Software Inspections

CSCE 747 - Lecture 22 - 03/31/2016

Low Tech Approach to a High Tech Problem

- Too many dependencies to test the existing classes?
- Code too complex to apply analysis?
- Have you tried reading the source code?
  - That is - have you performed an **inspection**?
    - Manual, collaborative review of project artifacts.

Software Inspections

- Can check properties that are hard to verify dynamically.
- Flexible approach:
    - Code does not need to execute.
        - Can be applied before code is complete.
    - No limitations regarding scalability, data structures used, pointers, etc.
    - Can be applied to *any project artifact*.
- Effective in revealing faults earlier in development than testing.

Social and Educational Benefits

- Creates incentive to build better artifacts.
  - It is embarrassing when others find and discuss flaws in your work.
    - Goal should not be to embarass, but that is a common side-effect.
- Effective way to form and communicate organizational standards.
  - Engineers tend to be quick to share experience and knowledge relevant to a shared problem.
  - When a new practice is introduced, inspections are a quick way to share awareness of it.

Social and Educational Benefits

- New staff can be immediately productive.
  - Can self-inspect against the checklists used in inspection.
  - Taking part in a group inspection can be a fast training method.
- Social and educational benefits should be taken into account when designing inspection process and forming teams.

Software Inspections

- Characterized by:
  - **Roles** - who are the inspectors?
  - **Process** - how the inspectors organize and synchronize their work.
  - **Reading Techniques** - how inspectors examine an artifact.
- Not a full-time job:
  - Productivity drops after two hours of work.
  - No more than two inspection sessions per day.

The Inspection Team

- Inspectors are usually a combination of existing team members:
  - Junior and senior engineers, test engineers, project managers, analysts, architects, technical writers.
- Efficacy lower if developers feel like they are being judged.
  - Senior engineers and managers usually pulled from unrelated projects to ensure unbiased inspection.

The Inspection Team

- Inspection team should balance perspectives, knowledge, and cost.
  - A developer is most knowledgeable about their own work, but may be blind to weaknesses in their work.
  - Inspection benefits from differing perspectives and expertise.
- Cost grows with the size of the team.
  - Classic - 4 to 6 people. Modern - pairs may be best.
  - Levels of inspection: simple with one inspector, complex with two, larger groups for special occasions that need particular expertise.

Inspection Team Sizes

- Inspection team members should never be responsible for the artifact being inspected.
  - Often borrowed from another team entirely.
- Simple inspections:
  - Single junior engineer.
  - Combines inspection and training.
- "Standard" inspections:
  - Pair of a junior and a senior engineer.
  - Senior engineer acts as a moderator.
    - Organizes the inspection.
    - Responsible for final results.

Inspection Team Make-up

- Larger groups (four to six) used for complex modules, looking for integration problems.
  - A senior engineer or manager organizes the process and assembles final results.
  - Mix of senior and junior engineers read and inspect the artifact, discuss possible issues.
  - Developer of the artifact is often present to answer questions or explain design choices.
  - Often used when particular specialties are needed to understand parts of the module.

Reward Mechanisms

- Developers must be motivated to collaborate.
- Reward mechanisms can influence attitude.
    - Must be carefully designed to avoid negative effects.
        - Assessment of fault density that includes faults revealed by inspection might encourage developers to hide faults.
        - Incentives that naively reward faults found can also punish developers that bring high-quality code.

Inspection Process

- Systematic, efficient, repeatable process.
- Expensive and not incremental.
  - Reinspection costs as much as initial inspection.
  - Should be placed to reveal faults early.
    - Do not inspect if still under active construction.
    - But does not need to be completely finished.
- Activities can take place at different phases:
  - Check consistency and completeness of comments before testing.
  - Check for semantic consistency of code after testing.

Inspection Process

- Three main phases - preparation, review, follow-up.
- **Preparatory Phase**
  - Inspectors check that artifacts to be inspected are ready.
  - Assign inspection roles.
  - Acquire information needed for inspections.
  - Plan individual inspection activities.
  - Schedule inspection meetings.

Inspection Process

- **Review Phase:**
  - Artifact reviewed individually, then in teams.
  - Artifact closely examined for issues by checking the contents against one or more checklists.
    - Based on fault types, style expectations, regulations, practices, etc.
- **Follow-Up Phase:**
  - Developers notified of results.
  - Developers and test engineers identify faults to fix, and create a schedule for making changes.
  - Follow up checks may be scheduled.

Checklists

- Summarize experience accumulated in previous projects.
- Contains a set of questions that help identify faults in the artifact.
  - Updated regularly to add new checks and remove obsolete elements.
- Length and complexity depends on use.
  - Should be completable in one review session.
  - Long list of simple questions for syntactic review.
  - Short list with complex questions for semantic review.

Checklists

- Can be applied to a variety of artifacts.
  - Source code, requirement specification, design description, test suites, reports.
- Can assess functional correctness, consistency, completeness, ambiguity of natural language, compliance with regulations, etc.
- Structured hierarchically, used incrementally.
  - Simple checks conducted by single inspectors.
  - Complex checks conducted in group reviews.

Java Checklist - Single Inspector

- File Header
  - Are the following included and consistent?
    - Author and current maintainer.
    - Cross-reference to design entity.
    - Overview of package structure, if the class is the entry point of the package.
- File Footer
  - Is there a revision log to minimum of one year or most recent point release?
- Import Section
  - Is there a brief comment on each import with the exception of standard java.io.* or java.util.*?
  - Does each imported package correspond to a dependence in the design documentation?

Java Checklist - Single Inspector

- Class Declaration
  - Is the constructor explicit?
  - Is the visibility of the class consistent with the design document?
  - Does the JavaDoc header include:
    - A one sentence summary of class functionality?
    - Guaranteed invariants (for data structures)?
    - Usage instructions?
- Class
  - Are names compliant with the following rules?
    - Class or interface: CapitalizedWithEachWord
    - Exception: ClassNameEndsWithException
    - Constants (final): ALL_CAPS_UNDERSCORES
    - Field Name: capsAfterFirstWord
      - Must be meaningful outside of context.

Java Checklist - Single Inspector

- Methods
  - Are names compliant with the following rules?
    - Method name: capsAfterFirstWord
    - Local variables: capsAfterFirstWord
      - Names may be short (e.g., i for integer) if scope of declaration and use is less than 30 lines.
    - Factory method for X: newX
    - Converter to X: toX
    - Getter for attribute X: getX();
    - Setter for attribute X: void setX;

Java Checklist - Inspection Team

- Data Structure Classes:
  - Does the class keep a design secret?
  - Is the substitution principle respected?
    - Instance of class can be used in any context allowing an instance of superclass or interface.
  - Are methods correctly classified as constructors, modifiers, and observers?
  - Is there an abstract model for understanding behavior?
  - Are the structural invariants documented?
- Methods:
  - Are method semantics consistent with similarly named methods?
    - (put(O) matches put(O) use for other classes)
  - Are usage examples provided for nontrivial methods?

Java Checklist - Inspection Team

- Fields
  - Is the field necessary (cannot be a local variable)?
  - Is the field protected or private?
    - If not, is there justification for public access?
  - Are there comments describing the purpose of the field?
  - Are there any constraints or invariants documented in the field or class comment header?
- Design Decisions:
  - Is each design decision hidden in one class or a minimum number of closely-related classes?
  - Do classes encapsulating a design decision unnecessarily depend on other design decisions?
  - Are adequate usage examples provided?
  - Are design patterns reference and used when appropriate?
    - If so, does the code match the pattern?

Checklist Organization

- Consists of a set of features, and items to be checked for each feature.
  - Directs the reviewers' attention to the right set of checks during review.
  - Items to be checked ask whether certain properties hold over the artifact.
    - A positive answer should indicate compliance.
  - Inspectors check "yes" or "no", and add comments explaining their decision.
    - Should include the location where a violation occurs.

Checklist Items

- Should not include items that can be easily checked with automated analyses.
    - A copyright statement could be automatically included, and doesn't need to be checked.
    - Maintainer name might not be auto-inserted and can be out of date.
- Properties should be objective and unambiguous.
    - "Are comments well-written?" is subjective.
    - "Is there a one sentence description of class functionality?" is not.

Checklist Items

- The items should be tuned to the type of artifact being inspected.
- What kind of “faults” can be inserted in that artifact?
  - In requirements, a specification can be wrong.
    - It can also be inconsistent, written ambiguously, incomplete, unrealistic to implement.
    - Could have a checklist to evaluate the writing style used to draft the specification.

Specification Writing Style Checklist

1. Have you varied the stress pattern in a sentence to reveal alternative meanings?
2. Could you commit to implementing this requirement within a week?
3. If a term is defined elsewhere, can you substitute the term for its definition?
4. When a graphical element is described in words, can you sketch a picture of it?
5. When a picture describes a graphical element, can you redraw the picture in a form that emphasizes different aspects?
6. When there is an equation, can you expressing the meaning of the equation in words?
7. When a calculation is specified or implied in words, can you expressing it in an equation?
8. When a calculation is specified, can you work through at least two concrete examples by hand?

Specification Writing Style Checklist

9. If there are statements that imply certainty or are used to persuade the reader, is evidence provided to back those assertions?
10. Are vague words used that need clarification?
11. Are non-committal words used?
12. Are lists complete?
    a. If “etc” is used, is the meaning clear?
13. If assertions are made, do they contain unstated assumptions?
14. Are there requirements without examples (or too few/too similar examples)?
15. Are vague verbs used?
16. Is passive voice used? Passive voice does not name an actor.
17. Are comparisons made without clearly stating what is being referred to?
18. Are pronouns clear to both the writer and the reader?

Test Plan Checklist

- Items to be tested or analyzed:
  - For each item, does the plan include a reference to the specification for that item?
  - For each item, does the plan include a reference to installation procedures for the item, if any?
- Test and analysis approach:
  - Are the techniques to be applied cost-effective for items of this type?
  - Do the techniques to be applied cover the relevant properties cost-effectively?
  - Is the description sufficiently detailed to identify major testing and analysis tasks and estimate time and resources?

Test Plan Checklist

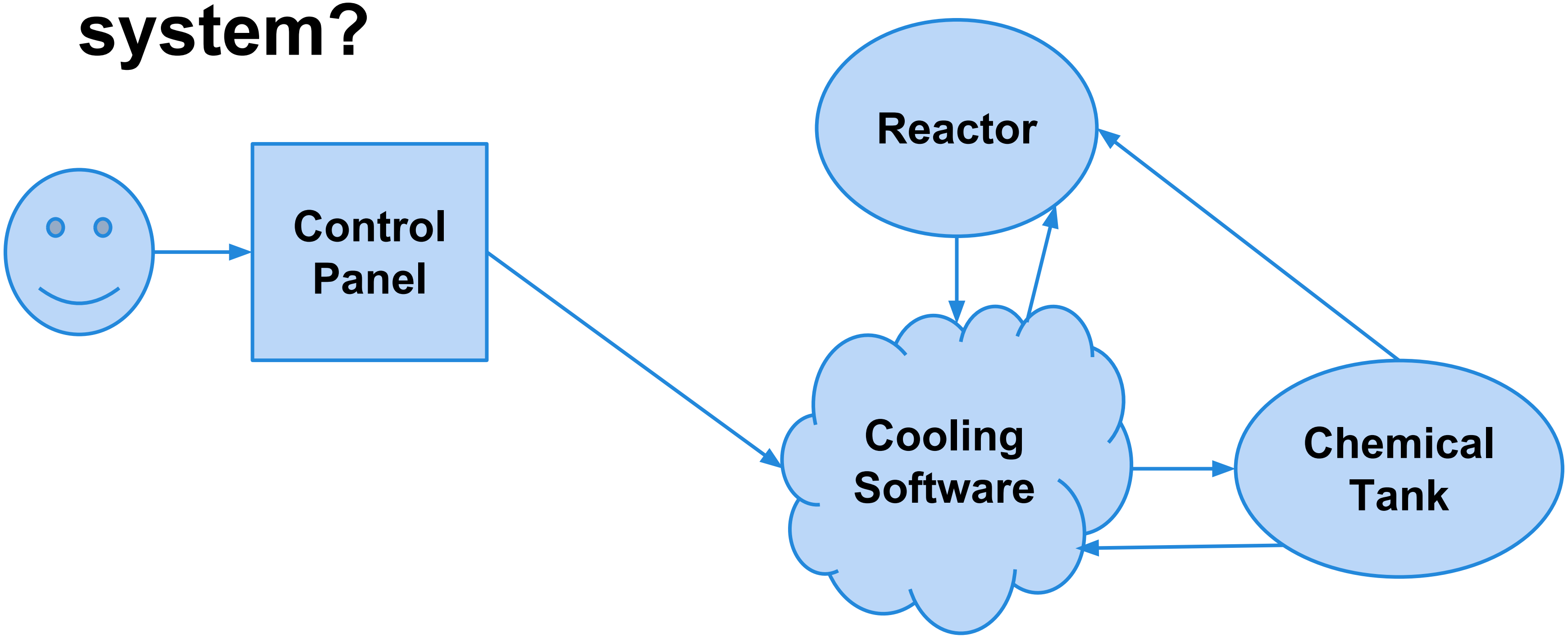
- Pass/Fail Criteria:
  - Do the criteria clearly indicate the pass/fail conditions?
  - Are the criteria consistent with quality standards specified in the test and analysis strategy?
- Suspend/Resume Criteria:
  - Do the criteria clearly indicate threshold conditions for suspending test and analysis due to excessive defects?
  - Do the criteria clearly indicate conditions for resuming test and analysis after suspension and rework?
- Risks and Contingencies:
  - Are the following risks addressed?
    - Personnel risks, technology risks, schedule risks, development risks, execution risks, risks from critical requirements.

Test Plan Checklist

- Contingency Plan:
  - Is each identified risk adequately considered in the contingency plan?
- Tasks and Schedule:
  - Do the tasks cover all aspects that need to be tested?
  - Is the description of the tasks complete?
  - Are the relations among tasks complete and consistent?
  - Is the resource allocation and constraint list adequate?
  - Does the schedule satisfy all milestones?
  - Are critical paths minimized?

Domain-Specific Checklists

What problems and test scenarios can we anticipate in the automated cooling system?



Checklist for Embedded Systems

1. Is the software’s response to out-of-range values specified for every input?
2. Is the software’s response to not receiving an expected input specified?
   a. Are timeouts provided?
   b. Does the software specify the latency of the timeout?
3. If input arrives when it shouldn’t, is a response specified?
4. On a given input, will the software always follow the same path through the source code?
5. Is each input bound in time?
   a. Does the specification include the earliest time at which it will be accepted and the latest time it will be considered valid?
6. Is a minimum and maximum arrival rate specified for each input?
   a. What if input arrives too often?
   b. Is there a capacity limit on interrupts?

Checklist for Embedded Systems

7. If interrupts are masked or disabled, can events be lost?
8. Can software output be produced faster than it can be used by the receiving system?
   a. Is overload behavior specified?
9. Can all of the outputs from the sensors be used by the software?
10. Can input received before startup, while offline, or after shutdown influence the software's startup behavior?
    a. Are the values of any counters/timers/signals retained following shutdown? Is the earliest or most recent value retained?
11. In cases where performance degradation is the chosen error response, is the degradation predictable?
12. Are there sufficient delays incorporates into error-recovery responses?

Generality of Checklists

Domain-specific checklists focus on common pitfalls of one domain, but hold important lessons for other problems.

Use checklists to set expectations, but not to limit analysis of an artifact.

Checklists are Effective

On two NASA spacecraft projects, 192 critical errors were found during integration and testing.

- 142 of those were found and addressed after using a simple safety checklist.
- Most were problems with unexpected input.
  - Unexpected values, and more importantly, unexpected timing (recall the embedded system checklist).

Pair Programming

- Practice associated with agile processes.
- Two programmers work together at the same computer.
  - While one types, the other inspects the code.
  - The pair actively discuss implementation decisions.
  - The developer not typing can also plan ahead and think about design alternatives.
  - Merges development and inspection.
    - Less code written, but can be more effective by producing higher quality code.

Pair Programming

- Inspection not driven by checklists, but based on shared programming practice and style ideas.
- Inspector and coder swap roles, and take leadership on parts of the system.
  - Code is “owned” by the team, rather than by individual programmers.
  - Requires attitude of “egoless programming”
    - Criticism of artifacts is not regarded as criticism of authors.

Activity

- You are inspecting the source code for the Graduate Record and Data System (GRADS).
  - A system that graduate students can log into and use to view their transcript or a summary of their progress towards graduation.
- Your current task is to inspect the class “Session”.
  - A class used to track information about a user of the system, as well as to store the contents of databases in memory.
- You have been provided with a checklist of common Java code style issues, and are to inspect the Session class against that list.

- **Working in pairs, document below which checklist items were not met, and why they were not met. Provide advice on how to address that shortcoming.**

Activity - Failed Checklist Items

- File Header
  - Are the following included and consistent?
    - Author and current maintainer.
      - **No - maintainer is not included.**
- Import Section
  - Is there a brief comment on each import with the exception of standard java.io.* or java.util.*?
    - **Imported class CourseTaken has no comment.**
- Class Declaration
  - Is the constructor explicit?
    - **No constructor is included.**
  - Is the class protected or private?
    - If not, is there justification for public access?
      - **No justification provided. Perhaps this should be at least protected.**

Activity - Failed Checklist Items

- Methods
  - Are names compliant with the following rules?
    - Local variables: capsAfterFirstWord
      - **Variable “toreturn” violates naming convention.**
    - Getter for attribute X: getX();
    - Setter for attribute X: void setX;
      - **getUser and setUser should be getCurrentUser and setCurrentUser.**

Activity - Failed Checklist Items

- Fields
  - Is the field necessary (cannot be a local variable)?
    - **userId can just be a local variable (or eliminated entirely - it is passed into each method that uses it.**
  - Are there any constraints or invariants documented in the field or class comment header?
    - **No, but does there need to be?**
      - **I.e., do not blindly apply checklist criteria.**

We Have Learned

- Inspections are one of the most flexible analysis techniques.
  - All documents can be inspected.
  - Inspection can take place before code can execute.
  - Can “scale” to any complexity and has no limitations on the type of programs that can be studied.
- Teams compare artifacts to document-specific checklists.
  - Can check functional correctness, writing style, completeness, consistency, regulatory compliance, etc.

Next Time

- Testing as we near release.
  - System, acceptance, and regression testing.
  - Chapter 22

- Homework:
  - Assignment 4 - due April 5!