Software Construction and User Interfaces (SE/ComS 319)

Ali Jannesari

Department of Computer Science

Iowa State University, Spring 2019

INTRODUCTION

Outline

- Introduction
  - Software construction
  - Goals, outcomes and challenges

What is Software Construction (Software Engineering)?

- Software engineering is the technological and organizational discipline for the systematic development and maintenance of software systems that fulfill specified functional and non-functional attributes.

Software Construction (Engineering)

- Set of concepts & practices
- An engineering discipline about all aspects of software production
  - Gathering requirements
  - Design
  - Development
  - Testing & debugging
  - Maintenance

What is the goal of software engineering?

- Produce good and functional software, within the given time schedule and resource budget
- Good programmers ➔ good software
- Good Programmers?
  - Write good programs on-time that work correctly (functional, time-to-market, budget and resources)

Criteria for good software?

- Reliable/correct (few bugs)
- Efficient (run fast)
- Easy for maintenance
- Good usability
- Good security
- ➔ **Software quality**

What are the challenges?

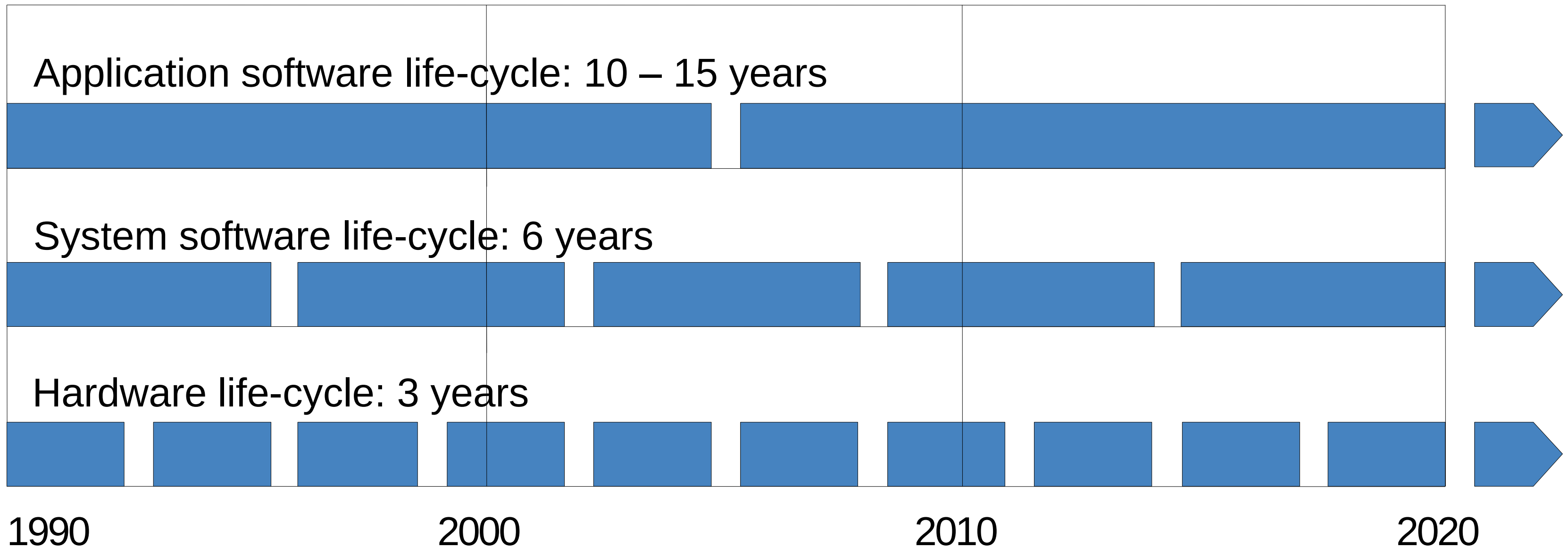
- Large code sizes (not easy to debug)
  - Linux kernel 1.0.0 (1994): 100K+ ➔ Linux kernel 2.6.0 (2003): 5M+
  - Boeing 787: 5M+
  - Chrome? Firefox? Mac OS X?
    - See: https://informationisbeautiful.net/visualizations/million-lines-of-code
- Changing requirements
  - User, hardware, platform (portability), new GUI, new Interfaces (web interface, speech control, app), …
- Large development teams distributed in different offices around the globe
- Time-to-market

What are the challenges?

Example @ Google:

- 15K+ developers in 40+ offices
- 4K+ projects under active development
- 5K+ submissions per day on average
- Single monolithic code tree with mixed language code
- Development on one branch - submissions at head
- All builds from source
- 20+ sustained code changes per minute with 60+ peaks
- 50% of code changes monthly
- 75+ million test cases run per day

Problems in creating time-to-market software



Problems in creating time-to-market software

If you develop application software, then it means that ...

- During the lifetime of the application software at least once the underlying system software and at least twice the hardware is replaced
- The target systems for which software is developed may not yet be available at development time
- Software products distributed throughout the world may need country-specific variants (e.g. native-language user interfaces, etc.)

How to overcome and handle the challenges?

- Practices/disciplines
- Tools
- Necessary to have **Software Engineering:**
    - Practices and tools about design, development, and maintenance of software

Changes in the software in the last decades

- Increasing Importance – software is becoming more and more critical and important in many in-demand areas
- Growing complexity
- Growing software in mobile devices (everywhere)
- Networking and geographical distribution
- Increasing quality requirements
- Increasing standard software
- Increasing ”legacy code“
- Increasing "off-the-shelf" development

Importance of software

- Give examples of where you depend on software in your daily life!

Software quality

- Increasing quality requirements
- Software failures are the reason for 50% of industrial sector failures
- According to Cusumano, the defects found in every 1000 lines of source code developed as follows:
  - 1977: an average of 7 - 20 defects
  - 1994: an average of 0.2 - 0.05 defects
  - In 17 years, the defect density could be reduced by about 100 times

Software quality

- 0.1% defect level means:
  - per year:
    - 20,000 defective drugs and medications
    - 300 failing cardiac pacemakers
  - per week:
    - 500 errors in medical operations
  - per day:
    - 16,000 lost letters in the mail
    - 18 plane crashes
  - per hour:
    - 22,000 checks were not booked correctly

In the future: still massive quality assurance efforts needed!

Software quality problems

Example 1:

- Luggage transport system at Denver airport
  - Gigantic airport: 54 mi^2 , double the area of Manhattan, 10 times the width of Heathrow
  - 3 aircrafts can land at the same time (in bad weather)
  - Luggage transport system:

22000	mi long
4000	automatic vehicles
100	control centers by computers
5000	optical sensor
400	radio equipments
56	barcode readers

Luggage transport system of Denver airport

Luggage transport system of Denver airport

Software quality problems

- Denver Airport planned opening: 31 Oct. 1993
- Automatic luggage transport system delayed due to software problems (suitcase crushed and shredded)
- First delay until December 1993, then March 1994.
- June 1994: System still not operational, no estimate for startup
- Commissioning: Feb. 1995 (16 months delay)
  - Cost: $1.1 mil. / day (interest + empty airport operation)
  - Total: $560 million loss (see article in Scientific American, Sep. 1994)
- The luggage system was scrapped in 2005 and replaced with manual tractors and trailers. It never worked properly!

Software quality problems

Example 2:
The Ariane 5 rocket exploded 40s after the start. (1996, $500 million damage).

Details: http://www.around.com/ariane.html

Video: http://www.youtube.com/watch?v=kYUrqdUyEpI

Case study of the crash

- An overflow occurred when converting a 64-bit floating point number into a 16-bit integer in a program called the "Inertial Reference System".
- This overflow was not caught, so the whole attitude control stopped, causing the rocket to tilt in an improper direction, and then causing it to explode.
- The software had been developed for the Ariane 4 rocket, where this overflow could not occur.
- So the defect was a **code reusability error** caused by missing specification of the conditions under which the software works properly.

Ariane 5 – Navigation system (Ada code)

```
...
declare
 vertical_veloc_sensor: float;
 horizontal_veloc_sensor: float;
 vertical_veloc_bias: integer;
 horizontal_veloc_bias: integer;
 ...
begin
 declare
  pragma suppress(numeric_error, horizontal_veloc_bias);
 begin
  sensor_get(vertical_veloc_sensor);
  sensor_get(horizontal_veloc_sensor);
  vertical_veloc_bias := integer(vertical_veloc_sensor);
  horizontal_veloc_bias := integer(horizontal_veloc_sensor);
  ...
 exception
  when numeric_error => calculate_vertical_veloc();
  when others => use_irs1();
 end;
end irs2;
```

16-bit signed integer

Floating point value: 32768.0

Overflow on conversion, which was not caught.
$2^{15} - 1 =$ 32.767 (signed integer)

Switch to replacement computer.

Exact cause for the crash

- 37 seconds after the Ariane 5 was ignited, the horizontal speed sensor returned 32,768.0 at an altitude of 3,700 meters.
- An overflow occurred when converting the value 32,768,0 to a 16-bit integer.
  - The maximum number for 16 bits signed integer is: $2^{15} - 1 =$ 32.767
- However, this error was not caught.
  - The software therefore switched over to the replacement computer, which had previously shut down 72ms due to the same problem.
- The diagnostic data sent to the main computer was interpreted as trajectory data.
- This resulted in non-sense control commands for the engines, which led to an extreme position of the rocket.
- As the Ariane 5 threatened to break apart, it blew itself apart (39 seconds after ignition).

Need of better software engineering!

- Advanced tools and methods for software engineering
- Software process
- All aspects of software production, etc.

Summary

- Software construction/engineering
- Criteria for good software, challenges, etc.
- Importance of software and software quality, etc.

Literature

- https://www.tutorialspoint.com/extreme_programming/
- http://www.extremeprogramming.org/
- Beck, K.: Extreme Programming explained, Addison-Wesley.
- Fowler, M.: Refactoring: Improving the Design of Existing Code, Addison-Wesley.

WIKIPEDIA
The Free Encyclopedia