Lecture 25: Lessons from the History of Software Development (Part III)

Outline

- Is Software Different?
- Trends from the History of Software Development
  - Sophistication of skills (of developers and users)
  - Propagation of good development practices
  - Size of projects and products
  - Criticality of getting it right

References

- "Professional Software Development", by Steve McConnell
- The "Ariane 5" disaster
  - http://www2.vuw.ac.nz/staff/stephen_marshall/SE/Failures/SE_Ariane.html

"As The Size of the Software System Grows, The Key Discipline Changes"

10^3 LOC	→	Mathematics
10^4 LOC	→	Science
10^5 LOC	→	Engineering
10^6 LOC	→	Social Science
10^7 LOC	→	Politics
10^8 LOC	→	?

Source: David Notkin's lecture notes at http://www.cs.washington.edu/education/courses/503/00sp/lecture%202%20(overview).htm

Student Submission

Putting the Numbers in Perspective

How large is the largest software product you have ever worked on?
a) 10^3 LOC
b) 10^4 LOC
c) 10^5 LOC
d) More

What is the software industry's average productivity (in LOC) per person per year?
a) 10^3 LOC
b) 10^4 LOC
c) 10^5 LOC
d) More

Student Submission

Getting It Right: Is Software Born Correct?

- Should software be assumed to be faulty until the use of the best known methods can demonstrate that it is correct? Or should it be vice versa?
- **Which way do you lean?**
  - a) Faulty until proven correct
  - b) Correct until proven faulty
- **Why?**

Lessons from "Ariane 5"

(http://www2.vuw.ac.nz/staff/stephen_marshall/SE/Failures/SE_Ariane.html)

(A) **Test!**

(B) **Try to write code that cannot fail.**
- The SRI should never have sent data to the main computer that was not valid and open to be misinterpreted as flight instructions.

(C) **Don't allow errors or exceptions to propagate in an uncontrolled manner.**
- The possibility of an exception had been allowed for in the design, just not for this particular calculation.

(D) **When reusing code from another system, make sure any existing requirements / assumptions remain valid.**
- You should be able to track from requirements to code and vice-versa.

(E) **Reused code still needs to be tested.**
- Yes, its likely to have fewer bugs if it has worked before, but the reuse has changed the context, new bugs may appear or old ones be exposed.

(F) **Make sure that the documentation links aspects of the design with specific requirements and vice versa.**
- … so that when requirements are changed, affected design and coding decisions can be easily identified.

(G) **Test!**

Driving Forces behind the Evolution of Software Dev't

- Software becomes a business and a profession
  - No longer just a hobby
  - Building more and more complex projects
  - Standards emerge
- Best practices get distilled over time
  - Lifecycle processes
  - Designing for change, for test, for leanness, etc.
- Productivity tools appear that aid developers
- Economic and societal trends play an increasingly important role
  - Society increasingly depends on the technology and its robustness and safety

Lecture 26: Course Retrospective

Outline

- How this course differs from others
- What this course did not offer
- My version of the main take-away points

How Software Engineering Differs from Other Courses

Student Submission

- How is Software Engineering different from other courses and disciplines you have been exposed to?
  - (List the *most important* difference in your mind.)

How Software Engineers Differs … – My View

- Holistic nature of the discipline
- Making high-level decisions – beyond the technical ones – but at least as important
- Few clear-cut answers; mostly good practices
- Larger teams
- Opportunity to propose and work on your own ideas
- Instructors in the coaching role
- Mistakes along the way are encouraged, not penalized
- Plans (always) change
- Content topics: software design, testing, project management, etc.

What Software Engineering Encompasses (revisited)

- In contrast to many CS disciplines you have been exposed to, software engineering includes elements of:
  - **Computer science** (incl. algorithms, data structures, programming languages, tools)
  - **Business and management** (incl. project management, scheduling, prioritization)
  - **Economics/marketing** (incl. what makes a product sell, niche markets, monopolies)
  - **Communication** (incl. managing relations with stakeholders – customers, management, developers, testers, sales)
  - **Law** (incl. patents, licenses, copyrights, reverse engineering)
  - **Sociology** (incl. modern trends in societies, localization, ethics)
  - **Political science** (incl. topics at the intersection of law, economics, and global societal trends; (public) safety)
  - **Psychology** (incl. personalities, styles, usability, what makes things fun)
  - **Art** (incl. GUI design, what makes things appealing)
  - … more
- Hence, the flavor of the discipline is necessarily "softer" and there are fewer clearly right/wrong answers.

What I Think You Have Gotten out of This Course

- Get exposure to some of the best software development practices in use today
- Learn how to more effectively collaborate with others toward a common goal
- Understand how software is produced – from conception to shipping and subsequent maintenance
- Have experience working in a larger team toward a common goal
- Be able to lead an intelligent conversation with expert practitioners in the field of software engineering
- Understand the issues and tradeoffs involved in making decisions as software engineers and project managers

How This Course Differs from the "Real World" of Software

- Aspects of "real world" software development that this course has *not* exposed you to
  - Changing requirements
  - Real outside customers
  - Working full-time on a major software project
  - Working on an existing project with its constraints (i.e., not starting from scratch)
  - People leaving and joining the team partway into a project
  - Monetary compensation

The Main Take-Away Points in a Nutshell – My View

- It is all about *risk management*.
  - Requirements gathering, designing, prototyping, incremental releases, testing, code reviewing, refactoring, scheduling, prioritizing, etc.
- The notion that you provide *value* to your customers
- In all interactions, ask yourself who your audience is and what their expectations are.
- Good management and communication are crucial to success.
  - It sounds easy until one experiences it first-hand.
  - Software development is an inherently human activity.
- Frequent incremental releases are invaluable.

Peer Appreciation Activity

- **When:** After a big milestone/product delivery
- **Why:** To clear out any residual stress and (maybe) hard feelings, and to (re)unite the team
- **How:**
  - All teammates sit in a circle. One person speaks at a time.
  - Someone starts, turning to the person to their right, and addressing them with *exactly* the phrase below.
    - Name and blank must be substituted appropriately. J
  - The recipient repeats this with the person to their right, while everyone listens.
  - Turn around the circle several times for best results.
  - After each turn, switch places or direction of movement.
- **Example:** "*Joe*, I appreciate you for _____."