Topics
- types of software
- software quality
- software failures
- software engineering definition

What is software engineering?

What is software?
A software product can comprise:
- programs: executables, binary libraries, source code, generated code, ...
- documentation: requirements definition, technical manuals (design and implementation), user manuals, licenses, source code, ...
- associated data: test cases, configuration files, database framework, ...
- associated services: training, technical support, maintenance, ...

Types of software – by customer
- Generic software is developed to be sold to a general market with a range of different customers
  - PC software (Excel or Word)
  - Apps
- Bespoke (custom) software is developed for a particular customer according to their specification
- Mixture of the above: software product lines
  - Linux
  - SAP
  - automotive control software
Types of software – by domain
- system (e.g., Windows 7, delphi)
- large variety of different application domains
- with different
- characteristics
- constraints
- goals
- business (e.g., SAP, Oracle)
- PC / office software (word processor, spreadsheet)
- web applications (HTML, CGI, PHP)
- big data (data mining)

Making good software is difficult
Making good software is difficult

The “-ilities”: what is good software?
Software systems have different quality attributes:
- correctness
- efficiency
- usability
- reliability
- robustness
- dependability
- maintainability
- portability
- scalability
- reusability

Software engineering involves (implicit or explicit) trade-offs of the different quality attributes!

The “-ilities”: internal operation
- correctness: satisfies the functional requirements specifications
  – can be proven formally if specifications are formal
  – ... but typically shown by testing
- efficiency: does not waste system resources
  – profiling

The “-ilities”: user-oriented
- usability: ease of system use by expected users
- reliability: ability to deliver services as specified
  – probability of absence of failures for a certain time period (cf. also availability)
- robustness: behaves reasonably in unforeseen circumstances (incorrect input, hardware failure,...)
  – “correctness outside specification”
- dependability: extent to which software can be trusted by users
  – aggregate property:
    correctness + reliability + robustness + ...

The “-ilities”: developer-oriented
- maintainability: easy to evolve to meet changing needs, easy to repair defects
- portability: can run on different hardware platforms or software environments
- scalability: ability to handle growing workload
  – key problem for start-ups
- reusability: can be used (with minor modifications) to build another product

Making good software is difficult
Most projects fail (to some extent):

- Software
  - 25% completed on time, within budget
  - 25% completed on time, excessive feature creep
  - 25% completed on time, excessive feature creep
  - 25% completed on time, excessive feature creep

- Hardware
  - 50% completed on time, within budget
  - 50% completed on time, excessive feature creep

- Cost
  - 50% completed on time, within budget
  - 50% completed on time, excessive feature creep

- Delivered
  - 40% completed on time, within budget
  - 40% completed on time, excessive feature creep

- Additional work
  - 30% completed on time, within budget
  - 30% completed on time, excessive feature creep

- Quality
  - 25% completed on time, within budget
  - 25% completed on time, excessive feature creep

- Customer satisfaction
  - 10% completed on time, within budget
  - 10% completed on time, excessive feature creep

Source: CHAOS Report, Standish Group

But things are getting better...

More information:
Making good software is difficult

The major cause of failure is complexity.

Source: CHAOS Report, Standish Group

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

Software failures

• OS/360 (1965, $$ $$)
  – IBM next-generation operating system
  – too complex (second-system effect)
  – “adding manpower to a late software project makes it later...”
  – progenitor of software crisis

More information:


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:


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

Software failures

• US Patriot missile failure (1991, 28 deaths)
• American Airlines Cali crash (1995, 159 deaths)
• Mars Rover (1997, 2004)
• Mars Climate Orbiter & Polar Lander (1999)
• Marine Corps Osprey tilt-rotor (2000, 4 deaths)
• Panama Cobalt-60 radio-therapy (2001, 5 deaths)
• Northeast power blackout (2003, 3 deaths)
• Knight Capital “flash crash” (2012, $440 mil.)
• ...
• Risks Digest:
  http://catless.ncl.ac.uk/Risks/
Analysis of Ariane 501

Details:
- cause: overflow in conversion of variable E_BH from 64-bit floating point to 16-bit signed integer
- 7 risky conversions, 4 protected; E_BH was not
  - reasoning: physically limited or large margin of safety
  - exception handling:
    - report failure on databus and
    - shut down

Possible reasons:
- 
  - Testing!

Software failures (almost) never have a single cause or a single solution!
Software has many stakeholders.

- **user**: uses the software
- **customer**: pays for the software
- **developer**: produces the software = designers + programmers + managers
- **contractor**: markets the software

Communication patterns:

```
user → agrees ← customer
```

```
plan ↓
```

```
developer → orders ← contractor
```

Software has many stakeholders with different goals.

- **user**: “It’s easy to learn, efficient to use, helps me get my work done.”
- **customer**: “It solves problems at an acceptable cost in terms of money paid and resources used.”
- **developer**: “It is easy to design, easy to maintain, and easy to reuse.”
- **contractor**: “Sells many copies while costing less to develop and maintain.”

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**

What is software engineering?

- **SE vs CS and other fields**

  “Software engineering is the part of computer science which is too difficult for the computer scientist.”

  F. L. Bauer

  - **Computer science** deals with theory & foundations
  - **Software engineering** deals with the practicalities of developing and delivering useful software.
  - Computer science theories are still insufficient as a complete underpinning for software engineering – unlike (say) physics and electrical engineering

  - Computer science theories are still insufficient as a complete underpinning for software engineering – unlike (say) physics and electrical engineering

  - Computer science theories are still insufficient as a complete underpinning for software engineering – unlike (say) physics and electrical engineering

More information:

- http://www.computer.org/portal/web/swebk

What is software engineering?

- **SWEBOK** is a guide to various knowledge areas within the Software Engineering field – 335 pages
- covers knowledge after four years of practice
- not everybody agrees on it...
Summary

- software engineering is the application of engineering methods to software
- software engineering is concerned with big projects
- software projects have a huge variability
- software projects have many stakeholders with different interests
- many software projects fail