Introduction
Topics

- types of software
- software quality
- software failures
- software engineering definition
What is software engineering?
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 software** (Windows 7, drivers)
- **real-time software** (ticket online, control radar)
- **embedded software** (microwave control)
- **ubiquitous computing** (apps, wireless networks)
- **engineering / scientific software** (Mathlab)
- **high-performance computing** (weather forecast)
- **business software** (SAP, Oracle)
- **PC / office software** (word processor, spreadsheet)
- **web applications** (HTML, CGI, PHP)
- **big data** (data mining)

Large variety of **different application domains** with **different**
- characteristics
- constraints
- goals
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!

More information:

- ISO/IEC 9126 Software engineering — Product quality
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):

<table>
<thead>
<tr>
<th>Year</th>
<th>Completed on time, on budget</th>
<th>Late, over budget, or missing features</th>
<th>Cancelled before completion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1994</td>
<td>16%</td>
<td>53%</td>
<td>31%</td>
</tr>
<tr>
<td>1996</td>
<td>27%</td>
<td>33%</td>
<td>40%</td>
</tr>
<tr>
<td>1998</td>
<td>26%</td>
<td>46%</td>
<td>28%</td>
</tr>
<tr>
<td>2000</td>
<td>28%</td>
<td>49%</td>
<td>23%</td>
</tr>
<tr>
<td>2002</td>
<td>34%</td>
<td>51%</td>
<td>15%</td>
</tr>
<tr>
<td>2004</td>
<td>29%</td>
<td>53%</td>
<td>18%</td>
</tr>
</tbody>
</table>

Source: CHAOS Report, Standish Group

But things are getting better...

More information:
Making good software is difficult

The major cause of failure is complexity:

CHAOS RESOLUTION BY LARGE AND SMALL PROJECTS

- Small Projects:
  - 76% Successful
  - 20% Failed
  - 4% Challenged

- Large Projects:
  - 38% Successful
  - 10% Failed
  - 52% Challenged

Source: CHAOS Report, Standish Group

Project resolution for the calendar year 2012 in the new CHAOS database. Small projects are defined as projects with less than $1 million in labor content and large projects are considered projects with more than $10 million in labor content.
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:
• 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
Software failures

- US Patriot missile failure (1991, 28 deaths)
- American Airlines Cali crash (1995, 159 deaths)
- 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/
Ariane 501 – A software disaster...
Analysis of Ariane 501

Disintegration after 39 sec:
• caused by large correction for attitude deviation
• caused by wrong data sent to On Board Computer
• caused by software exception in Inertial Reference System after 36 sec
Analysis of Ariane 501

Details:

• cause: overflow in conversion of variable E_BH from 64-bit floating point to 16-bit signed integer

L_M_BV_32 := TBD.T_ENTIER_32S((1.0/C_M_LSB_BV)*G_M_INFO_DERIVE(T_ALG.E_BV));
if L_M_BV_32 > 32767 then P_M_DERIVE(T_ALG.E_BV) := 16#7FFF#;
elsif L_M_BV_32 < -32768 then P_M_DERIVE(T_ALG.E_BV) := 16#8000#;
else P_M_DERIVE(T_ALG.E_BV) := UC_16S_EN_16NS(TDB.T_ENTIER_16S(L_M_BV_32));
end if;
P_M_DERIVE(T_ALG.E_BH) := UC_16S_EN_16NS(TDB.T_ENTIER_16S((1.0/C_M_LSB_BH)*G_M_INFO_DERIVE(T_ALG.E_BH)));

• 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
Analysis of Ariane 501

Possible reasons:

• inadequate testing
  – specification does not contain Ariane 5 trajectory data as a functional requirement
  – in testing the IRS’s were simulated in software

• wrong type of reuse
  – component reused from Ariane 4…
  – … but Ariane 5 is much faster than Ariane 4, and horizontal velocity builds up more rapidly
  – values exceed margin of safety after ~36 secs
  – component has no purpose for the Ariane 5…
  – … but was still kept to minimize code changes
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
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
• Google, Amazon, Facebook, ...
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:
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?

• concerned with **BIG** systems
  – complexity  – heterogeneity
  – scalability  – correctness
• software **evolves**
  – plan for *change*, plan for *uncertainty*
• software must **effectively support users**
• development is a **group effort**
  – involves **different disciplines**
  – must still be **efficient**
• **no single best solution**
  – **compromise** is inevitable
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...

More information:
- http://www.computer.org/portal/web/swebok
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
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