CS-480
Midterm Review

April 29, 2013

1 Quality Insurance

1.1 Type of bugs

-**Race conditions and deadlocks**: Racecondition is the beahvior of an electronic of software system where output is dependent on the sequence or timing of other uncontrollable events. **For example:** Multi threads application has many threads that have the access to the same shared resource and can change these resource at the same time which make the output becomes very unpredictable(bugs).
- **Deadblock**: is a situation in which two or more competing actions are each waiting for the other to finish, and thus neither ever does. 'For example: Hold an Wait situation, a process ius currently holding at least one resource and requesting additional resources which are being held by other resources
- **Library Misuse**:
- **Logical error**: a logic error is a bug in a program that causes it to operate incorectly giving the wrong output, but not to terminate abnormally(crash). To solve this bug, we should output all the variables to a file or on the screen to define error in the code
- **Usability**: A usability bug is any unintended behavior by the product noticed by and impacting the user. **For example:** the Wii remote control detect the wrong motion from the users.
- **Performance Defects**: The application doesn't have the performance that qualifies the requirement. Performace defects involves things like scalability, reliability and resource usage.
- **Error-handling error**:
- **Requirement bugs**: misunderstanding the requirement of the application

1.2 Terminology of bugs:

- **Fault**: static problem in code, eg: unreachable code, in therac, entered data is invalide, increment errorCounter instead of setting it to one - **Error**: dynamic problem at runtime, eg: index out of bounds in array, also therac 25, the error-Counter got overflows and goes to zero - **Failure**: is the problem of the system. the machine. Eg: Server got shutdown, or Ther25 believes it has valid data when it doesn't and shoot a beam of radiation. - **Hazard** is the resulting risk that actually happens. Eg: power grid goes down, patient dies from radiation overdose - **Faults in dead code**: those does not cause error - **Multiple fauls:** may cause an error together - **Error:** Error does not always result in a failure to stop the system like the example of server shutdown. Instead, the system or application might still produce the output. Eg: Therac20- has the same faults, and the same error, but there was a hardware override that prevented the failure

1.3 The cost of defects

- 20% of modules cause 80% of defects. and 50% are defect free
- **cost of repair** is potentially increase from 5x to 10x from the requirements to detection phase, somtimes evenmore
- **Rule of thumb**: 3x to 10x per phase in software lifecyle.

1.4 Kinds of Quality Assurance

- **Types:**

- Testing:
- Inspection:
- Program analysis

-how to stop defect in the first place by Quality Assurance

- getting the requirement right
- getting the design right
- implementing the design properly

- you would never never detect all defects, or prevent all defects
- **V & V**: stands for verification and validation

- verification is checking wherether the product matches its requirements
- validation is checking whether the product accomplish its goal
- testing is used to **verify** the program match the specified test
- **validate** is used to see if the specified tests are testing the right thing

What kind of precisions of Quality Assurance

- True positive: Quality Assurance detect these, and they exists
- True negative: Quality Assurance says no problem, and there is no problem
- False Positive: Quality Assurance detects problem, but there is no problem
- False Negative: Quality Assurance detecs no problems, but problems exists

-Note: False Positive, when Quality Assurance detect no problems, but there is problem takes a lot of time to investigate because too many of them relative to true positive.

1.5 How to evalutate Quality Assurance

- **Overall** Cost: Cost of the techniques. Sunk Cost, and Recurring Cost
  - **Sunkcost**: are cost of purchasing tools and setting them up.
  - **Recurring Cost**: costs are the cost using the tool and maintaining any resources it needs.
- **Bugs Found**: what classes of bugs are found? What is the relative importance of these classes of bugs. What are their relative expensive if the leak into the production code. eg; UI flaw will be very cheap comparing to security bugs(getting sued!)
- **Bug missed** What classes of bug this technique of Quality Assurance missed. What the cause or reason
- **Techniqueover overlap** What other Quality Assurance techniques might find the similar type of bugs
- **Cost of fix** what are the cost of fixing a bug once it is found with this technique. 3 things to consider
  - cost to reproduce the bug for developer
  - cost to find the root cause of the bug
  - cost to fix the issue
- **Stopping Point** determine when to stop using the technique
- **Intangible benefits**: Question for for this....

1.6 Quality Assurance Plan

- A plan for doing QA helps us to decide what techniques to use, before it's too late.
- **Tetsting techniques**

- what will be testing
- how will testing take place
- when will testing take place, when it ends
- who will write the test, who will run them
- why do we believe the set of test is good

- In our plan, we should include the other techniques as well. For example; program analysis, or inspections. We should ask and answer the same question above for testing
- One more important thing is provide the evidences why this set of QA techniques is good for your project
-QA should be consious decision, not an accidential oversight

2 Testing

2.1 Terminologies and Definitions

- **Definition**: Direct execution of code on test data in a controlled environment
- **Importance**: It only reveals the inconsistencies with specifications. It does not say the program is wrong or specification is wrong.
- **goals of testing**:

- Reveal specific errors.
- Assess the overall quality.
- Verify the legal standards.
- Clarity the application specs and learn more about the program

- Categorize of testing

- Visibility
  - Black Box: These test are made by looking at the specification, withou looking at the internal code
  - White Box: These test are made by the knowledge of the code which are the internal control flow and structures
- Automation:

  - Manual testing: keyboard pouding by human
  - Automated testing: run by a machine
  - Semi-automated testing:

2.2 Unit Testing

- **Definition**the Ünitïmplies the level that we are at. In the industry, unit testing is usually considered with method-level, but same principles can be applied at any level which are

- entire system
- subsystem
- single class
- single method

- **Properties:**Unit testing is:

- manually to create
- white box
- could be blackbox if devs test to an API

- **Test Scarffolding**: is a temporary sub-structure software that is used to test some input.

- provide a way to run the test
- JUnit is a test scaffold:
- Driver: the component that calls the test
- Stubs: Component that the test calls and return canned data(the actual code that we wrote) **need more explanation**
- 3 phases of the test
  - Setup phase: Create all input for the test, or any stubs
  - Execute phase: Run the unit under test with the inputs
  - Verification phase: Check output against exepcted outputs
- Notes:
  - provide a different test for each different sets of input

  - need to control the inputs, includes database things
  - build stubs to handle these, stubs don't have to actually work, they are fake data
  - verify all parts of the output.
  - limit exercise phase to unit, so we can identify the faulty unit(**explanation**)

2.3 Regression

- the ability to re-ren your test at a later date
- it will prevent the old bug to recurring
- This is very low cost since you can store and run again at any moment
- Everytime you fix a bug, write the test for it, so you can run it in regression, so you don't re-create that bug again.

2.4 Test-driven Development

- There are 3 ways you can develop the testing by this way

- Write your test first, then develop your code
  - The tests your write must RUN and they should FAIL
  - Ensure that it actually tests something
  - Then write code to make it pass
- Develop your code and then write your test very shortly after
  - Upside: code is written first. Make stubbing easy because you know what the interface will be
  - Downside: Code is written first. You tend to write code to pass it easily
- Having a dedicated testee write the tests blackbox style

-There are major benefit to this development technique: you will be getting into a testing mindste and purposely trying to break your code

2.5 Coverage

Definition: is the proportion of code covered by your test -Kind of coverage

- line coverage
- branch coverage
- path coverage

- 100% can be very difficult because there are a lot dead code, and code is not important for testing
- However, coverage lets us see what code we have not yet written a test for

2.6 Input selection

- Do a structual analysis of code and select inputs to exercise each branch or path through the code
- Use equivalence classes. For example: list , list of 3 elements, list of 1 elements, empty list , null list, list of list.
- Randomly generate input.
- Automatically generate inputs across space of inputs.

3 How to report bugs

3.1 best practice for reporting bugs

- Provide the reproducing steps
  - provide detailed and simple steps how to reproduce the bugs
  - helps with debugging
- Be non-atagonistic
  - Don't blame developers. everone write bugs
  - Don't blame the testers. Testers frequently bring bad news.
- Avoid using defects in performance evaluation(for example: give money to testers if they found find bugs, or punish the devs for bugs)
  - Do that makde devs not take risks to add features
  - Testers can create blakcmarket in bugs
  - No one will try to find expensive bugs cuz there are eays one to find
- identify the root cause
  - how could this bug be prevented
  - Could we discover it earlier
  - any other related bugs. If yes, then fix them. Don't make tester find them all .

3.2 Bug Report Outline

- Repor step: detailed and simple
- Autal ouput comparison with the expected output
- Severity: how severe is the bug? does it crash the system or there is work around
- Priority: how important is the bug? Does it need to be fixed right away
- Status: what is the current status
  - unconfirmed
  - New/ confirmed
  - assignend
  - resolved
  - reopened
  - verified
  - closed
- Resolution:
  - Fixed
  - won't fix
  - works for me
  - duplicate

4 Inspection

4.1 Definition, Major benefits, inspection vs testing, formal vs informal Inspection

- **Definition:** Inspections are reviews of software artifacts, the reviews can be formal process or an informal one
- The artifact could be code designs, or specification
- Code reviews are the most common form of inspections
- Majors benefits are:
  - getting multiple perspectives on the artifacts
  - sharing knowledge among team members
  - finding defects earlier in the process
  - Reducing rework and later effort

  - Finding defects that are otherwise hard to detect through automated techniques

- inpsection vs testing
  - test a wider range of input
  - generally better for **checking quality attributes ranther than functional defects**
  - particularly gooat at **attributes such as maintainability, evolvability, and scalabality**

- Formal vs informal inspection
  - in both, we can use check list so that reviewers dont forget what to look for
  - formal reviews takes place as part of meeting with several people
  - informal reviews are usually online only and may invole only one or two reviewers
  - While formal process can find more defects, it is definitely more expensive

4.2 Formal inspection

- An example formal review process: Fagan-style inspection
- this techniqure require a group of 3 to 7 people
- Team size is larger for documents review and smaller for code review
- Team members have to read code in advance
- Expect people to be able to review 150 to 200 line of code per hour, or 3 or 4 pages of code
- Team members have a meeting to discuss, and each member has a role
  - moderator organizes the process and keep everyone accountable. The moderator has to ensure the outcome of the meeting happens
  - recoder record the log of what is discussed
  - reader presents the material to the group, but this can't be the author
  - the author can describe the intent of the code and reason for it, but otherwise keeps quiet during the meeting. Sometimes, authour might not attend at all
- the metting itself should take no longer than 2 hours because the brain of human can only process many line of code in an amount of time
- after the meeting, th moderator folows up with the author to ensure all changes are mde

4.3 inFormal inspection

- The process start with the author sends a patch file to a reviewer:
  - the patch file should be as small as possible, no more than 200 LOC
  - Larger changes should be broken into multiple patches
- the reviewers have to use an online tools to add comments
- the author makes suggested changes or responds to the comments
- when the reviewer is satisfied, the author is cleared to commit their code into the repository

5 Reliability

5.1 Definition

- **Definition**: Reliability is the realization that, despite your best efforts,error will happen. How do we prevent them from causing failures?
- We need to do an analysis of the inherent risks involved in the system
- These risks are inherent because they provide us with other benefits
- **inherent risk example**
  - safety critical devices: Therac25, radiation can kill people, but we use the them to save live of cancer patients
  - Interconected enterprises: example is the electric grid which are interconected to each other, we have to face cascade failures with these kind of inherent risk
  - End-user interaction: Direct users interaction allow users to have more features and more flexibility. However, we have higher risk due to the uninformed individuals taking actions with high consequence. This could come from the UI defects or user error
  - Electronic decision making system: we can save time by using electronic decision like in automobiles or trading. But the consequence happen much faster, perhaps even too fast to be able to mitigate them

5.2 Risk Mitigation

- If we cannot avoid risks entirely, we must consider how to mitigate them. We can either be preventative or reactive

- preventative

  - Inspections can identify before they enter production
  - Testing can be used to identify error
  - Can also use other coding practices and measurements to prevent faults

- reactive
  - dynamically adjust when failures are detected in order to avoid hazards
  - Fault tolerant system (handle multiple errors) prevents errors causing system wide failure

- Risk management
  - involves indentify the risks, tracking them, and mitigating them
  - might have a risk manager incharge of this
  - process might involve listing all the list or listing the top ten bug along with the mitigation techniques
  - might also perform a fualty tree for safety-critical system.

- Risk Coverage
  - There is no way to completely cover risk
  - There will be always be known risk(identified and hopefully be mitigated), and unknown risks(have not been identified)
  - it is frequent better to at least find as many known risk as possible, even if not all of them can be mitigated

6 Program analysis

6.1 Definition

- Program analysis tools read or run code to automatically detect defects or provide further understanding about the code

- Static analysis is the form which read the code(static: doesn't move)
- Dynamic analysis is the form which runs the code

7 Static analysis