

# Using Guided Inspection to Validate UML Models

**Melissa L. Major and John D. McGregor**  
**Software Architects**

Guided inspection is an inspection/review technique that is “guided” by test cases. Inspections are used to provide a detailed examination of a design or program by a human, as opposed to a machine’s execution of a prototype or completed application. However, even Fagan-style inspection processes focus more on the form of the inspection process rather than the substance of the material being inspected. Standard inspection techniques also focus on examining what is in the inspection material rather than determining whether there is something that is missing from the model or code.

These standard inspections are often a top down reading of the code or a scan of a diagram. The top down approach makes the measurement of coverage straightforward but it is more difficult for the inspector to ensure that appropriate connections have been made between objects. The use of test cases means that the inspection process can address more than just the syntax of the diagram or code being reviewed. The test cases come from test plans that are already a required part of the software development process.

Techniques such as checklists have been used to summarize the results of an inspection and to ensure that the inspector does a thorough job. Guided inspection supplements the checklist with the testing concept of “coverage”. Coverage measures determine how much of the product being inspected has been examined. Test cases are selected from the test plan so that, for example, every use case is represented by at least one test case.

Studies have reported widely varying savings ratios for finding faults early in the development process as opposed to during the compilation or system test phases. For example, IBM reported that repairing a fault found at system test time may cost as much as 100 times the cost of repairing the same fault found during design. With this amount of margin even a technique that is relatively expensive can still result in time and cost savings.

## **The Testing Perspective Applied to Inspections**

Applying a testing approach to inspections provides several benefits. In this section we examine these benefits and provide guidelines for the inspection process that ensure these benefits are realized.

### **Objectivity**

For testing to be effective it must be conducted objectively. A person testing their own code is seldom sufficiently objective to achieve optimum results. If the person has made a wrong interpretation of the inputs to their process, that mistake will simply be carried forward into the test cases. A second person, or even an automated tool, will provide a different, although not always correct perspective.

Guideline #1 – *Select inspectors from outside the immediate development team.*

### **Traceability**

For testing materials to be maintainable it must be easy to map changes in the model to needed changes in the test scenarios. In an iterative development environment changes occur frequently to all parts of the project. Changes in requirements are reflected in changes to the use cases. Changes in class specifications should signal the need for regression testing of the effected parts of the model.

Guided inspection uses scenarios derived from the use case description as the primary test case description. A project should maintain a matrix that associates a package with all the use cases for which that package is needed. Then each time changes are made to a package, the affected use cases and scenarios are easily identified.

Guideline # 2 – *Maintain a mapping between use cases and the classes/packages that realize those use cases.*

## Testability

For testing to be possible, the model must be testable. This implies that the model is sufficiently specific to support the evaluation of test execution results. Domain models are general by design and there is a fine line between vague generality and sufficient detail to support testing.

Guideline # 3 – *Assign a team member to write test cases as the modeling proceeds. Have the “testing” domain expert review these. Feedback, into the modeling process, any information indicating places where the model is too vague.*

This is common advice that we give to process definers at all levels. There should be a validation activity for each development phase. Preparation for that activity should proceed in parallel with the development activity. This allows the act of preparation to actually help improve the product before the formal validation. Writing test cases is an excellent technique for providing continuous feedback during development.

## Coverage

For testing to provide us with confidence, we need to know how thoroughly the product under test has been examined. The general term for this type of metric is *coverage*. When we speak of “functional” testing, we mean that the coverage will be expressed in terms of the functional specification of the product under test. The metric is chosen to give some notion of completeness at the appropriate level.

For guided inspection there are two different possible bases for coverage: the class/state/activity diagrams and use case diagrams. The use case diagram is a good source of scenarios; however, we are more concerned that the domain model contains a complete set of concepts for the domain. These are represented in the class diagram and further clarified in the state and activity diagrams for each class.

Guideline # 4 – *Use copies of the model’s diagrams and mark off each element in a diagram as it is used in a test scenario.*

Developing a test scenario for each actor in the use case diagram is a minimal level of coverage. One scenario per primary use case is a stronger coverage criterion. Covering every primary use and then adding coverage for all “alternate courses of action” for use cases that are rated frequent and critical is an even stronger criterion. Once the set of scenarios are run through the model, the resulting coverage of the class diagram and state diagrams provides a check of the thoroughness with which the model has been inspected.

## Criteria for a Good Model

The Guided Inspection evaluation criteria used by models are described more completely in [1]. They are:

- correctness
- completeness
- consistency

**Correctness** is a measure of how accurately the model represents the information. Correctness of the model is really the aggregate of judgements from the individual test cases. Each test case includes a description of the results expected from executing the test case. This expected result is based on a source that is assumed to be (nearly) infallible, a “test oracle”. The oracle usually is a human expert whose personal knowledge is judged to be sufficiently reliable to be used as a standard. The tester judges the accuracy of the model’s representation of concepts relative to the results expected by the oracle.

A model is correct with respect to a test case if the result of the execution is the result that was expected. A model is correct if each of the test cases produces the expected results. The problem here is whether the

“expected” result really is the appropriate one. In the real world, we must assume that the oracle can be incorrect on occasion.

**Completeness** is a measure of whether a necessary, or at least useful, element is missing from the model. It is judged by determining if the entities in the model describe the information being modeled in sufficient detail for the goals of the current portion of the system being developed. This judgement is based on the model’s ability to represent the required situations and on the knowledge of experts. In an iterative incremental process, completeness is considered relative to how mature the current increment is expected to be. This criterion becomes more rigorous as the increment matures over successive iterations.

One factor directly affecting the effectiveness of this criterion is the quality of the test coverage. The model is judged complete if the results of executing the test cases can be adequately represented using only the contents of the model. For example, a sequence diagram might be constructed to represent a scenario. All of the objects needed for the sequence diagram (SD) must come from classes in the class diagram or it will be judged incomplete. However, if only a few test cases are run, missing classes may escape detection. In most cases, this type of testing is sufficiently high level that coverage of 100% is achievable and desirable.

**Consistency** is a measure of whether there are contradictions among the various diagrams within the model and between models produced during various phases. This may be partially judged by considering whether the relationships among the entities in the model allow a concept to be represented in more than one way. For example, each name should be unique. In an incremental approach the consistency is judged locally until this increment is integrated with the larger system. The integration process must ensure that the new piece does not introduce inconsistencies into the integrated model.

Consistency checking can determine whether there are any contradictions or conflicts present either internal to a single diagram or between two diagrams. For example, one diagram, perhaps a sequence diagram, might require a relationship between two classes while another diagram, such as the class diagram, shows none. Inconsistencies will often initially appear as incorrect results in the context of one of the two diagrams and correct results in the other. Inconsistencies are identified by careful evaluation of the results of a simulated execution.

## A Basic Process

### Roles

There are several roles in this process. Several roles may be assigned to a single person; however, to ensure objectivity there should be a clear distinction between the producers of the model under test (MUT) and the testers/inspectors.

**Test oracle** - These personnel are the source of truth (or at least expected test results). They define the expected system response for a specific input scenario. These will usually be either domain experts or system engineers.

**Test case writer** - These personnel perform the analysis necessary to select test cases. They also record the expected result for each test case as defined by the Oracle. These people may be developers who did not create the model or system test personnel.

**Symbolic executioner** - These personnel provide the actual system response as defined to this point in the software development process. These will typically be members of the team developing the MUT since they understand the operation of the individual elements of the model.

**Moderator** - The Moderator controls the session and advances the discussion through the scenario.

**Recorder** - This person makes modifications to the reference models as the team agrees upon changes. The Recorder also makes certain that these changes are taken into consideration in the latter parts of the scenario. The Recorder also maintains a list of issues to record questions that are not resolved during the testing session.

**Drawer** - This person constructs the SD as the scenario is executed. He/ she concentrates on capturing all of the appropriate details such as returns from messages and state changes. They may also annotate the SD with information between the message arrow and the return arrow.

## Steps

The model testing process is tightly coupled with the model development process [2]. We have found it useful to iterate within the modeling process by periodically switching from the modeling activity to the testing activity. This provides quick feedback and often provides new information to be modeled.

The basic steps are the same as for any testing process:

- **Analyze** - Much of the testing analysis has been done if the use case descriptions contain sufficient information to allow them to be prioritized. We use a weighted frequency profile to prioritize use cases for testing. The weight is based on how critical the use is to the success of the system.
- **Construct** - Write scenarios from the use cases. Each scenario must be made more specific by providing exact values for attributes. These values are selected by first establishing equivalence classes of values. Equivalence classes of values are all values that will provide the same behavior in a given context. For example, {0,1,2,3} all produce the same response from the statement  $x > -1$  and  $x < 4$ . Many of the test parameters will be objects. Equivalence class translates to “objects that are in the same state no matter how they got there.” Different use cases will have different numbers of test cases. We select from some states more frequently than others due to their participation in high priority use cases.
- **Execute and Evaluate** – The inspection process combines the application of a checklist with the execution of test cases. The test session involves role-playing in which the modelers and developers step through the model. The test session is a group meeting since no individual developer will understand all of the classes in the model and few models contain all the relevant information. The moderator selects one of the scenarios and triggers the use. The developer/ owner of the class that begins the scenario by describing the action taken by his/her object and describes its interaction with other objects. For each interaction, the owner of the class receiving the message describes their interaction with other objects and execution proceeds along each of these links.

## Summary of Our Experience

The Guided Inspection technique has been used in a variety of forms on a number of projects that differ in size, complexity and domain. The technique has been used in the usual analysis and design contexts where a development organization applied the technique to each model produced by an increment team. It has also been used in limited engagements where a domain model or an architectural model was the only artifact being evaluated. Our experience and that of knowledgeable clients is that this technique has greater defect finding power than other widely used inspection techniques. The technique does require more effort (the construction of test cases) than other inspection techniques but it is effort that would be expended anyway.

The use of test cases brings a logical continuity to the inspection. Each step in the test case is a logical consequence (rather than a syntactic necessity) of the previous steps. This guides the inspectors through the material to be inspected in a path that allows them to judge the semantic validity of the model in

addition to evaluating its syntactic correctness. The result is that the defects that are found have the potential of greater impact on the system than the syntactic bugs found in a sequential search.

## **Conclusion**

We have presented an overview of guided inspection. This quality technique provides a means for examining models and code in a semantically meaningful way rather than examining disjoint pieces of syntax. Detecting defects in the early analysis and design models makes a major contribution to the quality of the application and to an on-time, on-budget delivery. Our presentation at the workshop will elaborate on the steps in the basic process and illustrate the models being inspected.

## **References**

1. John D. McGregor. *The Fifty Foot Look at Analysis and Design Models*, **Journal of Object-Oriented Programming**, July/August 1998.
2. John D. McGregor. *Testing Models, The Requirements Model*, **Journal of Object-Oriented Programming**, June 1998.

# A Process Definition for Guided Inspection

**John D. McGregor**  
**Melissa L. Major**  
**Software Architects**

*Goal:* To identify defects in artifacts created during the analysis and design phases of software construction.

## *Steps in the Process*

1. Define the scope of the Guided Inspection
2. Identify the basis model(s) from which the material being inspected was created
3. Assemble the GI team
4. Define a sampling plan and coverage criteria
5. Create test cases from the bases
6. Apply the checklist and tests to the material
7. Gather and analyze test results
8. Report and feedback

## *Detailed Step Descriptions*

### Define the scope of the Guided Inspection

**Inputs:**

The project's position in the life cycle.  
The materials produced by the project (UML models, plans, use cases).

**Outputs:**

A specific set of diagrams and documents that will be the basis for the evaluation.

**Method:**

Define the scope of the GI to be the set of deliverables from a phase of the development process. Use the development process information to identify the deliverables that will be produced by the phase of interest.

**Example:**

The project has just completed the domain analysis phase. The development process defines the deliverable from this phase as a UML model containing domain-level use cases, static information such as class diagrams and dynamic information such as sequence and state diagrams. The GI will evaluate this model.

Identify the basis model(s) from which the material being inspected was created

**Inputs:**

The scope of the GI.  
The project's position in the life cycle.

**Outputs:**

The material from which the test cases will be constructed (The Model Under Test - MUT)

**Method:**

Review the development process description to determine the inputs to the current phase. The basis model(s) should be listed as inputs to the current phase.

**Example:**

The inputs to the domain analysis phase is the "knowledge of experts familiar with the domain". These mental models are the basis models for this GI.

### Assemble the GI team

**Inputs:**

The scope of the GI.  
Available personnel.

**Outputs:**

A set of participants and their roles.

**Method:**

Assign persons to fill one of three categories of roles: Administrative, Participant in creating the model to be tested, Objective observer of the model to be tested. Choose the objective observers from the customers of the model to be tested and the participants in the creation of the basis model.

**Example:**

Since the model to be tested is a domain analysis model and the basis model is the mental models of the domain experts, the objective observers can be selected from other domain experts and/or from application analysts. The creation participants are members of the domain modeling team. The administrative personnel can perhaps come from other interested parties or an office that provides support for the conduct of GIs.

## Define a sampling plan and coverage criteria

### Inputs:

The project's quality plan.

### Outputs:

A plan for how test cases will be selected.

A description of what parts of the MUT will be covered.

### Method:

Identify important elements of this MUT. Estimate the required effort to involve all of these in the GI. If there are too many to cover, use information such as the RISK section of the use cases or the judgement of experts to prioritize the elements.

### Example:

In a domain model there are static and dynamic models as well as use cases. At least one test case should be created for each use case. There should be sufficient test cases to take every "major" entity through all of its visible states.

## Create test cases from the bases

### Inputs:

The sampling plan.

MUT

### Outputs:

A set of test cases.

### Method:

Obtain a scenario from the basis model. Determine the pre-conditions and inputs that are required to place the system in the correct state and to begin the test. Present the scenario to the "oracle" to determine the results expected from the test scenario. Complete a test case description for each test case.

### Example:

A different domain expert than the one who supported the model creation would be asked to supply scenarios that correspond to uses of the system. The experts also provide what they would consider an acceptable response.

## Apply the checklist and tests to the material

### Inputs:

Set of test cases.

Checklist for the type of model being inspected.

MUT

### Outputs:

Set of test results.

Completed checklist.

### Method:

Apply the test cases to the MUT using the most specific technique available. For UML models in a static environment, such as Rational Rose, an interactive simulation session in which the Creators play the roles of the model elements is the best approach. If the MUT is represented by an executable prototype then the test cases are mapped onto this system and executed. After the model has been thoroughly examined, complete the checklist.

### Example:

The domain analysis model is a static UML model. A simulation session is conducted with the Observers feeding test cases to the Creators. The Creators provide details of how the test scenario would be processed through the model. Sequence diagrams are used to document the execution of each test case. Use agreed upon symbols or colors to mark each element that is touched by a test case.

## Gather and analyze test results & coverage

### Inputs:

Test results in the form of sequence diagrams and pass/fail decisions.

The marked-up model.

### Outputs:

Statistics on percentage pass/fail.

Categorization of the results.

Defect catalogs and defect reports.

A judgement of the quality of the MUT and the tests.

#### Method:

Begin by counting the number of test cases that passed and how many have failed. Compare this ratio to other GIs that have been conducted in the organization. Compute the percentage of each type of element that has been used in executing the test cases. Use the marked-up model as the source of this data. Update the defect inventory with information about the failures from this test session.

Categorize the failed test cases. This can often be combined with the previous two tasks by marking paper copies of the model. Follow the sequence diagram for each failed test case and mark each message, class and attribute touched by a failed test case.

#### Example:

For the domain analysis model we should be able to report that every use case was the source of at least one test case, that every class in the class diagram was used at least once. Typically on the first pass, some significant states will be missed. This should be noted in the coverage analysis.

## Report and feedback

#### Inputs:

- Test results.
- Coverage information.

#### Outputs:

- Information on what new tests should be created.
- Test report.

#### Method:

Follow the standard format for a test report in your organization to document the test results and the analyses of those results. If the stated coverage goals are met then the process is complete. If not, use that report to return to step 5 and proceed through the steps to improve the coverage level.

#### Example:

For the domain analysis tests, some elements were found to be missing from the model. The failing tests might be executed again after the model has been modified.

## *Roles in the Process*

### Administrator

The administrative tasks include running the GI sessions, collecting and disseminating the results, and aggregating metrics to measure the quality of the review. In our example, personnel from a central office could do the administrative work.

### Creator

The persons who created the MUT are the creators. Depending upon the form that the model takes, these people may “execute” the symbolic model on the test cases or they may assist in translating the test cases into a form that can be executed with whatever representation of the model is available. In our example the modelers who created the domain model would be the “creators”.

### Observer

Persons in this role create the test cases that are used in the GI. In our example they would be domain experts and preferably experts who were not the source of the information used to create the model initially.