Software Assurance

Session 13
INFM 603

Bugs, process, assurance

- Software assurance: quality assurance for software
  - Particularly assurance of security
- Bad (buggy, insecure software) comes not from discrete, unpredictable, uncontrollable human errors, but from bad processes.
- We can't control human fallibility, but we can control processes

Today: from hands-on to meta-process

- Today's session:
  - Hands-on vulnerability detection and correction (**bug-hunting**)
  - Methods and **processes** for preventing, correcting, and managing bugs and vulnerabilities
  - **Models** for measuring and improving processes

Bug-hunting in a simple web-app

- Webapp allows users to log in and record their SSN (see separate code, running site)
- Written (directly) in PHP, backed by MySQL database
- Find the vulnerabilities!

(switch to browser)

Testing

- Manual vs. automated testing
  - What are their pros and cons?
  - How can you design to facilitate automation?
- Unit, integration, and system testing
  - Test: components separately; integrated subsystems; then full system for implementation of requirements
  - How to design for this model of testing?
- Regression testing, and test-driven development
  - Keep bugs fixed; keep non-bugs absent; test, then code

(jump to example)

Coding practice

- Coding standards
  - Layout: readable code easier to debug
  - Practice: avoid common pitfalls, build code in expected manner
- Code review
  - Computers don't criticize; other coders do!
  - Formalized in pair programming
- Code less
  - Bugs per 100 lines surprisingly invariant
  - Maximise re-use of code, yours and others

Design, proof, etc.

- Care in design of product
  - Bad design leads t o messy workarounds; messy workarounds lead to bugs and vulnerabilities
- Formal and semi-formal proofs of correctness and code checkers
- Use appropriate libraries, levels of abstraction
- Catch mistakes early!
  - The later in the development process you find bugs, the more difficult and expensive they are to fix.

Debugging is harder than coding!

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it"

– Brian W. Kernighan and P. J. Plauger, *The Elements of Programming*

Bug hunting and vulnerability spotting

- Bugs are your code not behaving as you designed it.
  - Many can be found by testing for expected behaviour
  - Users report, workaround bugs
  - Maximum damage is normally loss of functionality
- Security vulnerabilities are someone smart making your system doing something unanticipated
  - Difficult to test for in routine way
  - Valuable knowledge to others; may not be reported!
  - Maximum damage: ???

Security requires experience

- Develop knowledge of possible types of security vulnerability (buffer overflow, SQL injection, etc.)
- Brainstorm possible vulnerabilities
- Act as or employ white-hat hacker
- Monitor security updates for packages you use
- Reduce attack surface area
- Learn from the mistakes of others!

(switch to CERT)

Managing bugs and vulnerabilities

- Even with good processes, (alleged) bugs will still turn up in system-level products, both in development and in deployment
- Tools for managing, tracking, performing statistics on such bugs and vulnerabilities essential, particularly on large projects.
- A core tool is the bug tracker

(jump to Bugzilla)

Developing and measuring process

- Heroic age of software development: small teams of programming demigods wrestle with many-limbed chaos to bring project to success, or die in the attempt.
- Kind of fun for programmers ...
- … not so fun for project stakeholders
- Current age of managed development:design controllable, measurable, repeatable processes for managing software development.

Models for software quality assurance

- Models and standards developed for software assurance, after pattern of other quality assurance standards (e.g. ISO 9000)
- Models don't tell you how to write good software
- … and they don't tell you what process to use to build good software
- They provide a yardstick for measuring the quality of your process management
- They measure whether **you** can measure your process

CMMI Maturity Levels

CMMI has five levels of process maturity (with process areas to verify at each level):

1. Initial
2. Managed (e.g. Measurement and Analysis)
3. Defined (e.g. Organizational Process Focus)
4. Quantitatively Managed (e.g. Quantitative Project Management)
5. Optimizing (e.g. Causal Analysis and Resolution)

ISO 15504

ISO 15504 has six capability levels (each practice develops through these levels):

1. Not performed
2. Performed informally
3. Planned and tracked
4. Well-defined
5. Quantitatively controlled
6. Continuously improved

Qualitative, Quantitative, Improved

Both CMMI and ISO 15504 embed the same sequence:

1. Qualitative management (e.g. process for code reviews, testing, etc.)
2. Quantitative management (metrics of performance)
3. Improvement (change process, check with metrics that improvement in quality results)

Example: MS SDL process

Process: Security Development Lifecycle (SDL)

Metric: Bug count (critical and serious, within year of release), on product versions before and after adoption of SDL.

Result:

Product	Pre-SDL	Post-SDL
Windows 2000/2003	62	24
SQL Server 2000	16	3
Exchange Server 2000	8	2

Applying Bug Count

(jump to Bugzilla)

- How good a metric of software quality is “number of oustanding bugs”?
- Are there other reasons you (as a manager) might want to introduce it as a metric?
- What would you expect to be the most immediate effect if you introduced it as a metric (and tied programmer appraisal to it)?