SOFTWARE ENGINEERING BASICS

Why design is important

Topics for this course

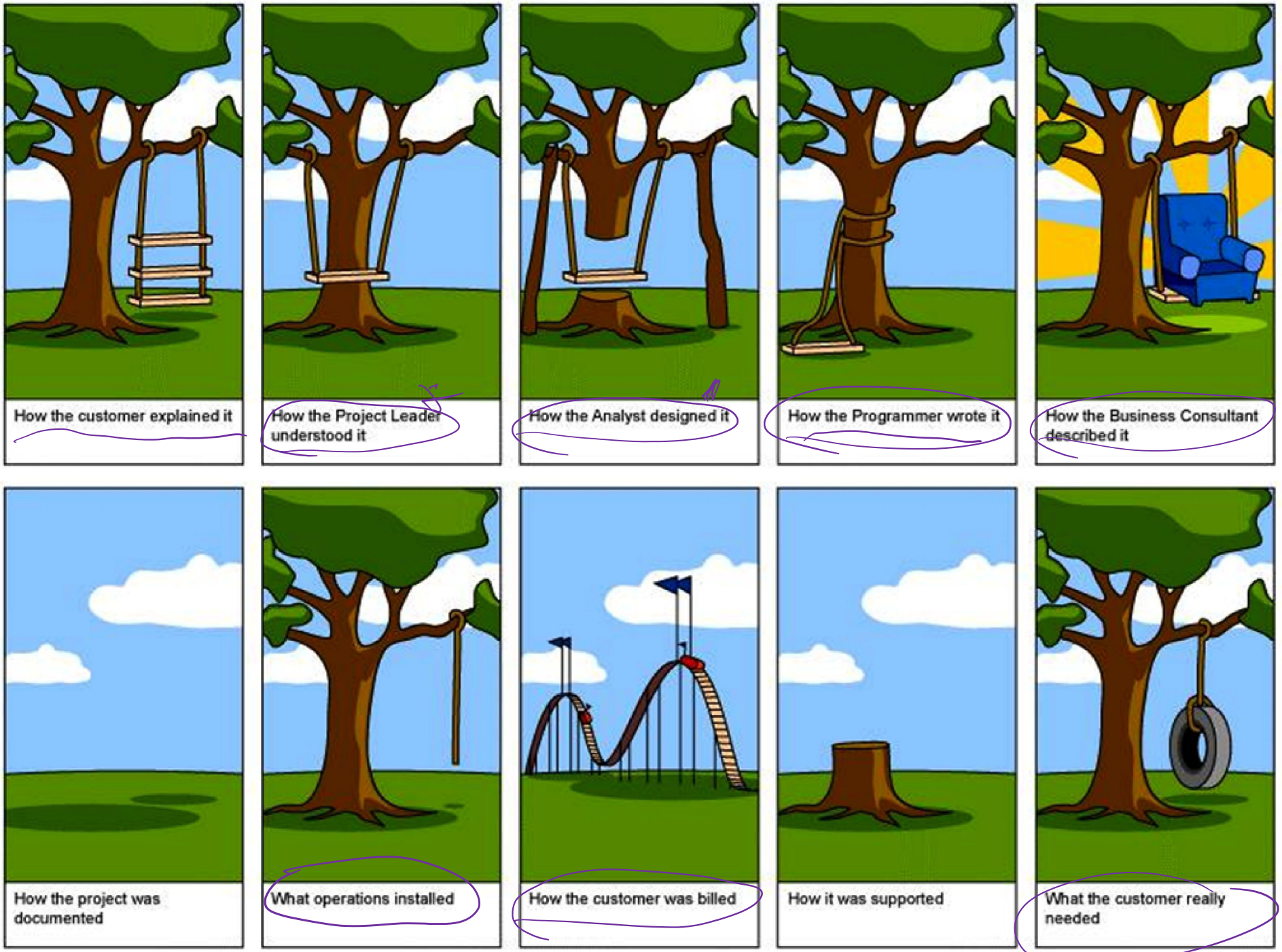
Important in this course

WHY DESIGN IS IMPORTANT

Motivation

Numbers

Examples for success and failure


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

MAKING GOOD SOFTWARE IS DIFFICULT

Software projects have properties that make them very different to other engineering projects:

- the product is intangible
- the product is uniquely flexible
  - easy to change, easy to break...
- almost all costs are development costs
  - many software projects are one-off projects
- the technology changes very quickly

NUMBERS

WHAT IS SOFTWARE ENGINEERING?

IEEE definition:

Software Engineering is the application of a **systematic**, **disciplined**, **quantifiable** approach to the **development**, **operation**, and **maintenance** of software; that is, the ***application of engineering to software***

MAKING GOOD SOFTWARE IS DIFFICULT

Source: CHAOS Report, Standish Group

MODERN RESOLUTION FOR ALL PROJECTS

	2011	2012	2013	2014	2015
SUCCESSFUL	29%	27%	31%	28%	29%
CHALLENGED	49%	56%	50%	55%	52%
FAILED	22%	17%	19%	17%	19%

The Modern Resolution (OnTime, OnBudget, with a satisfactory result) of all software projects from FY2011–2015 within the new CHAOS database. Please note that for the rest of this report CHAOS Resolution will refer to the Modern Resolution definition not the Traditional Resolution definition.

More information:

- `http://www.versionone.com/assets/img/files/CHAOSManifesto2013.pdf`
- RL Glass, The Standish Report: Does It Really Describe a Software Crisis? CACM, 49(8):15-16, 2006.
- J Laurenz Eveleens, C Verhoef: The Rise and Fall of the Chaos Report Figures. IEEE Software 27(1):30-36, 2010

MAKING GOOD SOFTWARE IS DIFFICULT

CHAOS RESOLUTION BY PROJECT SIZE

	SUCCESSFUL	CHALLENGED	FAILED
Grand	2%	7%	17%
Large	6%	17%	24%
Medium	9%	26%	31%
Moderate	21%	32%	17%
Small	62%	16%	11%
TOTAL	**100%**	**100%**	**100%**

The resolution of all software projects by size from FY2011–2015 within the new CHAOS database.

Source: CHAOS Report, Standish Group

EXAMPLES

SOFTWARE FAILURES

- Therac-25 radio therapy (1985, 3 deaths)
  - machine delivered radiation over-dose
  - software interlocks replaced earlier hardware to prevent delivery
  - … but failed due to race conditions
  - widely studied failure of safety-critical software

More information:

- NG Leveson, CS Turner: Investigation of the Therac-25 Accidents. IEEE Computer 26(7): 18-41, 1993.
- `http://sunnyday.mit.edu/papers/therac.pdf`

SOFTWARE FAILURES

- Pentium FDIV Bug (1994, ~$500 mil.)
  - error in floating-point division algorithm
    - look-up tables incomplete
    - small inaccuracies
  - chips recalled
  - sparked huge investment in formal verification
    - from model-checking to higher-order logic

ARIANE 501 - A SOFTWARE DISASTER...

ANALYSIS OF ARIANE 501

Possible reasons:

- wrong design philosophy
  - "if something breaks down, it is caused by a random hardware failure"
  - Action: shut down that part
  - no provision for design errors!

Software failures (almost) **never have a single cause or a single solution!**

More information:

- http://www.esa.int/For_Media/Press_Releases/Ariane_501_-_Presentation_of_Inquiry_Board_report
- http://en.wikipedia.org/wiki/Cluster_(spacecraft)

SOFTWARE SUCCESSES

Some software systems are extremely successful:

- OS/360
  - morphed into z/OS (current IBM mainframe OS)
- SABRE (American Airlines reservation system)
  - evolving since 1960
  - multiple variants for other airlines
- NASA's Apollo guidance system
  - get to the Moon and back, with 75k memory
- Excel
  - world's most widely used programming environment
- Morris' internet worm

SUMMARY

- Design is important
- Helps understand what system does and how it works
- Bad Design/Implementation might be
  - Expensive
  - Cost lives
  - Lead to failure

WHAT YOU NEED TO DO

(SOME) BRANCHES OF SOFTWARE ENGINEERING

- Core processes:
  - Planning
  - Analysis
  - **Design**
  - Programming
  - Validation and verification

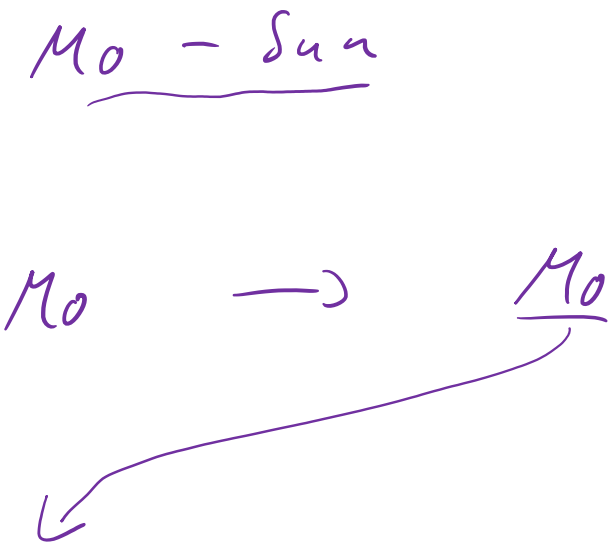
- Support processes:
  - Requirements engineering (SER415)
  - Project management
  - Quality assurance (SER316)
  - Configuration management (SER316)
  - Documentation

GENERAL DESIGN

- What is design in general, what is Software design?
- Software Development LifeCycle
- Reverse Engineering
- **Software Design**
  - **UML Design**
  - **User modeling and User Interface Design**
  - **Design Patterns**
  - **Formal specification**

ENTERPRISE MODEL

- Lecture/Video/Reading
- Followed by assignment
- Followed by apply in context (project)



TODO

- 7 Lab assignments – 30% of final grade (one per week)
- 1 semester long Project in a team of 5-6 students – 40% of final grade
  - approx. Kickoff October 16th (Monday) – more information will follow
  - Google project site should always be up to date on Sundays
  - 4 deliverables for project (2 for peer review, 2 for grading)
- Exam at the end of semester – 30% of final grade

➔ **Detailed information in Kickoff document/video**

IMPORTANT

- You will create different UML diagrams and a formalization
  - They all belong together and form one Design model
  - They all need to be consistent
  - Do not do a divide and conquer approach
- We will talk about different diagrams
  - Most diagrams can be used in different levels of abstraction, we will discuss some of them
  - One main goal is for you to understand diagram syntax, what they are used for and what level of abstraction can be used
  - We will ask a specific process of you (similar to RUP). This process uses specific diagrams and levels of abstraction. Make sure you use them correctly.
  - Different processes, companies, projects might use different diagrams, levels of abstraction etc.