Introduction

Software Testing and Maintenance
SWE 437

Brittany Johnson

Adapted from slides by Jeff Offutt

“Traditional” Quality Attributes (1980s)

1. Efficiency of process (time-to-market)
2. Efficiency of execution (performance)

We often teach these as priorities in undergrad computer science classes…

This was true…in 1985

Modern Quality Attributes

1. Reliability
2. Usability
3. Security
4. Availability
5. Scalability
6. Maintainability
7. Performance & time to market

All of these factors (sometimes called "-ilities") are important in the 2000s

Based on an informal survey of around a dozen software development managers, 2000

Software Projects in the 1960s

In the **1960s** we built **log cabins**...

Single programmer

Not much **complexity**

No process needed

Design could be kept in **short term memory**

Software Projects in the 1970s

In the **1970s** we built **bigger houses**…

Still **single** programmer

- focus on algorithms & programming

A little **more complex**

Lack of process = **disasters**

Quality didn't affect **bottom line**

But **costs** were starting to **increase…**

Software Projects in the 1980s

In the **1980s** we built **office buildings**...

We needed **teamwork** and **communication**

A lot more **complex** + **data abstraction**

Needed written **requirements** and **design**

Poor process → spectacular **failures**

Missing skills and knowledge for **successful engineering**

Software Projects in the 1990s

In the **1990s** we built **skyscrapers**…

Teamwork + communication **not enough**

Needed **new technologies** – languages, modeling techniques, and processes

Software development **changed** completely

New languages (Java, UML, etc) led to **revolutionary procedures…**

But (sadly) education fell **behind…**

Software Projects in the 2000s

In the **2000s** we build integrated collections of continuously **evolving cities**…

Primary focus shift from algorithm design and programming

CS education fell behind so much it became **obsolete**

Developers get more from **training courses** than college

Not much **new development**

Pace of Change is Exhilarating

We have gone from…

- **log cabins**…to **houses**…to **office buildings**…to **skyscrapers**… to building the most **complicated engineering systems** in human history!

Civil engineers took **thousands** of years for this kind of change

- And the most complicated civil engineering products pale in comparison to the complexity of a modern IT system

Electrical engineers took a couple of **centuries**

No way researchers, educators, or engineers could keep up!

Theory, Practice, & Education

What have you learned in college?

How to build houses

General software engineering courses (SWE/CS 321) introduce a few concepts about **buildings**

The way we build software has changed dramatically since the CS curriculum stabilized in 1980!!

What about…

Maintenance…evolution…re-engineering…maintainability…being "agile"

What Can You Do?

As a **developer** or **software engineer**…

- Program **neatly**
- **Design** for change
- Follow **processes** that make change easy

As a **professional**…

- **Listen** when colleagues teach you new things
- Take **training classes** eagerly
- Further your **education** (MS degree)

Goals of this class

1. **Reliability & Testing**
2. Usability
3. Security
4. Availability
5. Scalability
6. **Maintainability**
7. Performance & time to market

First third of SWE 437

Last two thirds of SWE 437

Why focus on these topics??

Most software development is actually maintenance

- or more accurately, "evolution"

Evolution is not as boring as it was in the 1980s

- and the support is so much better!

"We have as many testers as we have developers. And developers spend half their time testing. We're more of a testing organization than we're a software organization."

- Bill Gates of Microsoft

This class teaches modern methods for the two dominant portions of software development:

testing and maintenance