Softwaretechnik / Software-Engineering

Lecture 02: Project Management, Cost Estimation

2015-04-27

Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lecture:

- Introduction: Engineering, Quality, Software, Software Specification

This Lecture:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - what characterises a project, life cycle, ...?
  - what is a role, a phase, a milestone, ...?
  - what are common activities and roles in software development projects?
  - what are goals and activities of project management? why project managent?
  - what is COCOMO, what is function points? what is it good for? why to use it with care?
- **Content:**
  - the notion of 'project'
  - project management activities
  - what to manage: activities, people, cost and deadlines
  - cost estimation, project planning

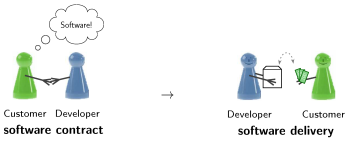
(Software) Project

project – A temporary activity that is characterized by having a start date, specific objectives and constraints, established responsibilities, a budget and schedule, and a completion date.
If the objective of the project is to develop a software system, then it is sometimes called a software development or software engineering project. **R. H. Thayer** (1997)

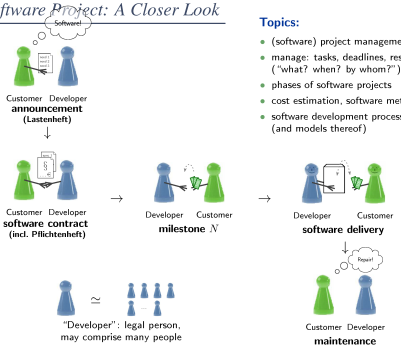
(software) project – characteristics:

- The duration of a project is limited.
- Each project has an "originator" (person or institution which initiated the project). The **project owner** is the originator or its representative. The project leader reports to the project owner.
- Each project has a **purpose**, i.e. pursue a bunch of goals. The most important goal is usually to create or modify software; this software is thus the result of the project, the **product**. Other important goals are extension of know-how, preparation of building blocks for later projects, or utilisation of employees. The project is called **successful** if the goals are reached to a high degree.
- The product has a recipient (or will have one). This recipient is the **customer**. Later **users** belong to the customer.
- The project links people, results (intermediate/final products), and resources. The organisation determines their roles and relations and the external interfaces of the project. **Ludewig & Lichter** (2013)

Software Project: The Very Big Picture



Software Project: A Closer Look



Topics:

- (software) project management
- manage: tasks, deadlines, resources ("what? when? by whom?")
- phases of software projects
- cost estimation, software metrics
- software development processes (and models thereof)

Cycle and Life Cycle

cycle — (1) A period of time during which a set of events is completed. See also: ... **IEEE 610.12** (1990)

software life cycle — The period of time that begins when a software product is **conceived** and ends when the software is no longer **available for use**.

The software life cycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and, sometimes, retirement phase.
Note: These phases may overlap or be performed iteratively. **IEEE 610.12** (1990)

software development cycle — The period of time that begins with the **decision to develop** a software product and ends when the software is **delivered**.

This cycle typically includes a requirements phase, design phase, implementation phase, test phase, and sometimes, installation and checkout phase.

Notes: (1) the phases listed above may overlap or be performed iteratively, depending upon the software development approach used. (2) This term is sometimes used to mean a longer period of time, either the period that ends when the software is no longer being enhanced by the developer, or the entire software life cycle. **IEEE 610.12** (1990)

system life cycle — The period of time that begins when a system is **conceived** and ends when it is **no longer available for use**. **IEEE 610.12** (1990)

Project Management

Project Management

Goals of Project Management

- **Main and general goal**: a **successful** project, i.e. project delivers
  - defined results
  - in demanded quality
  - within scheduled time
  - using the assigned resources.

software delivery

Secondary goals:

- build or strengthen good **reputation** on market,
- acquire **knowledge** which is useful for later projects,
- develop **re-usable components** (to save resources later),
- be attractive to **employees**.
- ...

Activities of Project Management

- **Planning** – without plans, a project cannot be managed. Mistakes in planning can be hard to resolve.
- **Assessment and Control** – work results and project progress have to be assessed and compared to the plans; it has to be observed whether participants stick to agreements.
- **Recognising and Fighting Difficulties as Early as Possible** – unforeseen difficulties and problems in projects are not exceptional but usual. Therefore, project management needs to constantly "screen the horizon for icebergs", and, when spotting one, react timely and effectively. In other words: systematic risk management.
- **Communication** – distribute information between project participants (project owner, customer, developers, administration).
- **Leading and Motivation of Employees** – leading means: going ahead, showing the way, "pulling" the group. Most developers want to achieve good results, yet need orientation and feedback.
- **Creation and Preservation of Beneficial Conditions** – provide necessary infrastructure and working conditions for developers (against: demanding customers, imprecisely stated goals, organisational restructuring, economy measures, tight office space, other projects, ...)

What to (Plan and) Manage?

Managing software projects involves

- tasks and **activities**,
- people and **roles**,
- **costs** and **deadlines**.

What to (Plan and) Manage (1/3)? Tasks and Activities

What to (Plan and) Manage (1/3)? Tasks and Activities

Work that commonly needs to be done in order to develop or adapt software:

- **Analysis** – Software is developed to solve a problem/satisfy a need. Goal of analysis: understand the problem, assess whether/in how far software can be used to solve it.
- **Specification of Requirements** – sort out, document, assess, extend, correct . . . results of analysis. Resulting documents are basis of most other activities!

  Formal methods: check consistency, realisability.
- **Design, Specification of Modules** – Most software systems are not monolithic but consist modules or components which interact to realise the overall functionality. Overall structure is called **software architecture** (→ later). Design architecture, specify component interfaces as precise as possible to enable concurrent development and seamless integration.

  Formal methods: verify that design meets requirements.
- **Coding and Module Test** – The needed modules are implemented using the chosen programming language(s). When ready, tested as needed, and ready for integration.

  Formal methods: verify that code implements design.
- **Integration, Test, Approval** – System is constructed from completed components, interplay is tested. Customer checks system and declares approval (or not).
- **Deployment, Operation, and Maintenance** – System is installed up to customer needs and becomes operational. Occurring errors are fixed. New requirements (changes, extensions): new project (so-called maintenance project).
- **Dismissing and Replacement** – Most software systems (sooner or later) become obsolete, and are often replaced by a successor system. Common reasons: existing system no longer maintainable, not adaptable to new or changed requirements.

What to (Plan and) Manage (2/3)? People and (other) Resources

(Plan and) Manage (2/3) — People and (other) Resources

Customer Developer

Clients Software people

Recall: roles "Customer" and "Developer" are assumed by **legal persons**, which often represent many people.

The same legal person may act as "Customer" and "Developer" in the same project.

Useful and common roles in software projects:

- **customer**, **user**
- **project manager**
- **(sytems) analyst**
- **software architect**, **designer**
- **(lead) developer**
  programmer, **tester**, . . .
- **maintenance engineer**
- **systems administrator**
- **invisible clients**: legislator, norm/standard supervisory committee

Excursion: The Concept of Roles

In a software project, at each point in time:

- there is a set P of people, e.g. $P = \{$ $\}$
- there is a set R of (active) roles, e.g. $R = \{$ mgr, prg, tst, ana $\}$
- there is a (many-to-many) relation between elements of P and R

$$assumes \subseteq P \times R$$

each person has a role ($\downarrow_1 assumes = P$), each role a person ($\downarrow_2 assumes = R$).

- **Example**:

$assumes = \{$ ( , mgr), ( , prg), (, prg), (, prg), (, tst), (, ana) $\}$

- Possible visualisation:

Excursion: The Concept of Roles Cont'd

Roles typically come with **responsibilities** and **rights**.

For example,

- tst: a test engineer
  - is responsible for quality control
  - has the right to raise issue reports
- mgr: a project manager
  - has the right to raise issue reports
  - is responsible for closing issue reports
- prg: a programmer
  - is responsible for reporting unforeseen problems to the project manager
  - is responsible for respecting coding conventions
  - is responsible for addressing issue reports

(Plan and) Manage (2/3) — People and (other) Resources

Some truisms and commonplaces to keep in mind:

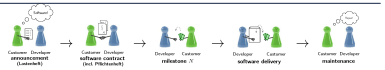
- "Software engineering [...] takes place in the heads of humans, who like to get software or develop it. **Humans are central** [in Software Engineering]; for us, that's not an empty phrase ('Floskel'), but a factual statement." (Ludewig and Lichter, 2013)
- **Being discontent** with the team situation, doesn't make people better developers. (Other way round, in most cases.)
- Recognising and resolving tensions in your team (or at least trying to) is an activity towards project success, thus a **responsibility** of each and every team member.

  "Everybody is responsible, the **project manager** is a little bit more responsible."
- "If somebody stronly insists on a claim which feels obviously wrong to you, he/she may be true given her/his respective (implicit) **terms** and **assumptions**." (source?)

  Try to understand and explicate these terms and assumptions.
- "Never attribute to malice that which can be adequately explained by stupidity." (Hanlon's Razor)

What to (Plan and) Manage (3/3)? Deadlines and Costs

What to (Plan and) Manage (3/3)? — Deadlines and Costs



A **phase** is a continuous, i.e. not interrupted range of time in which certain works are carried out and completed. At the end of the phase, there is a **milestone**.
A phase is **successfully completed** if the criteria defined by the milestone are satisfied.

Ludewig & Lichter (2013)

- Phases (in this sense) do not overlap! There may be different "threads of development" running in parallel, structured by different milestones.
- Splitting a project into phases makes controlling easier; milestones may involve the customer (accept intermediate results) and trigger payments.
- The **granularity** of the phase structuring is critical:
  - very short phases may not be tolerated by a customer,
  - very long phases may mask significant delays longer than necessary.

If necessary:
define **internal** (customer not involved) and **external** (customer involved) milestones.

Deadlines Cont'd

- Whether a milestone is **reached** must be **assessable** by
  - clear,
  - objective, and
  - unambiguous

  criteria.
- The **definition of a milestone** often comprises:
  - a definition of the **results** which need to be achieved,
  - the required **quality** properties of these results,
  - the desired **time** for reaching the milestone, and
  - the instance (person or committee) which **decide**s whether the milestone is reached.
- Milestones can be part of the **development contract**;
  not reaching a defined milestone as planned can lead to **legal claims**.

Costs

> 'Next to '**Software**', '**Costs**' is one of the terms occurring most often in this book."
>
> Ludewig and Lichter (2013)

A first approximation:

- **cost** ('Kosten') — all disadvantages of a solution, quantifiable in terms of money or not.
- **benefit** ('Nutzen') (or: negative costs) — all benefits of a solution.

Note: costs and benefits can be very subjective — and are not necessarily quantifiable...

Super-ordinate goal of many projects:

- **Minimize overall costs**, i.e. **maximise difference** between **benefits** and **costs**.
  (Equivalent: minimize sum of positive and negative costs.)

Costs vs. Benefits: A Closer Look

The benefit of a software is determined by the advantages achievable using the software; it is influenced by:

- the degree of coincidence between **product** and **requirements**,
- additional services, comfort, flexibility etc.

Some examples of cost/benefit pairs: Jones (1990)

Costs	Benefits
Labor during development	Use of existing labor
Labor during operation	Reduced operational labor
New equipment? (purchase, maintenance, depreciation)	Replacement of equipment maintenance? (sale, maintenance)
New software purchases	(Other) use of new software

Costs	Benefits
Conversion from old system to new	Improvement of system
Increased data gathering	Increased control
Employee discontent	Employee satisfaction
Training for employees	Increased productivity
Lost opportunities	Better market stance, basis for further growth

Costs: Economics in a Nutshell

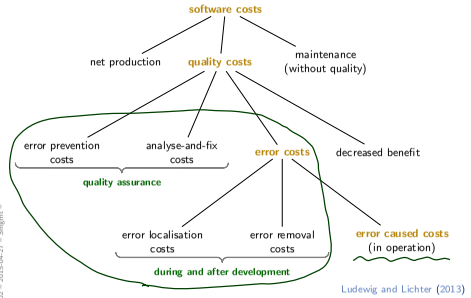
Distinguish **current cost** ('laufende Kosten'), e.g.

- wages
- management, marketing
- rooms
- computers, networks, software as part of infrastructure
- ...

and **project-related cost** ('projektbezogene Kosten'), e.g.

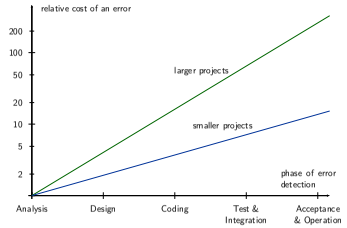
- additional temporary personnel
- contract costs
- expenses
- hardware and software as part of product or system
- ...

Software Costs in a Narrower Sense



Ludewig and Lichter (2013)

Discovering Errors Late Can Be Expensive



Relative error costs over latency according to investigations at IBM, etc. by (Boehm, 1979).

Visualisation: Ludewig and Lichter (2013)

Software Project Management Bottom-Line

"Management, management... Can't we just sit down and write some software?"

- **Quantity as Quality** (Ludewig and Lichter, 2013) — the large is in general not just a multiple of the small; solutions for small problems don't scale in general.

  Example: reliability. Consider a software system with N modules, each module being **correct** with probability p.

  N modules are correct with probability p^N. Example $N = 100$:

p	0.9	0.95	0.99	0.999
p^{100}	0.0000267	0.006	0.37	0.90

- Software Engineering as **defensive discipline**

  Analogy: hygiene in **hospital**.
  "Dear patient, we're working hard to protect you from an infection."
  — "Well, doctor, I thought you were working to **get me well** again."

  "Software Engineering is **boring** and **frustrating** for people who don't value the defense of failures as a positive achievement." (Ludewig and Lichter, 2013)

Project Planning and Cost Estimation

From 'Lastenheft' to 'Pflichtenheft'

software life cycle — The period of time that begins when a software product is conceived and ends when the software is no longer available for use.

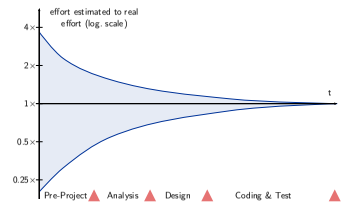
The software life cycle typically includes a concept phase, [...]. **IEEE 610.12 (1990)**

Lastenheft (Requirements Specification) Vom Auftraggeber festgelegte Gesamtheit der Forderungen an die Lieferungen und Leistungen eines Auftragnehmers innerhalb eines Auftrages.
(Entire demands on deliverables and services of a developer within a contracted development, created by the customer.) **DIN 69901-5 (2009)**

Pflichtenheft (Feature Specification) Vom Auftragnehmer erarbeiteten Realisierungsvorgaben aufgrund der Umsetzung des vom Auftraggeber vorgegebenen Lastenhefts.
(Specification of how to realise a given requirements specification, created by the developer.) **DIN 69901-5 (2009)**

- One way of getting there: a pre-project.

The "Estimation Funnel"



Uncertainty with estimations (following (Boehm et al., 2000), p. 10).

Visualisation: Ludewig and Lichter (2013)

Expert's Estimation

One approach: the **Delphi** method.

- Step 1:

- Step 2:

- Step 3:

- Then take the median, for example.

Algorithmic Estimation: COCOMO

- **Co**nstructive **Co**st **Mo**del:
  Formulae which fit a huge set of archived project data (from the late 70's).
- Flavours:
  - COCOMO 81 (Boehm, 1981): basic, intermediate, detailed
  - COCOMO II (Boehm et al., 2000)
- All based on estimated program size S measured in DSI or kDSI (thousands of Delivered Source Instructions).
- Factors like security requirements or experience of the project team are mapped to values for parameters of the formulae.

- COCOMO examples:
  - textbooks like Ludewig and Lichter (2013) (most probably made up)
  - an exceptionally large example:
    COCOMO 81 for the Linux kernel (Wheeler, 2006) (and follow-ups)

COCOMO 81

Software Project Type	Characteristics of the Type				a	b	c	d
	Size	Innovation	Deadlines/ Constraints	Dev. Environment				
Organic	Small (<50 KLOC)	Little	Not tight	Stable	2.4	1.05	2.5	0.38
Semi-detached	Medium (<300 KLOC)	Medium	Medium	Medium	3.0	1.12	2.5	0.35
Embedded	Large	Greater	Tight	Complex HW/ Interfaces	3.6	1.20	2.5	0.32

Basic COCOMO:

$$E \text{ (effort required)} = a(S/kDSI)^b \quad \text{[person-months]}$$
$$TDEV \text{ (time to develop)} = cE^d \quad \text{[months]}$$

Intermediate COCOMO:

$$E \text{ (effort required)} = \underbrace{M}a(S/kDSI)^b \quad \text{[person-months]}$$

. . . where

$$M = RELY \cdot CPLX \cdot TIME \cdot ACAP \cdot PCAP \cdot LEXP \cdot TOOL \cdot SCED$$

factor	very low	low	normal	high	very high	extra high
RELY required software reliability	0.75	0.88	1	1.15	1.40	
CPLX product complexity	0.70	0.85	1	1.15	1.30	1.65
TIME execution time constraint			1	1.11	1.30	1.66
ACAP analyst capability	1.46	1.19	1	0.86	0.71	
PCAP programmer capability	1.42	1.17	1	0.86	0.7	
LEXP programming language experience	1.14	1.07	1	0.95		
TOOL use of software tools	1.24	1.10	1	0.91	0.83	
SCED required development schedule	1.23	1.08	1	1.04	1.10	

References

References

Boehm, B. W. (1979). Guidelines for verifying and validating software requirements and design specifications. In *EURO IFIP 79*, pages 711–719. Elsevier North-Holland.

Boehm, B. W. (1981). *Software Engineering Economics*. Prentice-Hall.

Boehm, B. W., Horowitz, E., Madachy, R., Reifer, D., Clark, B. K., Steece, B., Brown, A. W., Chulani, S., and Abts, C. (2000). *Software Cost Estimation with COCOMO II*. Prentice-Hall.

Buschermöhle, R., Eekhoff, H., and Josko, B. (2006). success – Erfolgs- und Misserfolgsfaktoren bei der Durchführung von Hard- und Softwareentwicklungsprojekten in Deutschland. Technical Report VSEK/55/D.

DIN (2009). *Projektmanagement; Projektmanagementsysteme*. DIN 69901-5.

IEEE (1990). *IEEE Standard Glossary of Software Engineering Terminology*. Std 610.12-1990.

Jones, G. W. (1990). *Software Engineering*. John Wiley & Sons.

Knöll, H.-D. and Busse, J. (1991). *Aufwandsschätzung von Software-Projekten in der Praxis: Methoden, Werkzeugeinsatz, Fallbeispiele*. Number 8 in Reihe Angewandte Informatik. BI Wissenschaftsverlag.

Ludewig, J. and Lichter, H. (2013). *Software Engineering*. dpunkt.verlag, 3. edition.

Metzger, P. W. (1981). *Managing a Programming Project*. Prentice-Hall, 2 edition.

Noth, T. and Kretzschmar, M. (1984). *Aufwandsschätzung von DV-Projekten, Darstellung und Praxisvergleich der wichtigsten Verfahren*. Springer-Verlag.

Thayer, R. H. (1997). *Tutorial – Software Engineering Project Management*. IEEE Society Press, revised edition.