CBOE Code Inspection Process

Introduction

This document specifies the process to be used for inspecting/reviewing code. The inspection is performed in an effort improve the quality of the software being constructed. The specific goals of the code inspection are to:

- To find errors that automated tests, such as compilation and unit testing, do not find.
- To ensure that all required and/or applicable programming standards are followed.
- To ensure that code meets the design requirements and is traceable to the low-level software system design.
- To share knowledge between individuals, and to transfer knowledge from the more experienced individuals to the less experienced individuals.

There are several techniques that can be used to review code [Objectory41]:

- Inspection. A formal evaluation technique in which the code is examined in detail. Inspections are considered to be the most productive review technique, however it requires training, and preparation.
- Walkthrough. An evaluation technique where the author of the code, leads one or more reviewers through the code. The reviewers ask questions, and make comments regarding technique, style, possible error, violation of coding standards, and so on.
- Code reading. One or two persons read the code. When the reviewers are ready, they can meet and present their comments and questions. The meeting can be omitted, however, and reviewers can give their comments and questions to the author in written form instead. Code reading is recommended to verify small modifications, and as a "sanity check."

The CBOE Code Inspection Process will use an informal code walkthrough approach.

Implementers are expected to use any tools and techniques necessary to produce quality code prior to the code inspection process. The code inspection process should not be used to identify problems that are easily prevented or corrected through use of tools or the application of coding standards.

Phases

Given the iterative, phased development process that we are using, each code inspection must be held under the context of the current phase. That is, until the final phase, not all functionality is expected to be implemented.

Required Material

1. Class Diagrams: The class diagrams show the inheritance relationships of the various classes and the services (methods) that they provide. (hardcopy or online)
2. Class Specifications: The class specifications give the full details of the classes. (hardcopy or online)
3. Design Sequence Diagrams: The class-based design sequence diagrams show the detailed design of how a class provides the functionality required by the use case sequence diagrams. (hardcopy or online)
4. Source code (hardcopy or online)
5. Appropriate programming guidelines (e.g. CBOE Java Programming Guidelines document)

Participants

Each code inspection requires a minimum of three people. The participants should come from people who worked on the overall design of the system, people working on the detailed design of interconnected parts of the system, people who may implement the detailed design, and people responsible for testing the system. All participants are called `inspectors,' though some inspectors will have other roles as well.

- Moderator: The moderator will schedule and monitor the inspections. The duties of the moderator will be to:
    1. Assign inspectors to specific subsystems, packages, or classes.
    2. Focus the inspection meeting on finding defects in the product under inspection.
    3. Classify defects as major or minor.
    4. Assign remedial actions to the author(s).
    5. Determine whether a follow-up inspection is required and, if so, verifies that remedial action has been taken prior to the inspection.
    6. Authorize the promotion of the current product within the version control software to the official current version.
- Reader: The reader will present the implementation at the inspection. The reader shall be one of the authors. The reader will lead the other inspectors through the implementation, presenting the material in a logical progression.
- Recorder: The recorder will record the problems found, their level of severity, and any remedial actions called for. The moderator may also function as the recorder.
- Author: The author is a person who has developed the code. There may be more than one author. The purpose of the author during the inspection is to answer questions and help with the inspection.
- Inspector: The purpose of the inspector is to identify defects, and offer solutions.

Preparation

The moderator will identify a subset of classes within a subsystem or package for review. The number of classes selected should be based on the ability of the inspectors to comfortably review them within the allotted timeframe. Thus, fewer classes will be reviewed as their size and complexity increases. The following guidelines should be used by moderators in identifying classes to be reviewed:

- Classes that have been identified by a metrics tool as having a high level of complexity.
- Classes that implement a significant business rule(s) or process.
- Classes that inherit from a framework base class, requiring extensive use of overridden and inherited methods.
- Classes that the author(s) have requested to be reviewed.
- Classes that have not been reviewed previously.
- Classes that have a relatively low level of unit test coverage, a possible indicator of complexity, or paths that are difficult to exercise.
- Classes that have a significant number of versions, a possible indicator of complexity.
- Classes that have been significantly reworked since their last review.
- Classes that have been identified as contributing to a bug identified during testing.

Inspectors will be assigned to review specific subsystems, packages, or classes by the moderator.

Pre-Inspection

The inspectors will be allowed 2 working days to review the classes prior to the walkthrough. Constructive comments should be prepared and brought to the review.

The author(s) should prepare hardcopies of the source code with line numbers for the review. Design material will be available online during the review.

Inspection

Each meeting should be kept to a maximum length of two hours. The moderator and the reader will guide the participants through the inspection. The recorder will record any issues brought up during the inspection. At the end of the meeting, a course of action must be decided:

- Accept the implementation as-is.
- Accept the implementation with minor revisions

- Require major revisions along with another inspection.
- Require a complete redo along with another inspection.

Remember that the purpose of the inspection is to find problems, not fix them. Also remember that the purpose of the inspection is to critique the implementation, not the authors.

Follow-up Inspection (Optional)

If the author(s) so desire(s), there shall be an optional follow-up inspection. During this inspection, the focus should be on verifying that appropriate corrective actions have been taken.

Checklists

Checkpoints for General Issues

- Does the code implement the design as reflected in the design model?
- Does the code follow the Programming Guidelines?
- Is the code self-documenting? Is it possible to understand the code from reading it?

Checkpoints for the Comments

- Are comments up to date?
- Are comments clear and correct?
- Are the comments easy to modify, if the code is changed?
- Do the comments focus on explaining why, and not how?
- Are all surprises, exceptional cases, and work-around errors commented?
- Is the purpose of each operation commented?
- Are other relevant facts about each operation commented?

Checkpoints for Source Code

- Does each operation have a name that describe what the operation does?
- Do the parameters have descriptive names?
- Is the normal path through each operation, clearly distinguishable from other exceptional paths?
- Is the operation too long, and can it be simplified by extracting related statements into private operations?

- Is the operation too long, and can it be simplified by reducing the number of decision points? A decision point is a statement where the code can take different paths, for example, if-, else-, and-, while-, and case-statements.
- Is nesting of loops minimized?
- Are the variables well named?
- Is the code straightforward, and does it avoid "clever" solutions?
- Is the code memory efficient?
- Does the code reuse existing data structures and utility classes?