Lecture 7: Developer Collaboration

COSI 120b, Principles of Software Engineering
Importance of Developer Collaboration

• Coordination of effort
  – Ensure that developers are working together, not against each other

• Distribution of knowledge
  – In a large system, no developer can know all of the system

• Understanding of system status
  – Developers need to know the status of the system
    • i.e. what is broken and what is working
Overview

• Collaboration Tools
  – Bug tracking
  – Source control
  – API documentation
  – Unit Tests
  – Smoke Tests
  – Visible Status

• Conclusions

• What’s next
Bug Tracking

• Bug tracking (and task tracking) allow developers to talk about the state of their system
  – What tasks are pending
  – How bugs were resolved
  – Priorities and criticalities of existing issues
• Ensures that issues are not lost
• Tools
  – Bugzilla
  – Scarab
Source Control

• Source (and document) control is necessary for allowing developers to collaborate
  – Propagation of changes
  – Integration and merging of changes

• Imperfect
  – If two changes conflict, source control systems are not smart enough to merge them

• Tools
  – CVS
  – Subversion
  – RCS
  – Visual Source Safe
API Documentation

• Documentation of the source code, especially the external interfaces
  – Assumptions
  – Pre-conditions
  – Post-conditions
  – Side effects
• Must be kept up to date
• Tools
  – JavaDoc
Unit Testing

- Test specific areas of code
  - Subsystems
  - Classes
  - Functions
- Simple sets of tests, exercise one piece of functionality
- Does not, necessarily, test the whole system
- Also documents correctness assumptions
- Continuous testing
- Tools
  - JUnit, NUnit
  - CruiseControl
Smoke Tests

• Does all anticipated functional tests work?
• Often automated, but may be manual
• Run infrequently, compared to unit tests
• Documents the expected behavior of the system
Visible Status

• A smoke test is great, but how does everyone stay informed of the status?
  – Email can work, but requires checking of email
  – The Lava Lamp

• Constant communication is important
  – A lagging indicator may be too late
Conclusions

• Developer communication breakdowns are a major source of system complexity
  – Committing conflicting changes
  – Adding features to bugging subsystems
  – Not knowing whether or not a subsystem is stable
  – Not knowing whether or not your code broke a subsystem
What’s Next

• Performance Engineering
  – How do we measure the performance of a system?
  – How do we specify the performance goals of a system?
  – Why do we keep performance in mind?