CBOE Design Inspection Process

Introduction

This document specifies the process to be used for inspecting detailed code designs. The inspection is performed in an effort to find and fix problems as early in the development process as possible, thereby reducing overall development costs. The specific goals of the detailed design inspection are to:

- Ensure that the design meets the approved requirements.
- Validate all logic algorithms, data structures, and calls within each module.
- Verify that the detailed design is complete for each class
- Ensure traceability of the design to the approved requirements.
- Ensure that all required and/or applicable programming standards are followed.
- Ensure that detailed design meets requirements and is traceable to the high-level software system design.

Phases

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

Required Material

1. Use Case Sequence Diagrams. The use case sequence diagrams specify the behavior of the system and the design must provide the functionality specified in the sequence diagrams. Each phase may implement new sequence diagrams and must do so without breaking the functionality provided for sequence diagrams from previous phases. Phases may also provide new levels of support, e.g., providing persistence to what was an in-memory data base.
2. Class Diagrams: The class diagrams show the inheritance relationships of the various classes and the services (methods) that they provide.
3. Class Specifications: The class specifications give the full details of the classes.
4. 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.
5. Statechart Diagrams: The statechart diagrams are used to model the dynamic behavior of a reactive class, or the system as a whole. A reactive class is one whose behavior is best characterized by its response to events dispatched from outside its context. Statechart diagrams are not required for all classes.

It is expected that the required material will be reviewed prior to the inspection using Rational Rose. It is also expected that the inspection itself will be carried out with the aid of Rational Rose and a projection device which allows simultaneous viewing of the material by all participants.

Participants

Each design inspection requires a minimum of three people; while there are more than three roles, one person can perform multiple roles. 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 lead the inspections. The duties of the moderator will be to:
    1. Ensure that all inspectors are prepared prior to the inspection meeting. The inspector will notify the participants which use cases and classes will be inspected.
    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 follow up meeting.
    6. Authorize the promotion of the current product within the version control software to the official current version.
- Reader: The reader will present the design at the inspection. The reader shall be one of the authors. The reader will lead the other inspectors through the design, presenting the material in a logical progression.
- Recorder: The recorder will record the problems found, their severities, and any remedial actions called for. The moderator may also function as the recorder.
- Author: The author is a person who has developed the detailed design. There may be more than one author. The purpose of the author at the inspection is to answer questions and help with the inspection. The author may not be the moderator.
- Inspector: The purpose of the inspector is to identify defects, but not to provide solutions.

Preparation

The moderator will make sure that the participants have access to the materials before the inspection with sufficient time to review the materials. Reasonably, this should be at least

one day prior to the inspection. Each participant will review the materials prior to the inspection and prepare a list of any concerns.

Inspection

Each meeting should be kept to a maximum length of two hours. The moderator and the reader will guide the participants through the design. 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 design as is.
- Accept the design 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 design, not the authors.

Follow-up Meeting (Optional)

If the author(s) so desire(s) it, there shall be an optional follow-up meeting, also called a third-hour meeting. At this meeting, resolutions to open issues recorded in the inspection meeting may be obtained and solutions for defects identified during the inspection may be discussed.

Checklists

Checkpoints for Rose

- Are major components defined within their own interface package within the Analysis Model?
- Is there a corresponding package in the design model?
- Are the necessary framework/CORBA classes defined in the Design Model (Home, Operations, Structs, etc)?

Checkpoints for the Design Model

- Have you met the actor's demands?
- Do you need all the identified relationships between classes for the use-case realizations? If not, the relationships are redundant and you should remove them.
- Do you need all the identified attributes for the use-case realizations? If not, the attributes are redundant and you should remove them.

- Have you described the same function in more than one place? You should not have, for example, two classes with different names but with more or less the same structure, content, or behavior. Combine the two classes to avoid redundancy.
- Does the Introduction of the design model contain all the necessary information?
- Do the diagrams depicting the design model as a whole illustrate the contents of the model?
- Do the diagrams depicting the design model as a whole give a good picture of the design model?
- Do the role names of the relationships describe the associated class's relation to the associating class, and are the multiplicities correct?

Checkpoints for Use Case Realization

- Have you found all the required objects and classes for each use-case realization?
- Have you distributed all use-case behavior among the participating objects, and adequately described the behavior in sequence diagrams? Have you taken care of all exceptional cases in the use cases, such as, error handling?
- Are all objects and operations used in the use-case realization part of a sequence diagram? Do the sequence diagrams define all the objects and operations used in the use-case realization? Is there any object in the sequence diagrams that does not participate in the use case?
- Do the sequence diagrams correctly describe the use case's flow of events?
- Does the use-case realization make requirements on the use case that are not visible in the design model but that must be taken care of in implementation? Is this documented in the Derived Requirements of the use-case realization?
- Have you defined all relationships between classes that are needed for the use-case realizations?
- Have you defined all attributes needed for the use cases?
- Does the use case flow naturally over the participating objects, or do some places need smoothing?
- Has the behavior been distributed to the right objects (See the section, "Distribute the Behavior of the Use Case to the Objects")?
- Have you chosen the right behavior structure for the use case? See Rational Objectory Process Modeling Guidelines,
- Use-Case Realization, "Distribution of Control in the Flows of Events".
- If there are several sequence diagrams for one use case, is it clear which diagram describes which flow of events?

Checkpoints for Classes

- Have you handled each object's complete lifecycle? Make sure that each object is created by a use-case realization, used by a use-case realization, and removed by a use-case realization. If an object is only used, but not created, then a use case is missing. If an object is never removed, the database might become very large.
- Does the class offer the operations required by other classes that will communicate with it (via messages)?
- Are all the relationships of the class related to potential changes in the rest of the classes?
- Are there any class properties that should be generalized, that is, moved to an ancestor?
- Are there specific requirements on the class in the requirement specification that have not been taken care of?
- Are the demands on the class (as reflected in the class description and by the objects in sequence diagrams) consistent with any state diagram that models the behavior of the class and its objects?
- Do the classes offer the behavior the use-case realizations and other classes require?
- Have you assigned operations for the messages of each object completely?
- Are all the identified attributes and relationships actually needed by the use-case realizations? Check each of them and remove them if they are redundant.
- Is the state description of the class and its objects' behavior correct?
- Have you defined parameters for each operation?
- Are the multiplicities of the relationships correct?
- Are the implementation specifications (if any) for an operation correct? Do they serve their purpose?
- Are the Special Requirements correct for each class?
- Does a class define any attributes, responsibilities, or operations that are not functionally coupled to the other attributes, responsibilities, or operations defined by that class? Split any such class into several separate classes.
- Does the same class represent different things? If so, consider splitting it.
- Does the same attribute represent different things? If so, consider splitting it.
- Are there too many parameters for an operation?
- Are the names of the classes descriptive?
- Are the names of responsibilities descriptive?
- Are the responsibilities of a class understandable for those who want to use them? Do they have a suitable description?
- Do the role names of the aggregations and associations adhere to the correct form, that is, do they describe the relationship between the associated class to the relating class?
- Are the names of the attributes descriptive?

- Are the names of the operations descriptive, and are the operations understandable to those who want to use them?