**POM 15 – Estimation and Scheduling**

**Estimation and Scheduling (Schätzung und Planung)**

*Facts about Estimates:*

* Nearly 2/3 of projects significantly overrun their cost estimates
* The average project exceeds its schedule by 100%

*Challenges for Estimation:*

* **Incomplete knowledge** about:
  + Project scope and changes
  + Prospective resources and staffing
  + Technical and organizational environment
  + Infrastructure
  + Feasibility of functional requirements
* Comparability of projects in case of new or changing technologies, staff, methodologies
* Learning curve problem

*Components of estimations:*

* **Cost** (Personnel, Material, Extra costs (travel expenses etc.))
* **Scope** (Number of requirements, Complexity of requirements)
* **Time** (Development Time, Project duration, Dependencies)
* **Quality** (Usability, Maintainability, Reliability, ...)

and also…

* **Infrastructure** (rooms, technical infrastructure)



*Estimating Personnel Cost:*

* Personnel type: Team leader, analyst, designer, programmer, tester…
* **Cost rate**: Cost per person per day
* 2 alternatives for cost rate:
  1. Single cost rate for all types
  2. different cost rates to different personnel types based on experience, skills, …
* **Personnel cost**: person days x cost rate

*Estimating Development Time:*

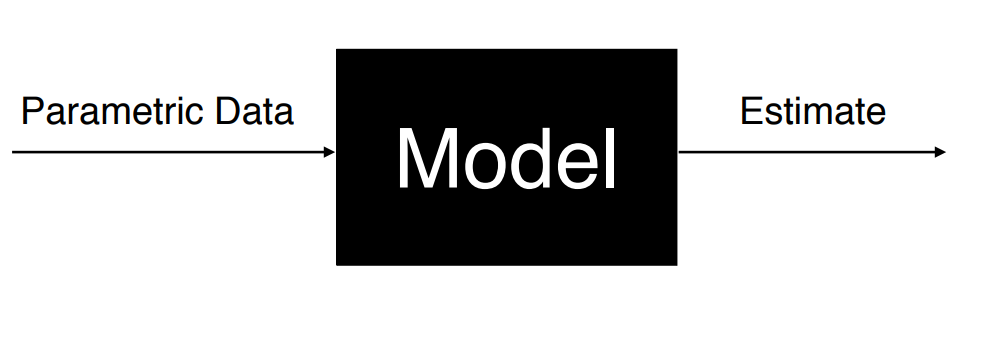
* Development time is often estimated by the formula

**Duration = Effort / People**

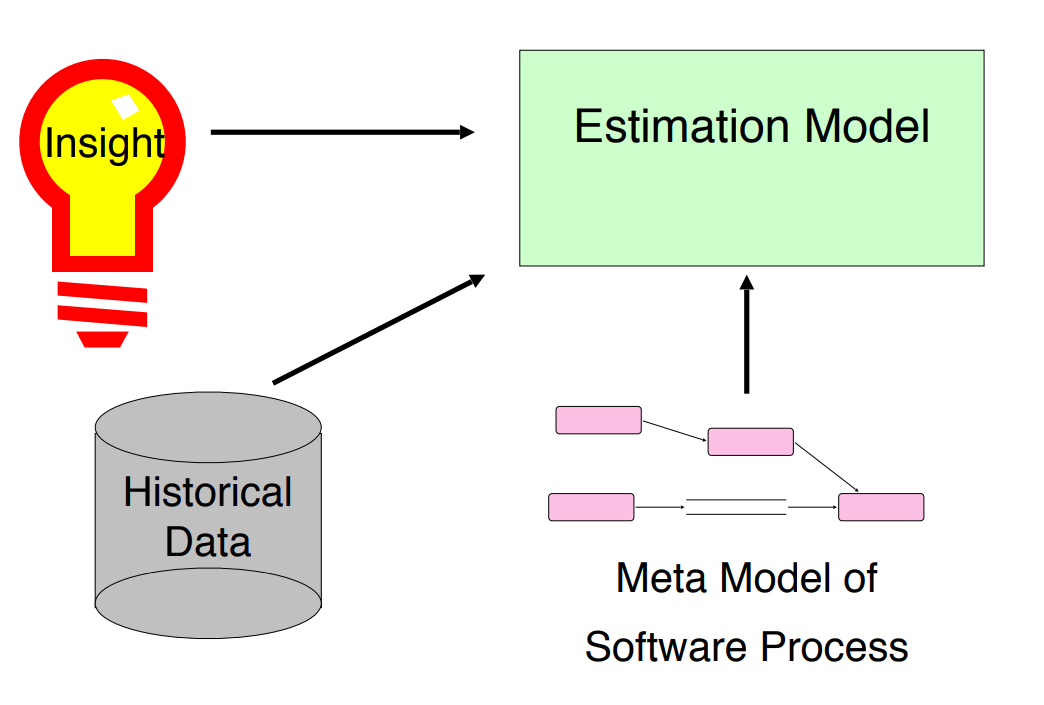
*Problems with this formula:*

* larger project team increases communication complexity which usually reduces productivity
* not possible to reduce duration arbitrarily (willkürlich) by adding more people to a project
* **Brooks Law:** “Adding people to a late project makes it even later”

*Basic Estimation Model:*



*How do you Build an Estimation Model?*



*Top-Down and Bottom-Up Estimation:*

2 common approaches for estimations

1. Top-Down Approach
   * Estimate effort for the whole project
   * Breakdown to different project phases and work products
   * If you did not break down the work before
2. Bottom-Up Approach
   * Start with effort estimates for tasks on the lowest possible level
   * Aggregate the estimates until top activities are reached
   * Preferred if you already have activities and tasks and knowledgable developers

*Methods for estimating cost and effort:*

|  |  |
| --- | --- |
| Expert estimation | Algorithmic estimation |
| Based on:   * Experience * Domain knowledge | Based on:   * Key performance indicators * Formulas * Metrics |
| Exemplary methods:   * T-Shirt sizes * Planning Poker * Assessment meetings | Exemplary methods:   * Lines of Code (LOC) * COCOMO II * Function Points |

*Expert Estimation:*

* Guess from experienced people
* No better than the participants
* Suitable for atypical projects (für atypische Projekte geeignet)
* Justification of the result difficult (Begründung des Ergebnisses schwierig)
* Important when no detailed estimation can be done
* Works best if the estimates are for short term items

*Algorithmic Estimation:*

* Calculations using input variables
* needs information about key perfomance indicators
* easier to justify results
* Suitable for traditional projects

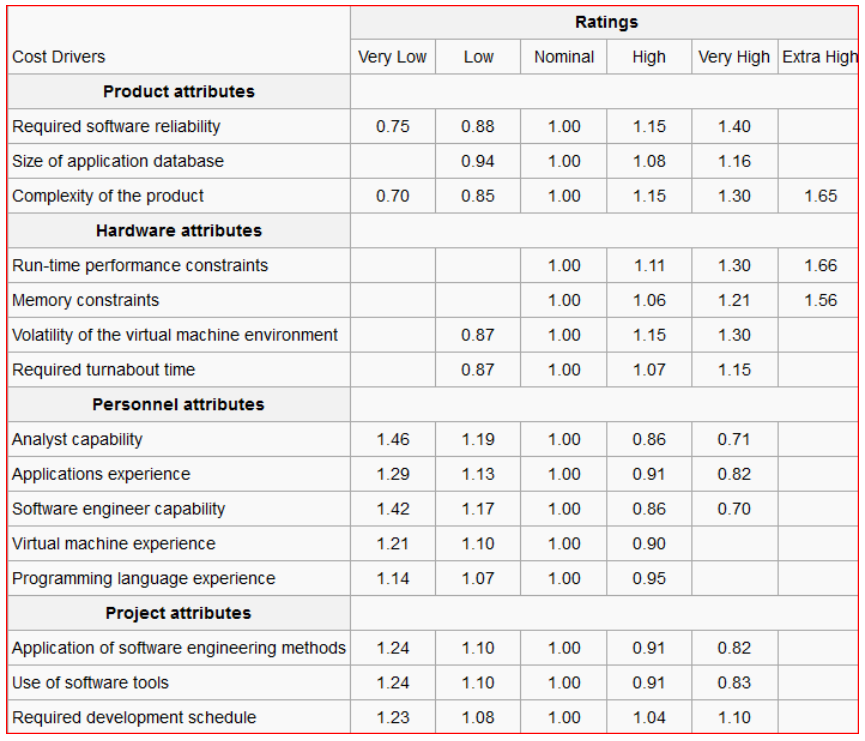
*Lines of Code:*

* Traditional way for estimating application size
* Advantage: easy to do
* Disadvantages:
  + Focus on developer’s point of view
  + No standard definition for “Lines of Code”
  + **programmers ignore reuse and refactorings**
* Caspers Jones: The use of LOC metrics for productivity should be regarded as professional **malpractice**

*COCOMO (COnstructive COst MOdel):*

* Developed by Barry Boehm in 1981
* Top-down approach to estimate cost, effort and schedule of software projects, based on size and complexity of projects
* Assumptions:
  + Derivability of effort by comparing finished projects (“COCOMO database”)
  + System requirements do not change during development
  + Exclusion of some efforts (for example administration, training, rollout, integration)
* COCOMO II: Revision of COCOMO I in 1997
  + Targeted for iterative software lifecycle models such Boehm’s spiral model (COCOMO I was based on the waterfall model)
  + It also addresses team experience, developer skills and distributed development

*Cost Drivers in COCOMO II:*



*Estimation Variability: Cone of Uncertainty:*

*Problems with Traditional Estimation Techniques:*

* Focus on the completion of activities and not on the delivery of features
  + Customers get no value from the completion of activities
* Wrong focus in schedule reviews
  + The reviewers look for overlooked activities, not for overlooked features
* When faced with overrunning a schedule, teams
  + Attempt to save time by reducing quality

*Review: T-Shirt sizes:*

* Estimation is particularly challenging for unexperienced developers and in the beginning of a project
  + Example: The 1st sprint planning usually needs a lot of time due to estimation
* Recommendation: start with a simplified model, e.g. with T-Shirt sizes
* You can map T-Shirt sizes to story points, e.g.
  + S —> 1
  + M —> 3
  + L —> 8
  + XL —> too big, split it into smaller issue
* When the developers get more familiar with estimation and the project itself, move to a more fine-grained estimation model, e.g. Planning Poker

*Planning Poker: an agile estimation approach:*

* Planning poker packages expert estimates into an enjoyable approach to estimating
* All the participants are estimators, in particular all the developers, not just the project manager!
* The estimation team should not have more than 10 people
  + If there are more than 10 people, create two or more estimation teams
  + The product owner can participate in the game, but does not estimate
* Most important aspect of planning poker:
  + Estimates are arrived at by **consensus building**, not by looking at a crystal ball

*Planning Poker Rules:*

1. Start: Each team member is given a set of cards
2. One person reads each of the items to be estimated

* Examples of items: Functional requirements, user stories, scenarios, features, design goals, implementation of algorithms, TODOs

1. The team discusses each item individually
2. Each team member estimates the difficulty of the item and privately selects a card representing the estimate
3. After all team members have chosen their card, everyone shows their chosen card to the others at the same time

* If the team member’s estimates match, the estimate for this item is established (for now)

1. If estimates are not the same, the group discusses the differences

* Steps 4 to 6 are repeated until a consensus is reached
* Important: Do NOT average during planning poker

*Challenges and advantages of planning poker:*

* **Challenges**:
  + based on expert opinion
  + same backlog provided to another Scrum Team, it could come up with estimation that vary drastically different
  + Estimation changes with the different skills and experience
  + Team estimates while individuals are owners
  + “What” factor when not well described makes it harder for the team to estimate.
* **Advantages**:
  + Group Estimating: wisdom of the Crowd
  + The conversation following the revealing of initial estimates is a is a great way to pool important issues
  + Story points can be used to determine the Sprint velocity

*Planning Poker:* ***Practical difficulties:***

* Many people in a team may lead to long discussion
* Lazy developers want to overestimate tasks, in order to have less user stories assigned to them in a sprint
* People tend to want to average the amount of points, which limits the discussion and may not represent the actual difficulty of the task

***Tips*** *for Planning Poker:*

* Keep discussions productive:
* Break out into smaller sessions
* Choose the right time to play
* Write down the reason for the awarded points
* Consider the level of uncertainty
* Distribution of workload according to skill

*Dependency Diagrams:*

**A dependency diagram is a formal notation that helps in the construction and analysis of complex schedules.**

Example:

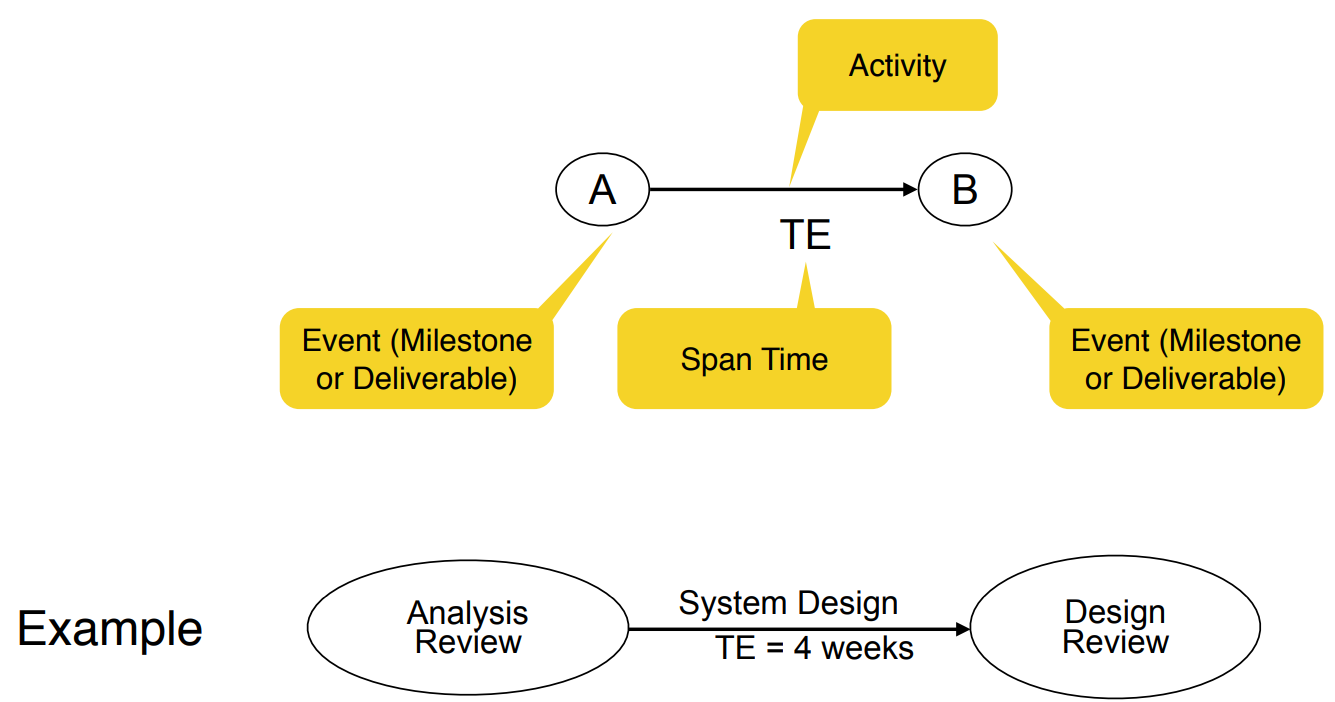
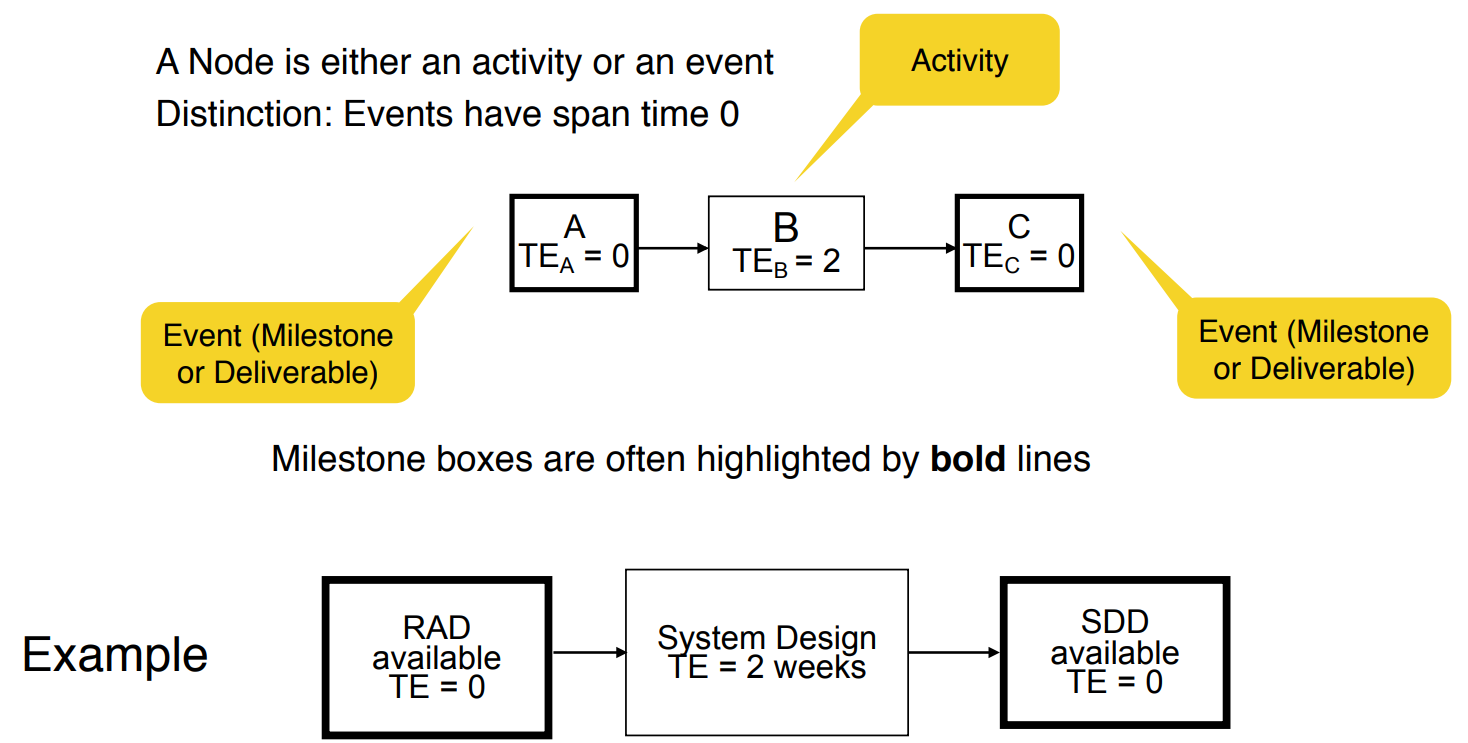
* You are assigned a project consisting of 5 tasks
* Task 1 has to be finished before any other task can start
* Task 2 and 3 can be done in parallel, task 4 and 5 cannot
* Task 5 depends on task 2
* Can the project be finished in 3 weeks, if each of the tasks takes a week to complete?

*Dependency Diagrams Overview:*

* Dependency diagrams consist of 3 elements
  + **Event**: A significant occurrence in the life of a project
  + **Activity**: Amount of work required to move from one event to the next
  + **Span time**: The actual calendar time required to complete an activity
    - parameters: availability of resources, parallelizability of the activity
* Dependency diagrams are drawn as connected graphs of nodes and arrows. 2 commonly used notations are:

1. Activity-on-the-arrow notation
2. Activity-in-the-node notation

*Activity-on-the-arrow Diagram Notation: Activity-in-the-Node Diagram Notation:*

*What do we do with these diagrams?*

1. Compute the project duration
2. Determine activities that are critical to ensure a timely delivery
3. Analyze the diagrams
   * To find ways to shorten the project duration
   * To find ways to do activities in parallel

* 2 techniques are used

1. Forward pass analysis (determine critical path)
2. Backward pass analysis (determine slack time)

*Critical Path and Slack Time:*

* **Critical path**
  + A sequence of activities that take the longest time to complete
  + The length of the critical path defines how long a project will take to complete
* **Noncritical path**
  + A sequence of activities that can be delayed and the project can still finish in the shortest time possible
* **Slack time**
  + The maximum amount of time that you can delay an activity and still finish your project in the shortest time possible

*Analyzing Dependency Graphs:*

* Determination of **critical paths**
  + Compute earliest start and finish dates for each activity
  + Start at the beginning of the project and determine how fast you can complete the activities along each path until you reach the final project milestone
  + Also called **forward pass analysis**
* Determination of **slack times**
  + Start at the end of your project, figure out for each activity how late it can be started so that you still finish the project at the earliest possible date
  + Also called **backward pass analysis**

*Definitions: Start and Finish Dates:*

* **Earliest start date** (ES): The earliest date you can start an activity
* **Earliest finish date** (EF): The earliest date you can finish an activity
* **Latest start date** (LS): The latest date you can start an activity and still finish the project in the shortest time
* **Latest finish date** (LF): The latest date you can finish an activity and still finish the project in the shortest time