3. Fix the errors, i.e.: start with step 1 again.


When?

- Tiny projects that you intend to throw away shortly after they are built
    - For small proof of concept programs

Advantages

- It has no overhead (do not spend time on planning, documentation, etc)
- Requires little expertise, (anyone who has ever had a computer program)

Disadvantages

- it provides no means of assessing progress and quality or identifying risks. This lifecycle model has no place on a rapid-development project

Spiral

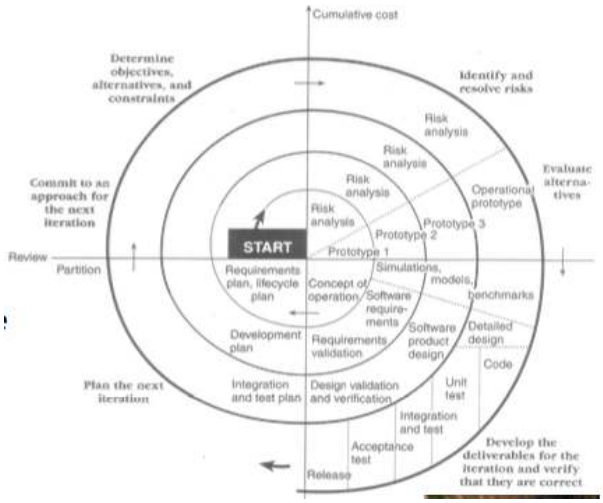
- Risk-oriented life cycle model
- Miniprojects

Each iteration involves:

1. Determine objectives, alternatives, and constraints
2. Identify and resolve risks
3. Evaluate alternatives

4. Develop deliverables
5. Plan the next iteration
6. Commit to an approach for the next iteration

Early iterations are the “cheapest”. Number of spirals is variable




Advantages:

- Risk orientation provides early warning
    - Early indication of fatal risks
- Flexible

Disadvantages

- More complex
- Requires attentive, and knowledgeable management

**With the purpose of correcting the major weaknesses in the pure waterfall model, minor modifications were introduced: Sashimi**

Stronger degree of overlapping phases. Advantages:

- Reduces overall schedule
- Reduces documentation
- Works well with personnel continuity

Disadvantages

- Milestones more ambiguous
- Progress tracking more difficult
- Communication can be more difficult

When:

- With small, and well defined projects

**Evolutionary prototyping**

- Begin with the most visible aspects
- Develop system concept as the project progresses
- Prototype

When

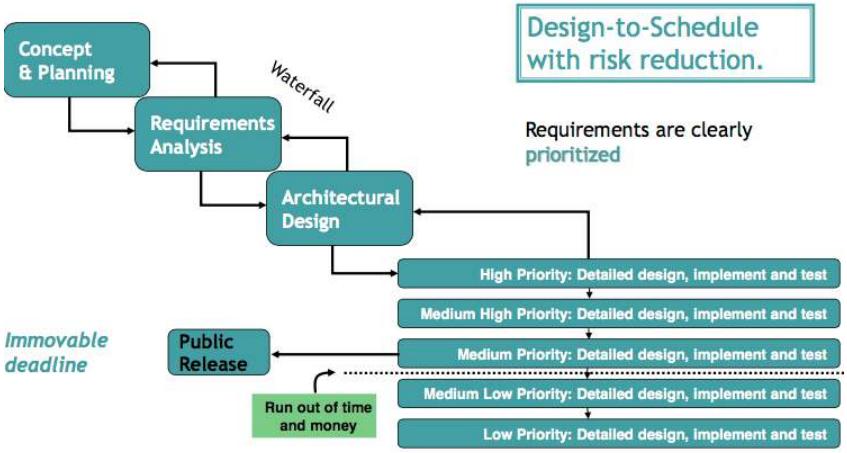
- It is good for situations with:
    - Rapidly changing requirements

    - Customer is reluctant to commit to a set of requirements
    - Vague problem domain
    - Provides steady, visible progress

Disadvantages

- Time estimation is difficult
    - Project completion date may be unknown
- An excuse to do “code-and-fix”

**Staged delivery**




Advantages

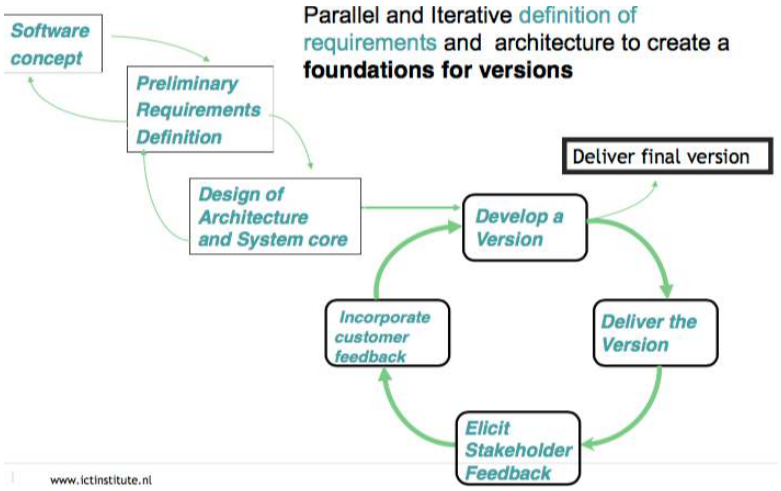
- Customer get product much sooner
- Tangible signs of progress sooner
- Problems discovered earlier
- Increases flexibility
- Reduces the integration and testing effort

Disadvantages

- Possible risk of feature creep
- Requires a careful planning at
    - Management level: the stages are meaningful to the customer
    - Technical level: Dependencies between different components of the product

Your delivery is in successive stages throughout the project. You know exactly when you set out to build it.

**Evolutionary delivery lifecycle model**




**Choosing an appropriate lifecycle**
How well are requirements understood, how well is the system architecture understood, how much reliability is needed, how likely are future revisions, how much risk, need to make midcourse corrections, need to provide visible progress to customers, need to provide visible progress to management, how sophisticated is the model

**Benefits of a lifecycle**
Streamline project, improve development speed, improve quality, improve project tracking and control, minimize overhead, minimize risk exposure, improve client relation

**Make the quality aspect more concrete**
**-**Quality aspects are not very precise or clear by themselves alone. Make quality aspects more concrete by translating them into non-functional (quality) requirements. Be as elaborate as required but not more elaborate than needed.

Non-functional requirements need to be SMART too. Specific non-functional requirements have:
- A value e.g. 10, 20, etc.
- A measurement unit
- A comparison

Examples of non-specific non-functional requirements:
- The system should be usable
- The system should perform fast
- The system should be power efficient

Examples of specific non-function requirements
- 90% of the intended users should rate
- The system should perform 80% of the functions within 1 second

Not all non-functional requirements are black or white. Sometimes a requirement is all or nothing. More often it is not

**MoSCoW and non-functional requirements**

- Must have; requirements labelled as MUST are critical to the current delivery timebox in order for it to be a succes
- Should have; requirements labelled as SHOULD are important but not necessary for delivery in the current delivery timebox
- Could have; requirements labelled as COULD are desirable but not necessary, and could improve user experience or customer satisfaction for little development cost
- Won't have; requirements labelled as WON'T have been agreed by stakeholders as the least-critical, lowest-payback items, or not appropriate at that time

**Measurable non-functional requirements**

- Non-functional requirements should be verifyable
- Therefore one should know how one can measure if the system performs within boundaries
- Sometimes it is easy to determine if a requirement is met, sometimes not

A quality section of the project management plan should not only contain the requirements, but also the planning of the quality.

- Write down when –during which iteration(s) – the requirement will be tested
- Write down who is responsible for the testing team

4: Effort estimation

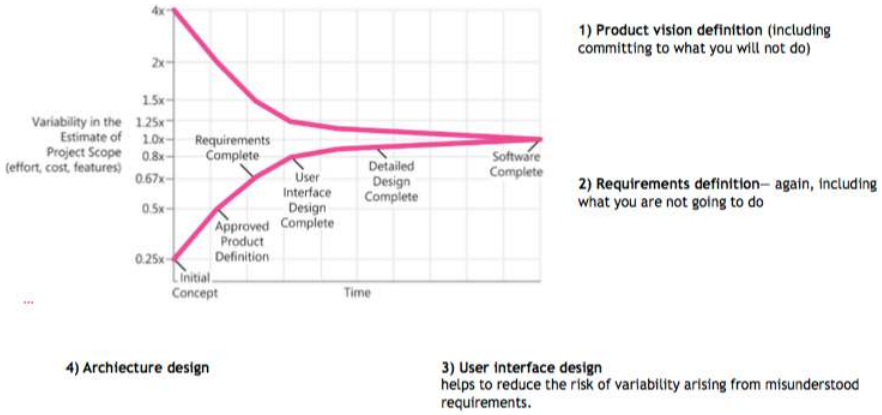
**Where are estimates done?**

Reasons of program management:

- Strategic planning
    - Project portfolio management
- Feasability study
    - Benefits of potential system will justify the costs
- Evaluation of suppliers' proposals

Reasons of project management

- Time of estimation depends on project ilfe-cycle
    - Plan-driven approaches after requirements
    - Agile approaches throughout the life-cycle




**Classification of estimation methods**

- Expert opinion: real experts produce good, but non-objective estimates (depends on hard-to-quantify expertise)
- Top-down (analogous): case-based, comparative
    - 1. Produce overall estimate using effort driver(s)
        - Uses information (database) about past projects, their aspects that influence effort (effort drivers; e.g. technology, complexity, team size, etc.) and their costs
        - Identify effort drivers of current project
        - Estimate current project effort analogous to effort of past projects that are similar
    - 2. Distribute proportions of overall estimate to components
    - Example: top-down estimation. Assume that you have information for matching on the basis of two parameters:
        - **The number of input screens**
        - **The number of report outputs**
    - The new project (**target cause**) requires **7 input screens and 15 report outputs**. You found two past cases (source causes):
        - **Project A**: has 8 input screens and 17 report outputs
        - **Project B**: has 5 input screens and 12 report outputs
    - Which is a more closer match for the new project, A or B?
        - Distance between new project and project A:
            - **Square-root of ((7-8)^2 + (15-17) ^ 2) = 2.24**
        - Distance between new project and project B:
            - **Square-root of ((7-5)^2 + (15-12) ^2) = 3.6**
        - These are the Euclidean distances
        - Project A is a better match because it has less distance than project B to the new project

- Bottom-up: Activity based, analytical
    - 1. Break project into smaller and smaller components
    - 2. Stop when you get to what one person can do in one/two weeks
    - 3. Estimate costs for the lowest level activities
    - 4. At each higher level calculate estimate by adding estimates for lower levels
- Parametric or algorithmic models: uses project characteristics (e.g. size -> function points)

**Bottom-up versus top-down**

Bottom-up:

- Use when you have no data about similar past project
- Identify all tasks that have to be done – so quite time-consuming

Top-down:

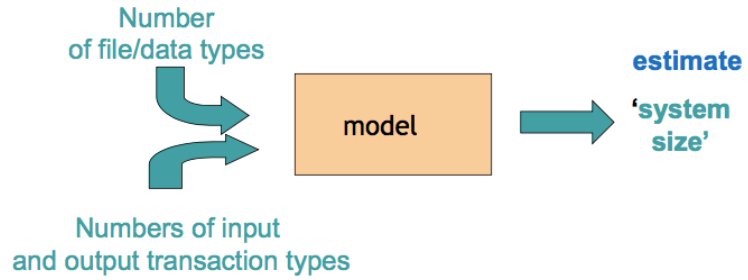
- Based on past project data
- Produce overall estimate based on project cost drivers
- Divide overall estimate between jobs to be done

**Parametric estimation models**

**Aspects of project -> =>('magic') -> effort -> cost**

Aspects of project: What is known about the project at the beginning? **The scope (functional requirements)**

Magic: How to quantify user stories/functional requirements. **Functional Size Metrics** (a.k.a. Function Points)



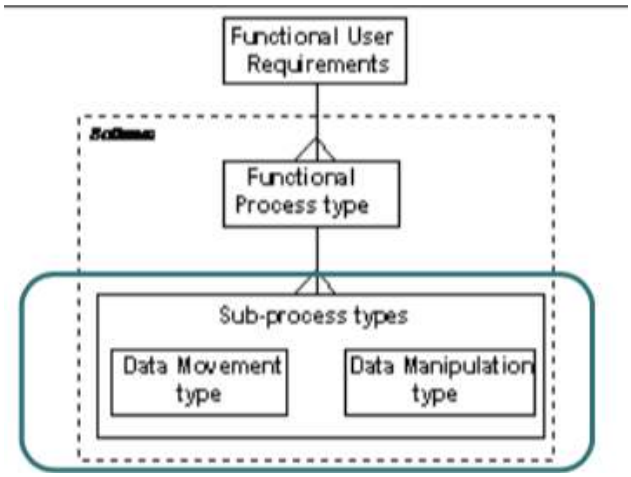

What is **FSM**?

- Functional Size: a size of the software derived by quantifying the Functional User Requirements
- Functional Size Measurement (FSM): the process of measuring Functional Size

There are many different 'flavours' of FSM, but COSMIC function points is suggested.

**COSMIC-FPP**

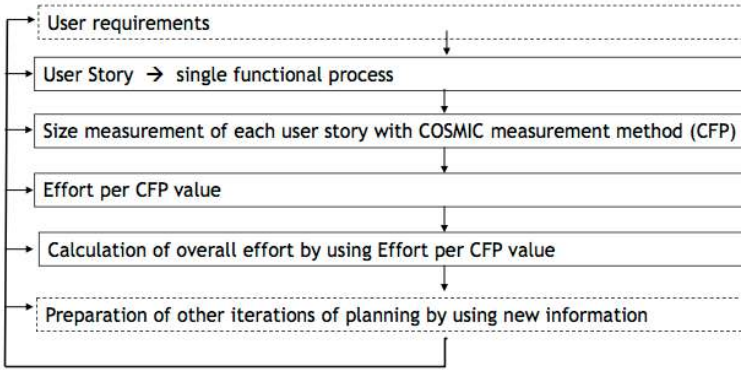
Each functional process is a unique set of sub-processes performing either data movements or data manipulation




Functional user requirements: user stories/functional requirements
Functional process type: Use case functional process (parts of a user story that does one thing)

**Counting COSMIC function points**

- Establish Functional Processes
- Determine the data movements
  - # **E**ntries
  - # **W**rites
  - # **R**eads
  - # e**X**its
- Each data movement is scored
  - Entry 1 CFP
  - Write 1 CFP
  - Read 1 CFP
  - eXit 1 CFP
- A data movement can be identified alone
- Each data movement is assigned a single unit of measure of 1: 1 data movement = 1 COSMIC Function point or CFP, also called the Cfsu (cosmic functional size unit). The total size of the software = addition of all data movements.



User stories: IT = Iteration, US = user story. E.g. IT1-US1: First user story of first iteration

Having converted the functional requirements into user stories, we need to convert user stories info functional processes. One functional process occurs when a user of the system/other system generates a group of data and sends it to the system as a single event. For every different way a group of data is sent (each different event), if a user story consists of more than one event, it has more than one functional process. E.g. entering data on two different forms and sending them to the server to perform a user story.

**Size measurement of each user story/functional process with COSMIC measurement method (CFP)**

For each functional process, perform the following steps:

- Entry: count the amount of different type of data groups that enter the system. This is usually the data sent by actor that initiates the functional process.
- Exit: count the amount of different type of data groups that leave the system. This is usually the data sent back to the actor as a result of the functional process.
- Read: count the amount of different type of data groups that are read from permanent storage. Data that is read does not leave the system (boundary). Data is typically read from a file or retrieved from a database.
- Write: Count the amount of different type of data groups that are written to permanent storage. Data is written does not leave the system (boundary). Data is typically written to a file or stored in a database.

Since we need to count each different type of data group entering and leaving the system, we do not count the data (group) twice if we send two records of the same type in one go, e.g. if we submit two records containing movie-info it still counts as one.

A data group is the information about one thing of interest. So we are talking about 'things' e.g. people, movies, etc. and not attributes e.g. name, age. Typically data group translates in a single record n a database or in a single class in an object oriented program.

**Effort = CFP x hours/CFP**

Depends on technology and experience. Typical 4.0 hours/CFP [3.7-4.5 hours CFP is reasonable]. Depends on technology and market. Typical 65-110€ an hour in NL.

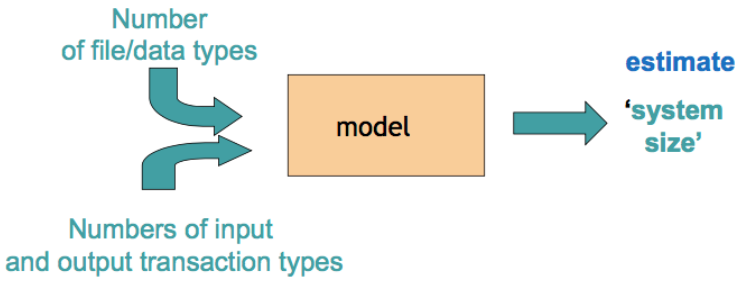
**Cost = hours x hourly rate**

**COSMIC-FPP Advantages**

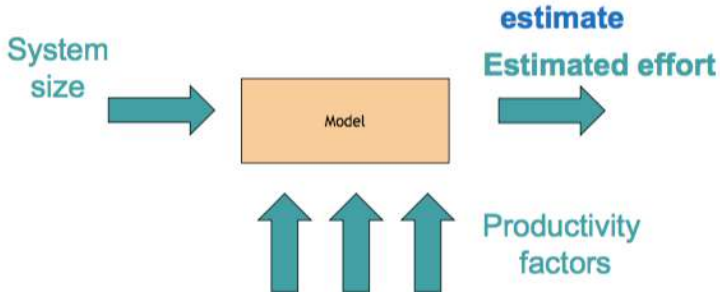
- Applicable **early** in the software lifecycle
    - Specific guidelines for measuring user stories

- **Independent** of software development methods and technology
  - Measurement procedures for methods based on UML, BPML
- Covers very different **software domains**
  - Real time systems
  - Management information systems
- Based on objective criteria
  - Number of data movements

**Parametric models**



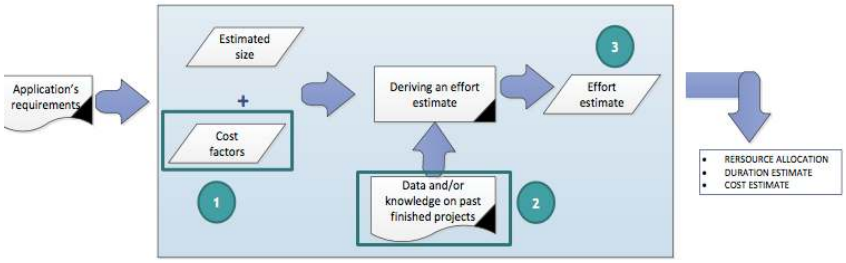

Problem: "bigger projects are more expensive". Relation size and effort not linear.
Problem: "more complex problems are more expensive". Not all projects are equal.
Other models focus on productivity: e.g. COCOMO. System size -> an input




COCOMO parametric estimation
Size of system in Lines of Code + characteristics -> Effort estimate. LOC -> effort using COCOMO formula -> Cost = hours x hourly rate

Problem: how do we know lines of code: Function points. Backfiring: CFP -> Lines of code. Back-firing to get from function points to lines of code. Conversion from function points to lines of code is programming-language dependent.

**Effort estimation process**




**Basic COCOMO**

Basic model:

- **Effort (E) = A x size^B (person-months)**
- A and B depends on **the type of system**: organic, semi-detached, embedded
- Size is measured in 'kloc' **(thousands of lines of code).**
- **Development time (D) = i x (Effort)^j (months)**
- **People required (P) = E/D (count)**

**Estimation example**

Let's develop a 100 function point (CFP) embedded system, written in the C language.

- First convert the function points to lines of code
- Look up in the QSM backfiring table the lines of code per CFP for the language
    - QSM table: on average 97 lines of code in C per function point
- Then convert the CFPs to LOCs
- Then divide by 1000 to get KLOCs
- Then look up the COCOMO constants
    - Embedded development
    - A = 3.6
    - B = 1.2
- Apply COCOMO formula effort = A * (size)^B

| | | |
|---|---|---|
| Size in function points | 100 CFP | |
| Chosen programming language | C | |
| Back-firing constant | ... LOC/CFP | |
| Estimated size in lines of code | | ... LOC |
| Estimated size in kilo lines of code | *devide by 1.000* | ... KLOC |
| | | |
| COCOMO project tyoe | ... | |
| COCOMO A-parameter | | |
| COCOMO B-parameter | | |
| COCOMO estimate (in months of work for 1 person) | A x size $^B$ | ... PMs |

| | | |
|---|---|---|
| **Size in function points** | 100 CFP | |
| **Chosen programming language** | C | |
| **Back-firing constant** | 97 LOC/CFP | |
| **Estimated size in lines of code** | | 9.700 LOC |
| **Estimated size in kilo lines of code** | *devide by 1.000* | **9.7 KLOC** |
| | | |
| **COCOMO project tyoe** | ... | |
| **COCOMO A-parameter** | | |
| **COCOMO B-parameter** | | |
| **COCOMO estimate (in months of work for 1 person)** | A x size $^{B}$ | ... PMs |

| | | |
|---|---|---|
| **Size in function points** | 100 CFP | |
| **Chosen programming language** | C | |
| **Back-firing constant** | 97 LOC/CFP | |
| **Estimated size in lines of code** | | 9.700 LOC |
| **Estimated size in kilo lines of code** | *devide by 1.000* | **9.7 KLOC** |
| | | |
| **COCOMO project tyoe** | Embedded | |
| **COCOMO A-parameter** | 3.6 | |
| **COCOMO B-parameter** | 1.20 | |
| **COCOMO estimate (in months of work for 1 person)** | A x size $^{B}$ | ... PMs |

| | | |
|---|---|---|
| **Size in function points** | 100 CFP | |
| **Chosen programming language** | C | |
| **Back-firing constant** | 97 LOC/CFP | |
| **Estimated size in lines of code** | | 9.700 LOC |
| **Estimated size in kilo lines of code** | *devide by 1.000* | **9.7 KLOC** |
| | | |
| **COCOMO project tyoe** | Embedded | |
| **COCOMO A-parameter** | 3.6 | |
| **COCOMO B-parameter** | 1.20 | |
| **COCOMO estimate (in months of work for 1 person)** | A x size $^{B}$ -> $3.6 * (9.7)^{1.20}$ = 55,00 | **55 PMs** |

## COCOMO II

There are different COCOMO II models for estimating at

- **'early design' stage**

    - counting unadjusted function points
- **'post architecture' stage**
    - lines of code counting rules

The core model is:

- **pm = A(**size**)^(sf) x (em1) x (em2) x (em3)...**
- Where:
    - pm = person months
    - A = constant (e.g. 2.94)
    - size = number of thousands of lines of code (KLOC)
    - em = cost driver effort multiplier
    - sf = the exponent scale factor (sum of project scale factors)

**COCOMO II scale factor**

Based on five scaling drivers that appear to be particularly sensitive to system size:

- **Precendentedness (PREC)**
    - Degree to which there are past examples that can be consulted
- **Development flexibility (FLEX)**
    - Degree of flexibility that exists when implementing the project
- **Architecture/risk resolution (RESL)**
    - Degree of uncertainty about requirements
- **Team cohesion (TEAM)**
- **Process maturity (PMAT)**
    - Could be assessed by CMMI

5: Activity planning

**Projects and activities**

A project is composed of a number of interrelated activities:

- A project may start when at least one of the activities is ready to start
- A project will be competed when all of the activities it encompasses have been completed

For a project to be predictable and planable:

- An activity must have a clearly defined start and a clearly defined end-point. Normally marked by the production of a tangible deliverable
- If an activity requires a resource then that resource must be forecastable
- The duration of an activity must be forecastable
- Some activities might require that others are completed before they can begin. These precedent requirements must be identified

**Identifying activities**

Three approach exist to identify a project's activities:

- The activity-based approach

    - Creating a list of all activities
    - Can be structured using a work breakdown structure
- The product-based approach
    - Creating a list of all products that are required
    - Can be structured using a product breakdown structure and product flow diagram
- The hybrid approach
    - Contains elements of the activity-based approach and the product-based approach

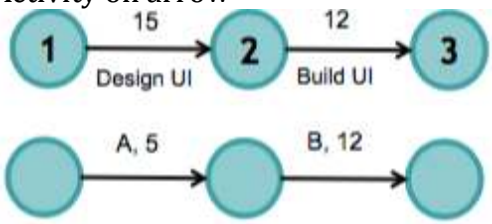
**Networks diagrams**

These help us to:

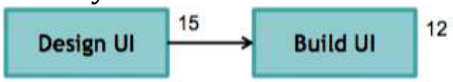
- Assess the feasibility of the planned project completion date
- Identify when resources will need to be deployed to activities
- Calculate when costs will be incurred
- This helps the co-ordination and motivation of the project team

Activities and their interrelationships as a graph; two forms of visualisation

- Activity on arrow



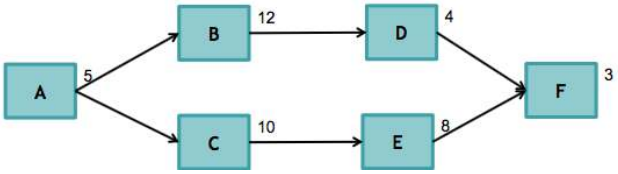

- Activity on node




Each activity labelled with identifier (usually a letter/code), duration (in standard unit like days). Time goes from left to right. There is one start & one end event.

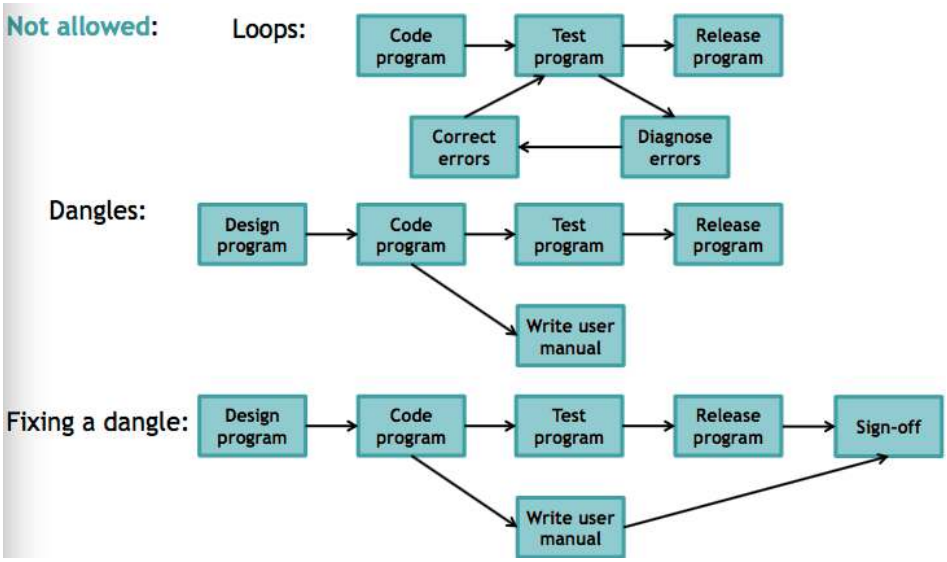
| Activities | Precedents | Duration |
|---|---|---|
| A | | 5 days |
| B | A | 12 days |
| C | A | 10 days |
| D | B | 4 days |
| E | C | 8 days |
| F | D,E | 3 days |

Drawing the network diagram



Rules on constructing a precedence network:

1. Project network should have only one start activity
2. Project network should have only one end activity
3. An activity has duration
4. Precedents are the immediate preceding activities
5. Project network may NOT contain loops
6. Project network should not contain dangles




Rest: slides

6: Organizational aspects

**In scope/out scope**

- In scope:
  - Detect offsides
  - Send signal to the referee
  - Interface
  - Provide a log with records of the playtime the system detected an offside
- Out of scope:
  - It does not make the call for the offside
  - It does not detect any other infringements, fouls or rules
  - The hardware of the system is provided by third parties

## Deliverables

| Name of Deliverable | Description | Due Date |
|---|---|---|
| Ball and Referee Recognition Phase | This phase will work on the functional requirements involving the ball and the referee. | 13/09/2016 |
| Player Recognition Phase | This phase will work on the functional requirements involving the players. | 27/10/2016 |
| Offside Detection Phase | This phase will work on finalizing the functional requirements for actually detecting the offsides. | 24/11/2016 |
| Quality Requirement Testing | This phase will test if the system is in accordance with the quality requirements previously decided upon. | 14/04/2017 |

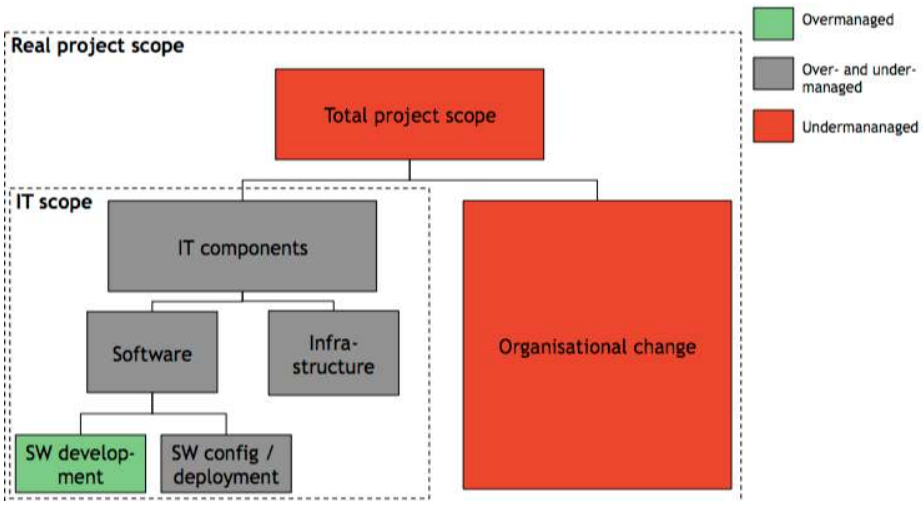
**Success rates** are hard to benchmark. It is hard to scientifically measure IT project success rates. Prof Verhoef from the VU has tried to validate the chaos reports, and found that there are significant measurement problems:

- Expectation management is just as important for success as IT expertise
- People manage towards budget rather than failure
- Only classic waterfall projects deliver "all or nothing": modern projects are more complex

**There are no IT projects:**

- Companies do not want software. They want people to work together in a certain way to achieve business results. As a consequence, most projects are about changing the way people work together to achieve different results, not about software or IT. As a further consequence, there are no IT projects, just projects with an IT component.

The difference between projects and software development projects




Example: on day 1 company XYZ orders a new inventory and finance IT systems. You, the vendor, have done the exact same project just a few weeks ago are able to

deliver perfect software on day 2. On day 3 the company is in chaos and the CEO is extremely unhappy. What could have gone wrong?

**Possible reasons:**

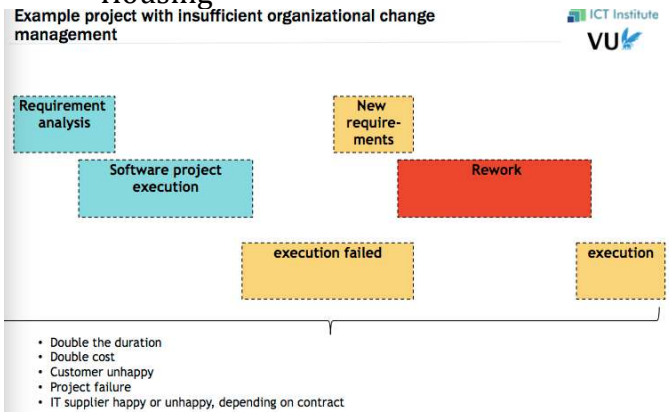
- Disruption of work for installation/deployment
- The new software needed data to run. The data was and is not available
- Roles/functions of people:
    - Current functions of people
    - New functions emerge
    - Old functions disappear
- No worker union/employee council consult
- Legal issues, e.g. tracking employees, violation of privacy laws
- No plan for phasing out old system

**Further problems:**

- Organizations misuse IT project/systems to constrain people to work in a certain way
- Other projects ask people to share all their knowledge for free so that they can be replaced by computers

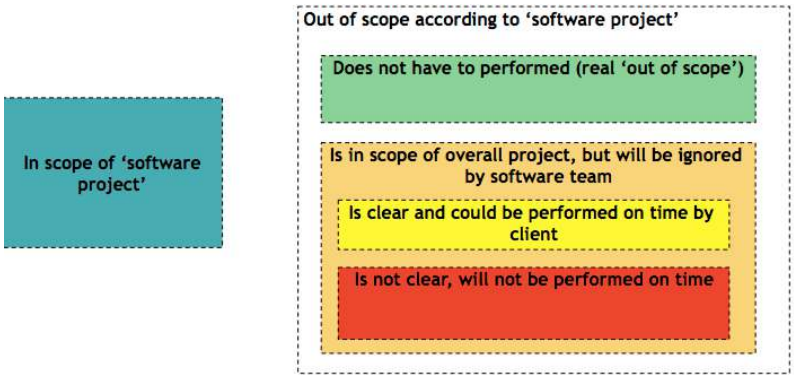
**Solution: COPAFITLSH:**

- Communication
- Organizational structure
- Personnel
- Administrative organization
- Finance
- Information flow
- Technology
- Legal aspects
- Security
- Housing




Total project plan = non-IT project plan(COPAFLSH) and IT project plan (T with some I).

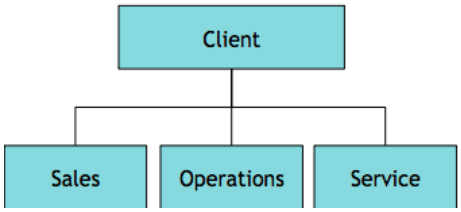
Scope management: the wrong way




**Communication**:

- Inform all staff that a project will be started
- Keep organization informed on progress and delays
- Inform all staff beforehand when they will be involved
- Allow all staff to provide input or feedback after they have received detailed information on changes
- Inform staff how their feedback has been incorporated in the project

**Organizational structure**




- Do you know all the staff on the organization that will use your software?
- Will students, customers, suppliers us it as well?
- Will the organization change as part of the project plan? Has a decision been made what it will be?

**Personnel:**

- Is everyone informed?
- Did you ask everyone for feedback?
- Do people want training? Do they need training?
- Are people involved in requirements? How/why not?
- Are people involved in testing? How/why not?

**Administrative:**

- Who maintains the data in the system?
- Who makes and updates the user accounts for the system?
- Who will be the helpdesk for the system? Who are the second line experts?

**Finance:**

- Does the new/changed software need a maintenance budget?
- Are there license fees and hosting fees? Who pays these?
- Which department owns the system? Who decides on future projects?
- Should it be in the books? What is the value?

**Information flow**

- Most IT is used to support information flows. This is basically the IT scope, but at a high level understandable for the non-IT users
- Describing the information flows should help understand which IT systems need to be created, changed, or connected

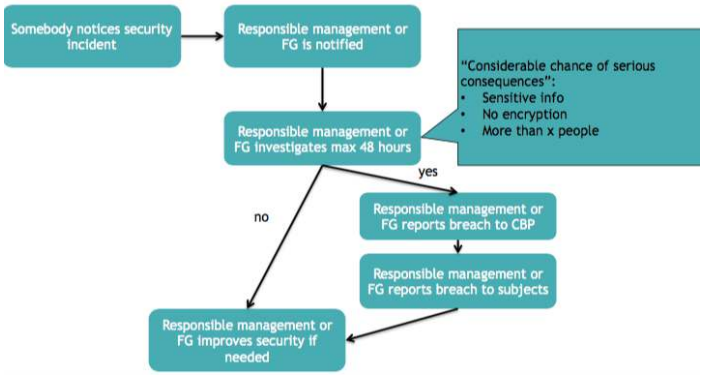
**Technology**

- Identifying the technology early on is important for planning purposes, in order to reserve the right resources
- Technology choice is also important for the handover from project to maintenance
- Licensing cost can be zero to very expensive, should be checked upfront

**Legal aspects**

- Copyright
- Open source
- Ownership
- Privacy

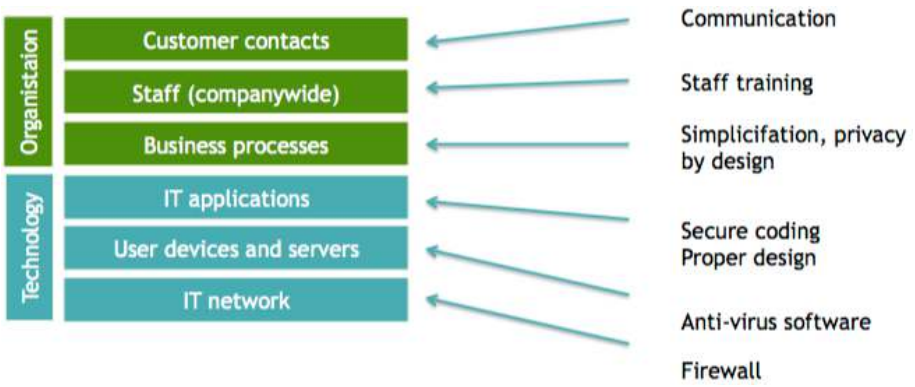
**Reporting Data Breaches**




**Security:**

- One must take suitable/reasonable technical and organizational measures to prevent loss of data or illegal processing of personal data
- Note: one can take cost and relative importance of data into account to determine what is suitable/reasonable

**IT security: layered approach**




**Housing**

- Does the project need a project room/floor/building?
- Does the system need dedicated hosting?
- Does the project success create new roles or jobs? Where will these people work?

**Why should you do SCOPAFITHL?**

It is mandatory:

- It is mandatory for all organizations to involve their worker's council
- It is mandatory for employees to provide proper training
- It is mandatory for companies to keep personal data private
- It is mandatory for organizations that have privacy-sensitive data to have an up to date security plan

7: Risk Management

**What does Risk mean for you and your stakeholders?**

- Risk is the potential of losing something of value. Something of value can be..
    - Money
    - Reputation
    - Continuity of the business
    - Wellness, health and life
- Or even worse
    - To lose more than what you have invested
        - Get a claim
        - Not able to pay your interest/loan/investment
        - Bankruptcy

**Definition of Risk score:**

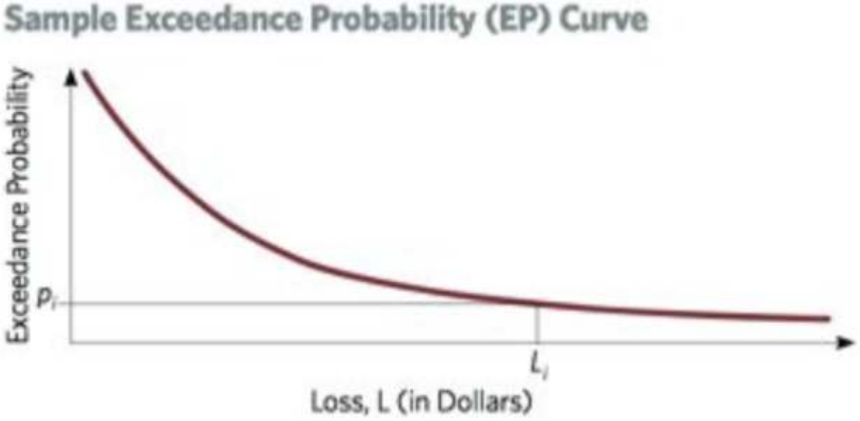
RISK = IMPACT * PROBABILITY

- Higher risk score means that you are going to want ti mitigate these as much as possible

- Lower risk score means you might want to leave it as it is

**Characteristics:**
- Risks relate to possible future problems, not to current ones
- Risks involve a possible cause and its effect(s)

In general you might see this relationship: higher impact, less probable




**Examples of software risks:**
- Nobody uses/likes your system -> people use workarounds/competitor
- Solving the wrong problem -> no business case
- It is harder than expected, unknown technical effects -> delay of delivery
- The application crashes in production -> lose of customers
- The database gets corrupt -> your business cannot operate anymore
- Someone steals information -> privacy claim

Are risks a problem? Not if you have managed them

**Steps for approaching Risk in projects:**
1. Risk identification – what are risks to a project?
2. Risk analysis – what is the probability and impact on the project?
    a. Qualitative: where descriptive terms are used to describe the impact and probability
    b. Quantitative: where the specific costs are outlined in real terms of financial costs and statistical probability
3. Risk prioritization – which ones are important for you?
4. Risk planning – what shall we do?
5. Risk monitoring – has the planning worked?

**1. Risk identification:**

How to get a risk of risks:
- Common sense, wisdom from the crowd and history
- Check lists available in the industry and science

- Quality models in order to structure reasoning
- Brain storming: sitting in a room with experts

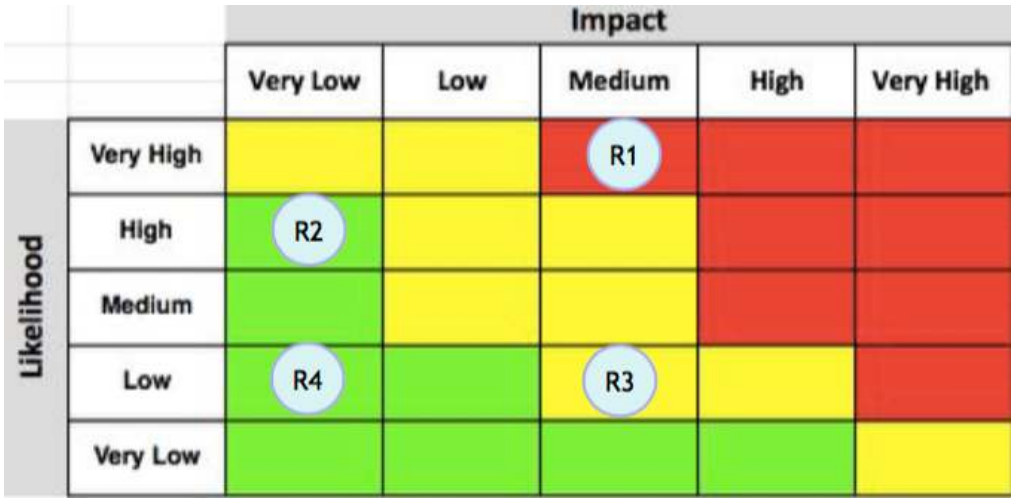
2. **Risk analysis**

- **Qualitative** allows for risks to be quickly categorized and ready for comparison
    - May give a lot of discussion between stakeholders
    - 'Uncountable' aspects have value, like aesthetics, emotions, though hard to explain in scientific and business language
- **Quantitative** ensures risks are fully understood
    - Reduce discussion as you must understand the risk in every detail in order to generate valid metrics
    - Expensive to obtain (i.e. very time consuming to obtain data)

**Qualitative: Probability/impact matrix**

- A probability/impact matrix or chart lists the relative probability of a risk occurring on one side of a matrix or axis on a chart and the relative impact of the risk occurring on the other
- Procedure:
    - List the risks and then label each one as (very) high, medium, or (very) low in terms of its probability of occurrence and its impact if it did occur.

| # | Description |
|---|---|
| R1 | Changes to requirements specification during coding |
| R2 | Specification takes longer than expected |
| R3 | Significant staff sickness affecting critical path activities |
| R4 | Significant staff sickness affecting non critical path activities |



**Some remarks**

- The qualitative approach provides most times a good impression of the risks
  - Requires low effort to construct risk evaluation
- The quantitative solution predicts the risks and costs of the options, especially if you have to perform the task multiple times
  - No better way to convince stakeholders, at least, it kills discussion
- The quantitative option is expensive and not always possible:
  - It works great for deterministic systems like train schedules
  - You need to collect a lot of data to filter out the effect of individual cases
- Finally: nobody can predict the future

### 3. Risk Prioritization

- When you have an overview of the risks, and risk score, you can order them
  - The first risk is the most important for project success
  - The risk mitigation effort and resources of the project should go mainly to the first one

| # | Description | Weight (Impact * Prob) |
|---|---|---|
| R1 | Changes to requirements specification during coding | 200 |
| R3 | Significant staff sickness affecting critical path activities | 75 |
| R2 | Specification takes longer than expected | 15 |
| R4 | Significant staff sickness affecting non critical path activities | 5 |

  - Note: it is fundamentally impossible to mitigate all risks as they might be conflicting. For example, security versus usability. The latter will suffer from the security measures.

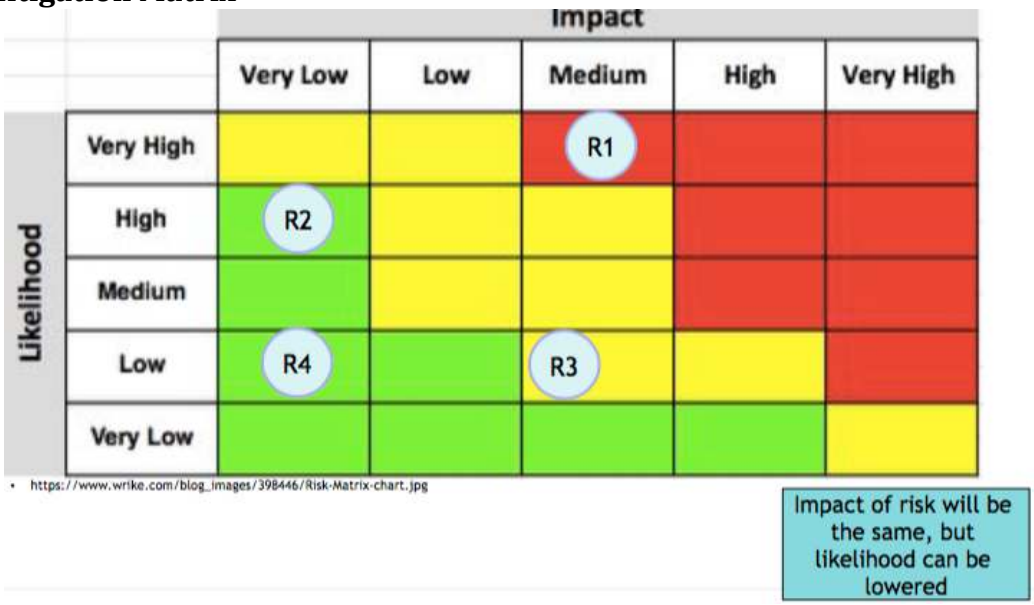
### 4. Risk Planning

- Keep the collected risks and weight in a register
- Assign a mitigation action to each risk
- Agree which mitigation should be implemented
  - Based on the costs, impact of mitigation and risk exposure
- Risk exposure is defined as potential damage * probability
  - For example: RE = 200000 * 0.1 = 2000

### 5. Risk Overview

| Nr | Risk | Description | Likelihood (1-20) | Impact (1-20) |
|---|---|---|---|---|
| 1 | Changes | Changes to requirements specification during coding | 20 | 10 |
| 2 | Overdue | Specification takes longer than expected | 15 | 1 |
| 3 | Sickness critical | Significant staff sickness affecting critical path activities | 5 | 15 |
| 4 | Sickness non critical | Significant staff sickness affecting non critical path activities | 5 | 1 |
| 5 | | | | |
| 6 | | | | |
| 7 | | | | |

**Risk Mitigation Matrix**




**Software project risks top ten of Boehm**

| Risk | Risk Mitigation Approach |
|---|---|
| Personnel shortfalls | Staffing with top talent; job matching; teambuilding; training and career development; early scheduling of key personnel |
| Unrealistic time and cost estimates | Multiple estimation techniques; design to cost; incremental development; recording and analysis of past projects; standardization of methods |
| Developing the wrong software functions | Improved software evaluation; formal specification methods; user surveys; prototyping; early user manuals |
| Developing the wrong user interface | Prototyping; task analysis; user involvement |
| Gold plating | Requirements scrubbing, prototyping, design to cost |

| Risk | Risk Mitigation Approach |
|---|---|
| Late changes to requirements | Change control, incremental development |
| Shortfalls in externally supplied components | Benchmarking, inspections, formal specifications, contractual agreements, quality controls |
| Shortfalls in externally performed tasks | Quality assurance procedures, competitive design etc |
| Real time performance problems | Simulation, prototyping, tuning |
| Development technically too difficult | Technical analysis, cost-benefit analysis, prototyping, training |

## 6. Risk Monitoring

- Involves executing the risk management process to respond to risk events
    - Track the status of each risk
    - Respond to triggers, symptoms and indicators of actual risk events
- This is an ongoing activity – new risks will be identified, old risks disappear, weaken or get stronger
- Workarounds are unplanned responses to risk events that must be done when there are no contingency plans
- Main outputs of risks monitoring and control are:
    - Requested changes
    - Recommend corrective and preventive actions
    - Updates to the risk register, project management plan

**Course Summary**

**SPM Fundamentals**

- A project should have a vision and goal
- A project is executed in the expectation of some benefit in the future
- A project is most times a new endeavour, so a methodology is needed to reduce the risks: Waterfall vs Agile
- A project involves multiple stakeholders, and all their stakes need to be taken into account to achieve success
- A project plan is for communication and should contain:
    - The vision
    - What
    - How
    - Milestones and KPI's
- The scope is about: what will the

**SPM Scope Management**

- The scope is about: what will the project produce in the end
- Stakeholders need to know and agree what will be IN the scope and what will be OUT of the scope of the project
- Project manager needs to manage changes in scope over time and control scope creep
- The balance can be informally defined as Quality = scope * time * money
- Use user stories to define functional features: As a <ROLE> I can <WHAT> so that <WHY>
- Define quality criteria for non functional requirements: You can use predefined quality models like ISO 25010

**SPM Lifecycle Planning Including Agile**

- Lifecycle: period of time that begins when a software product is conceived and ends when the product is retired from use
- Model for developing software: pure waterfall, code and fix, spiral, modified waterfall, evolutionary prototyping
- Choosing model based on reliability needed, future revisions, how well the project is understood and how progress needs to be tracked
- Quality aspects ripple through the system: make a prioritized list (MoSCoW) of quality requirements and how to track them (SMART)

**SPM: Effort Estimation**

- Why: to predict costs for strategic planning, feasibility studies and evaluation of suppliers' proposals
- How: expert opinion, top-down (case based), bottom-up (activity based), parametric/algorithmic models (COSMIC-FP, COCOMO)
- Use bottom-up if you have no data
- Use top-down if you already have data and/or experience
- Use COSMIC-FP if you like some magic, and have a clear scope definition

- Use COCOMO if you know lines of code (from COSMIC-FP)
- COSMIC-FP: User stories => Function points (FP) => € = FPs * Hours per FP * rate

**SPM Activity Planning**

- Build network diagrams, determine critical path and calculate float of activities
- Gives insight in dependencies of tasks:
    - Which needs to be executed sequentially
    - Which can be executed in parallel (scalability of your project!)
- A network diagram gives structure to your work process and see interdependence
- Add time annotations to your network nodes to estimate duration and slack
- Gantt charts help to structure the dependencies, timelines and milestones

**SPM Organizational Aspects**

- Companies do not want software. They want people to work together in a certain way to achieve business results
- COPAFITLSH model contains all aspects to on board complete organization: Communication, Organizational Structure, Personnel, Administrative organization, Finance, Information flow, Technology, Legal aspects, Security, Housing
- Important: keep all staff informed and gather early feedback
- Have a clear vision who is responsible for the operational data
- How are costs allocated, or are costs activated (is the system an asset) How is maintenance financed?
- Which technology stack: also think about recruitment
- Legal: hidden license fees, patents and/or open source licenses, and privacy
- Security: managing the risks of loss of data or illegal processing

**SPM Risk Management**

- Risk identification – What are the risks to a project?
- Risk analysis – What is the probability and impact on the project?
    - Qualitative: where descriptive terms are used to describe the impact and probability
    - Quantitative: where the specific costs are outlined in real terms of financial costs and statistical probability
- Risk prioritization – which ones are important for your project?
- Risk planning – which risk should you mitigate and how?
- Risk monitoring – has the planning worked