Software Testing and Maintenance

Maintenance and Evolution Overview

Jeff Offutt

Software Maintenance

"When the transition from development to evolution is not seamless, the process of changing the software after delivery is often called **software maintenance**"
– Sommerville, 2004

- Modifying a program after it has been put into use
- Maintenance does not normally involve major changes to the system's architecture
- Changes are implemented by modifying existing components and adding new components to the system
- Maintenance requires program understanding

Importance of Maintenance

- Organizations have huge investments in their software systems - they are critical business assets
- To maintain the value of these assets to the business, they must be changed and updated
- Much of the software budget in large companies is for modifying existing software

Software Changes are Inevitable

- We cannot avoid changing software
  - New requirements emerge when the software is used
  - The business environment changes
  - Faults must be repaired
  - New computers and equipment is added to the system
  - The performance or reliability of the system may have to be improved
- Software is tightly coupled with the environment
- A key problem for organizations is implementing and managing change to their existing software systems

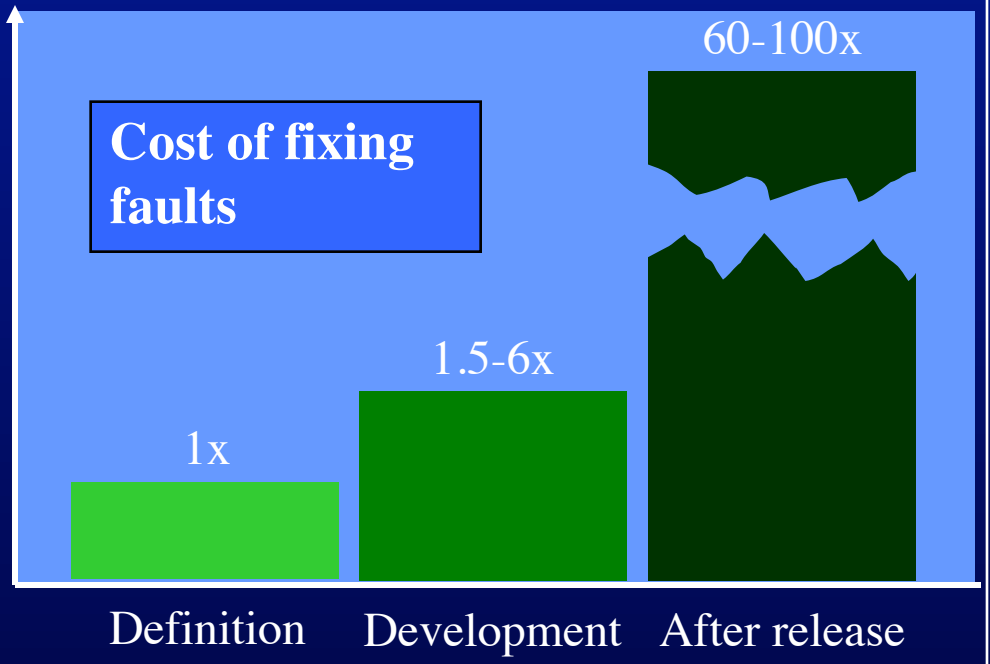
Management Myths

- *Myth*: We already have a book that's full of standards and procedures for building software, won't that provide my people with everything they need to know?
  - *Reality*: The book of standards may very well exist, but is it used? Not if:
    - Are software practitioners aware of its existence?
    - Does it reflect modern software engineering practice?
    - Is it complete?
    - Is it streamlined to improve time to delivery while still maintaining a focus on quality?
- *Myth*: If we get behind schedule, we can add more programmers and catch up
  - *Reality*: Software development is not a mechanistic process like manufacturing. As Brooks said: "adding people to a late software project makes it later"
- *Myth*: If I decide to outsource the software project to a third party, I can just relax and let them build it
  - *Reality*: If an organization does not understand how to manage and control software projects internally, it won't be able to outsource effectively

Customer Myths

- *Myth:* A general statement of objectives is enough to start writing programs—we can fill in the details later
  - *Reality:* A poor up-front definition is the major cause of failed software efforts. If you don't know what you want at the beginning, you won't get it

- *Myth*: Project requirements continually change, but change can be easily accommodated because software is flexible
  - *Reality*: It is true that software requirements change, but the impact of change varies with the time at which it is introduced



Practitioner's Myths

- *Myth*: Once we write the program and get it to work, our job is done
  - *Reality*: Someone once said that "the sooner you begin 'writing code', the longer it'll take you to get done." Industry data indicate that between 60 and 80 percent of all effort expended on software will be expended after it is delivered to the customer.
- *Myth*: Until I get the program "running" I have no way to assess its quality
  - *Reality*: One of the most effective software quality assurance mechanisms can be applied from the inception of a project—the formal technical review. Software reviews are more effective than testing for finding certain classes of software defects.
- *Myth*: Software engineering will make us create voluminous and unnecessary documentation and will always slow us down
  - Reality: Software engineering is not about creating documents. It is about creating quality. Better quality leads to reduced rework. And reduced rework results in faster delivery times. Document should be used if and only if it increases quality

Maintenance Costs

- Usually higher than development costs (2 – 100 times depending on the application)
- Affected by both technical and non-technical factors
- Increases as software evolves
  - Maintenance corrupts the software structure, making further maintenance more difficult
- Aging software can have high support costs (old languages, compilers etc.)

Maintenance Cost Factors

- Team stability
  - Maintenance costs are lower if the same staff stay involved
- Contractual responsibility
  - If the developers of a system are not responsible for maintenance, there is no incentive to design for future change
- Staff skills
  - Maintenance staff are often inexperienced and don't have much domain knowledge
- Program age and structure
  - As programs age, changes degrade the code, design, and structure and they become harder to understand and change

Additional Maintenance Terms

- Maintainability : The ease with which software can be modified
- Ripple effect : Changes in one software location can impact other components
- Impact analysis : Process of identifying how a change in terms of how a change will effect the rest of the system
- Traceability : The degree to which a relationship can be established between two or more software artifacts
- Legacy systems : A software system that is still in use but the development team is no longer active

!! Reality Check !!

Sorry to say, but …

Most of the previous data comes from publications in the 1990s …

Based on knowledge from the 1980s …

When our software was "single-building size" !

How out-of-date is this information for building integrated collections of continuously evolving cities ???

Very !!

Maintenance vs. Evolution

- Software Maintenance
  - Activities required to keep a software system operational after it is deployed
- Software Evolution
  - Continuous changes from a lesser, simpler, or worse system to a higher or better system

Software Evolution

> "Software development does not stop when a system is delivered but continues throughout the lifetime of the system"
> **– Sommerville, 2004**

- The system changes relate to changing needs—business and user
- The system evolves continuously throughout its lifetime
- Modern agile processes emphasize getting a few functionalities running, then adding new behaviors over time

The Pace of Change is Increasing

- Hardware advances lead to new, bigger software applications
- The rate of change (that is, new features) is increasing

How can we deal with the spiraling need to handle change ?

Summary

We will discuss specific ways to deal with maintenance and evolution