Principles of Software Engineering: Why Study Software Engineering?

COSI 120b, Spring 2005
Failure Rates

- Failure is an unexpected result
- Every “system” fails
  - Hard drives crash
  - Transistors burn out
  - Engines fail to start
  - Metal fatigues
  - Etc
- Every system, eventually, fails
Failure Rates

• Computer hardware has a relatively predictable failure curve
Failure Rates

• Hardware may be more prone to failure in its beginning of service
  – Defects in manufacturing
• It may also be prone to failure after time
  – MTBF, general parts wearing out
• Some hardware may be of better quality, and less prone to failure
• The chance of failure is \textit{NEVER} zero
Failure Rates

• What about software?
  – No parts to wear out
  – No manufacturing defects

• What should the software failure curve look like?
Failure Rates
Failure Rates

• Is this picture accurate? Why not?
Failure Rates

• Software is a complex system
  – Complex systems have unforeseen interactions
    • The underlying operating system may change
    • Unexpected conditions may arise
    • A key external library may be upgraded
    • A new feature may be needed
    • A bug fix may actually cause more bugs
Failure Rates
Failure Rates

• Failure for a software system depends on many factors
• As the circumstances surrounding the system change and become more complex, failure trends upward
• Complexity is a key concept
Complexity

• Complexity Theory talks about uncertainty in systems
  – A stable system has known, fixed laws
  – A complex (chaotic) system has uncertainty
    • The laws change, values change
    • Probabilistic versus fixed values and application
  – As complexity increases, the stability of the system decreases
Complexity

• As the system changes, its complexity increases
Complexity

• “x” can be a number of factors
  – Lines of Code (LOC)
  – Number of programmers
  – Number of changes
  – Number of days since release
  – Number of new developers
  – Number of different “functions”
Software Engineering

• Software systems will always be complex

Therefore, the study of software engineering is the study of how to manage complexity in software systems
Software Engineering

• What are the sources of complexity
  – Increasing project size
  – Increasing project requirements
  – Design changes
  – Bug fixes
  – Dependent system changes (operating system, libraries)
  – Increasing team size
  – …
Software Engineering

• What are the solutions to increasing complexity?
  – Process management
    • Identification, testing, integration, agreement
  – Communication
    • Not every developer needs to know the whole system
  – Limitations
    • Not every bug is critical
    • Not every feature is critical
Processes

• Software processes help manage product development and change
  – Traditional models change the system through well-defined stages
    • Waterfall
    • Spiral
  – Non-traditional models breakdown the boundaries between stages
    • Extreme Programming
    • Agile Development
Communication

• Communication between developers helps reduce complexity
  – Coordination of developer efforts
    • Awareness what other developers are working on
    • Status of project code
  – Coordinating representations
    • Bug tracking
    • Unit testing results
    • Smoke tests
  – Customer communication
    • Feedback on prototypes
    • Customer happiness with the project
Limitations

• Does every requirement from the customer get implemented?
  – What about conflicting requirements?
  – What about very difficult requirements that do not have clear value?

• Does every bug get fixed?
  – Cosmetic bugs vs. Show Stoppers

• Prioritization is key
Why Study Software Engineering?

- Why are we studying this topic?
  - As systems get larger, their complexity increases
  - As the complexity of the system increases, unintended consequences increase
    - Unexpected behaviors (i.e. bugs)
    - Inconsistent behaviors
  - So we study software engineering to try to mitigate the complexity