

STEP 1: Decompose the Application Functionality

STEP 6: Measure Test Coverage and Identify Gaps

STEP 2: Identify Core Features

STEP 3: Identify Crosscutting Concerns

# MIND T

## Using a Requirements Composition

STEP 6: Measure Test Coverage and Identify Gaps

STEP 2: Identify Core Features

STEP 1: Decompose the Application Functionality

STEP 5: Establish Traceability Between Requirements and Regression Tests

STEP 3: Identify Crosscutting Concerns

STEP 3: Identify Crosscutting Concerns

STEP 4: Develop a Requirements Composition Table

STEP 5: Establish Traceability Between Requirements and Regression Tests

STEP 1: Decompose the Application Functionality

STEP 6: Measure Test Coverage and Identify Gaps

# THE GAP

on Table to Assess Test Coverage

BY YURI CHERNAK

STEP 6: Measure Test Coverage and Identify Gaps

STEP 4: Develop a Requirements Composition Table

STEP 5: Establish Traceability Between Requirements and Regression Tests

**O**n critical projects, testers are commonly required to design, maintain, and execute a regression test suite to certify an application before it is deployed in production. In this case, we are concerned with the test suite completeness, as our final application certification is only as good as the regression suite coverage. To be used as a frame of reference, a conventional test-coverage measurement technique requires complete software requirements. We then establish traceability between requirements and related tests that allows us to measure test coverage and identify coverage gaps.

However, testers in the field commonly deal with incomplete and even missing software requirements. On such projects, they do not have sufficient visibility into the application's test coverage. This article provides a solution that I have found to be effective for this problem on many critical projects. The solution, a technique called requirements composition table (RCT), allows testers to assess the regression test suite completeness and identify test coverage gaps. Once gaps are identified, testers can better decide how to evolve the regression test suite to improve the application's test coverage.

The RCT is based on the insight that we do not need descriptions of software requirements in order to measure regression test coverage and identify coverage gaps; just having their inventory and structure can be sufficient. Identifying coverage gaps using this technique involves six steps:

1. Decompose the application functionality.
2. Identify core features.
3. Identify crosscutting concerns.
4. Develop a requirements composition table.
5. Establish traceability between requirements and regression tests.
6. Measure test coverage and identify gaps.

I illustrate the technique using an imaginary hotel management system (HMS). The following are terms used in its workflow:

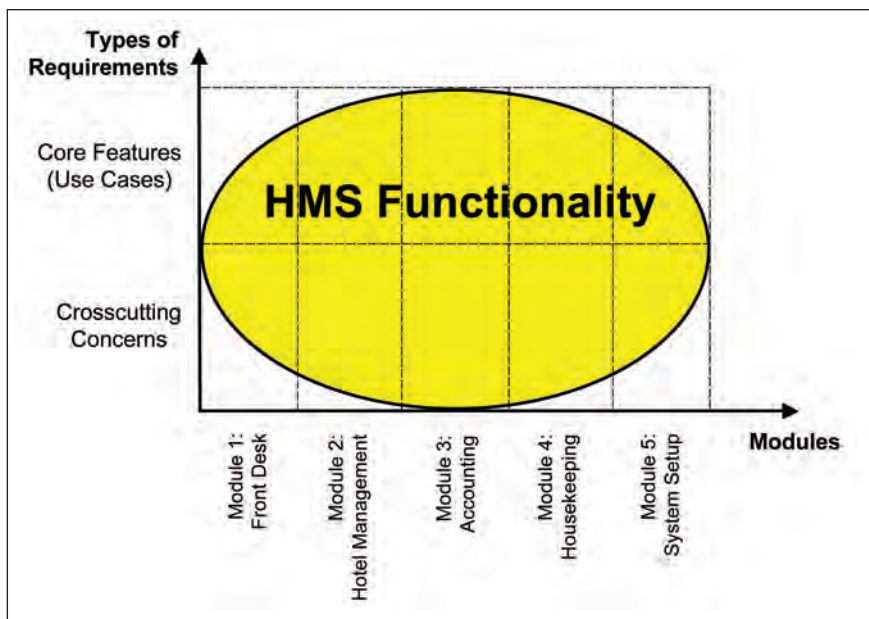


Figure 1: Application functional decomposition

- Core feature—This is a unit of the application functionality that, being executed, allows the user to achieve a particular business goal (result). For example, functionality captured by use cases can be qualified as core features.
- Crosscutting concern—This term comes from the aspect-oriented software development methodology (see the StickyNotes for more information). It refers to a specific category of software requirements that are scattered across the application and tangled with core features.
- 01. Front Desk—used by front-desk clerks for creating and managing reservations, handling financial charges, etc.
- 02. Hotel Management—used by hotel management for managing rate plans, etc.
- 03. Accounting—used by accounting personnel for creating and reviewing financial statements, conducting cashier audits, maintaining credit card limits, etc.
- 04. Housekeeping—used by housekeeping staff for maintaining the cleaning and occupancy statuses of rooms, assigning rooms to attendants, etc.
- 05. System Setup—used by hotel management for setting up hotel property defaults and user profiles, managing tax codes and charge codes, etc.

### STEP 1: Decompose the Application Functionality

We begin our analysis by decomposing the application functionality into a high-level structure of functional requirements from two perspectives: application modules and types of functional requirements.

Our HMS application can have the following modules:

- 01. Front Desk—used by front-desk clerks for creating and managing reservations, handling financial charges, etc.
- 02. Hotel Management—used by hotel management for managing

Each module still presents a complex chunk of functionality, so we continue decomposition from the second perspective—types of requirements—where we want to separate software requirements by different categories (i.e., concerns). Business applications implement core features that allow the user to achieve concrete business goals. They can be presented from the end-user perspective and specified by use cases. In addition, business applications are implemented based on some generic engineering principles related to data-entry validations, user role validations, concurrency control, handling front-end connectivity, data exchange with other systems, etc.

## "01. Front Desk" Module - Test Coverage Analysis

Testing Concerns	UC.01.01. Create Guest Reservation	UC.01.02. Update Guest Reservation	UC.01.03. Cancel Guest Reservation	UC.01.04. Check-In Guest	UC.01.05. Check-Out Guest	UC.01.06. Post Charges to Guest's Folio	UC.01.07. View, Update Folio Charges	UC.01.08. Create Message for Guest	UC.01.09. View, Cancel Message	UC.01.10. Add Travel Agency Commissions	UC.01.11. View, Update Travel Agency Commissions	UC.01.12. Manage Rooming List	Coverage by Concern Types
Core Functionality	1	1	1	1	1	1	1	1	1	1	1	1	75%
GUI Features	1	1	1	1	1	1	1	1	1	1	1	1	42%
<b>Crosscutting Concerns</b>													
UR - User Roles	1	1	1	1	1	1	1	1	1	1	1	1	25%
ST - Status	0	1	1	1	1	1	1	1	1	1	1	1	18%
FV - Field Validation	1	1	0	1	0	1	1	0	0	1	1	1	25%
DD - Data Dependency	1	1	0	1	0	1	1	0	0	0	0	0	0%
CC - Concurrency	1	1	0	1	0	0	0	0	0	0	0	0	0%
CN - Connectivity	1	1	1	1	1	0	0	0	0	0	0	0	60%
SI - System Interface	1	1	1	1	1	0	0	0	0	0	0	0	100%
<b>Test Coverage by Use Cases:</b>	75%	22%	50%	56%	83%	17%	0%	25%	0%	80%	40%	0%	37%

0 - means not applicable concern  
 1 - means applicable concern  
 Yellow-colored Cell - means missing tests (gap)

Table 1: Requirements composition table

We frequently find these concerns scattered across the application and tangled with core features. Hence, defining these concerns as a separate category of requirements (i.e., crosscutting concerns) can help us better structure requirements and achieve better traceability to functional tests. As a result of this step, we decomposed the application functionality by modules and types of requirements as shown in figure 1.

### STEP 2: Identify Core Features

We already noted that core features are commonly specified by use cases that achieve particular business goals for the end-user. To identify use cases, we take one module at a time and compile a list of goals that the user can accomplish when working with this module. Below is the list of use cases that we identified for the 01. Front Desk module:

- UC.01.01 Create Guest Reservation—has a goal to create a new guest reservation
- UC.01.02 Update Guest Reser-

vation—has a goal to modify an existing reservation

- UC.01.03 Cancel Guest Reservation—has a goal to cancel an existing reservation
- UC.01.04 Check In Guest—has a goal to check in a guest
- UC.01.05 Check Out Guest—has a goal to check out a guest
- UC.01.06 Post Charges to Guest's Folio—has a goal to post new charges to a guest's folio
- UC.01.07 View, Update Folio Charges—has a goal to view or update the folio charges
- UC.01.08 Create Message for Guest—has a goal to create messages for a guest
- UC.01.09 View, Cancel Message for Guest—has a goal to view or cancel existing messages
- UC.01.10 Add Travel Agency Commissions—has a goal to add commissions for a travel agency when a reservation was created via the travel agency
- UC.01.11 View, Update Travel

Agency Commissions—has a goal to view or update existing travel agency commissions

- UC.01.12 Manage Rooming List—has a goal to create, view, or update a rooming list allocated to a group reservation

Use cases are not supposed to capture all details of system functionality. Many other details will be captured by crosscutting concerns that we discuss in step 3.

### STEP 3: Identify Crosscutting Concerns

In general, crosscutting concerns can represent functional and non-functional requirements. However, in this article we focus our discussion only on functional crosscutting concerns. Identification of crosscutting concerns means analyzing the application functionality to find some categories of requirements that are scattered across the application and tangled with the context of various use cases. We perform this step based

on the following heuristics: *Can this requirement category impact the context of a use case?* and *Is this requirement category sufficiently scattered, affecting more than one to two use cases?*

As a result, we identified the following list of crosscutting concerns:

- **User Roles**—In the HMS application, different users can have different privileges (user roles). Depending on her user role, a given user can or cannot execute some use cases.
- **Status**—Some entities in the HMS application—e.g., reservation, company, bank, or travel group—have a certain lifecycle composed of different statuses. Some use cases can or cannot be executed depending on a status of the related entity. Hence, the system should validate the entity status before allowing the user to execute such use cases.
- **Field Validation**—This concern relates to validating individual

data-entry fields. If validation fails, a use case’s flow is interrupted and the system responds with an error message asking the user to correct data.

- **Data Dependency**—This concern relates to validating a combination of two or more fields. Even if individual fields have passed their field validation, their combination might fail the data dependency validation. Again, if validation fails, the use case scenario is interrupted.
- **Concurrency Control**—This concern relates to validating concurrent manipulation with the same data at the same time by more than one user. Failure of concurrency validation will result in the interruption of a use case scenario.
- **Connectivity**—Most business applications have complex architectures composed of different components implementing front end,

middleware, and back end. The connectivity among components is not always stable, which means that the front-end component can go into a disconnected state. The HMS system should implement generic functionality, first to validate whether the front end stays connected while a user completes a use case and, second, to define the alternative behavior in case the front end goes into a disconnected state.

- **System Interface**—This concern relates to sending data to and receiving data from other systems, which is a part of many use cases—for example, a new reservation is sent from the HMS system to the central reservation system, or credit card charges are sent to a credit card vendor system.

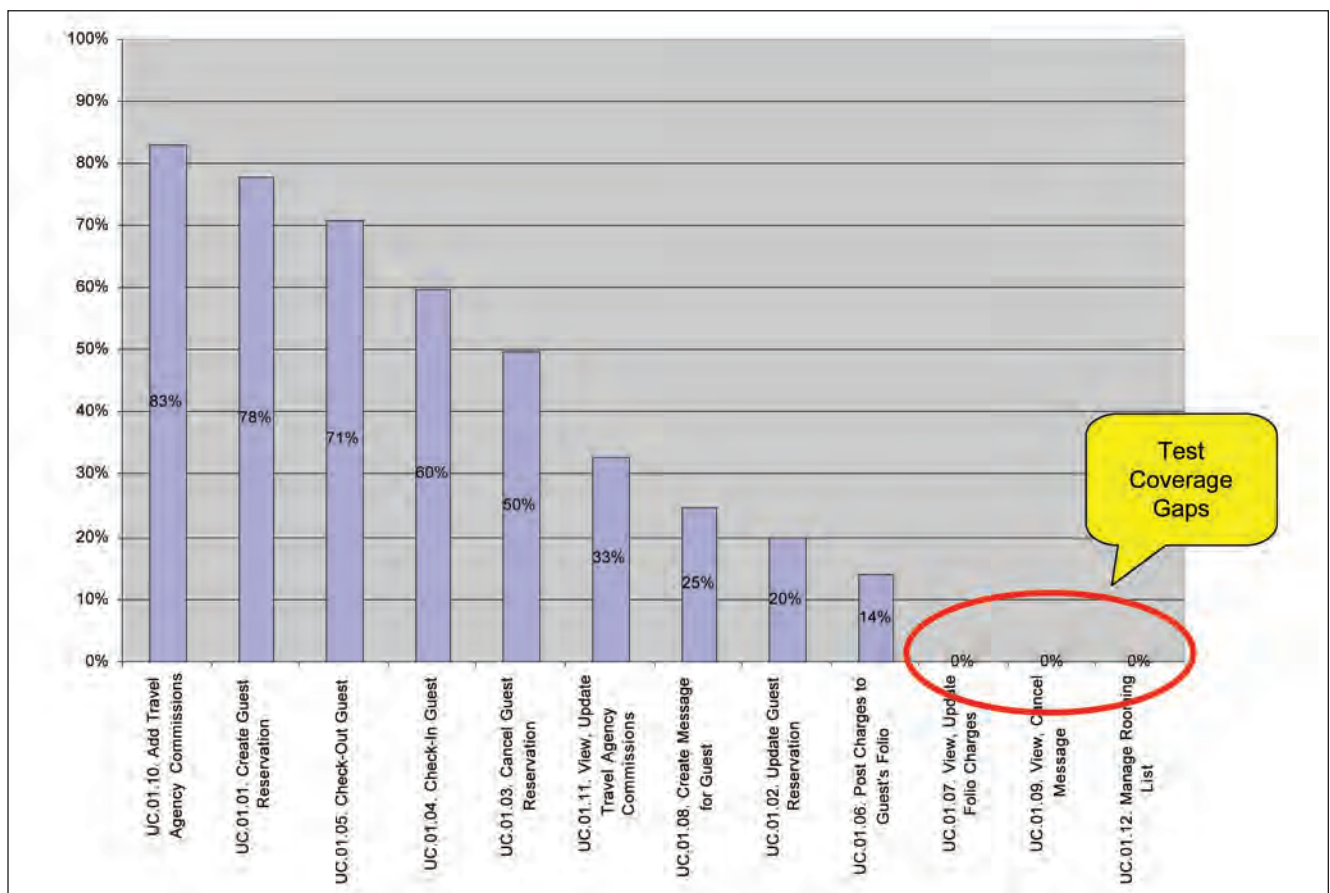


Figure 2: Test coverage by use cases

## STEP 4: Develop a Requirements Composition Table

In the previous steps we identified the inventory of core features and crosscutting concerns. We also know that crosscutting concerns are scattered across the application and affect core features. Therefore, the purpose of this step is to analyze the composition of requirements and specify their structure in an RCT, as shown in table 1. This table will be used as a frame of reference to establish traceability in step 5 and to assess test coverage in step 6.

Our HMS application consists of five modules, where each module includes a number of use cases. Depending on the module complexity, we can create a separate requirements composition table for each module. Table 1 shows an example created for the 01. Front Desk module. First, we populate columns in this table representing use cases, where each use case provides a context for analyzing functional features. Second, we compile a generic list of concerns composed of such requirements categories as core functionality, GUI features, and the identified crosscutting concerns. We then populate rows in the table with the list categories. As a result, we create a complete inventory of requirements that we will analyze to compose a structure of requirements.

Composing the requirements structure means that for each use case context we indicate which concerns are a part of or have impact on this context. We take one use case at a time and, going through the list of concerns, we make a decision about which concerns from the generic list should be applicable to that use case context. When we make a decision that a given concern is applicable, we indicate it in the corresponding cell in the table by entering 1; otherwise, we enter 0. We begin the requirements composition analysis with the first item in the list—Core Functionality. By default, each use case has core functionality, so we enter 1 in all cells related to this category.

Next in the list is the GUI Features category. Because each use case implements GUI, we enter 1 in all cells rep-

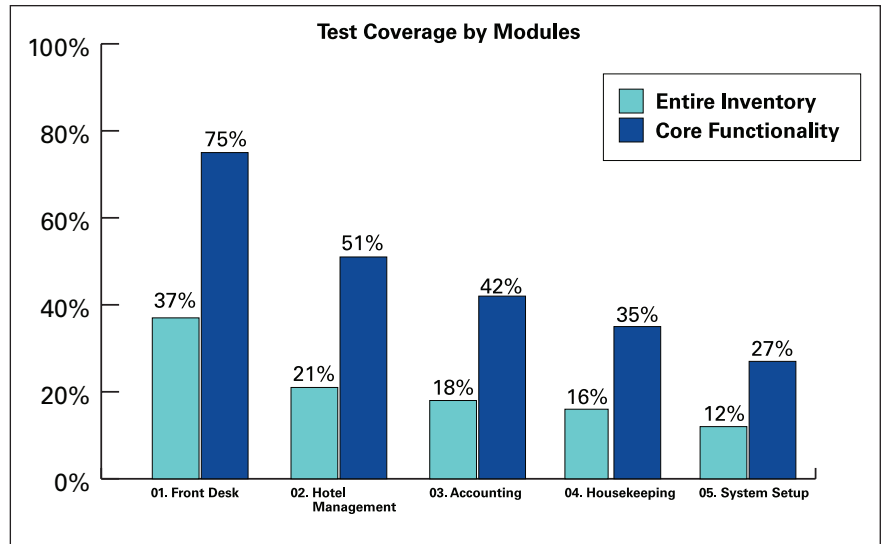


Figure 3: Summary of assessment results

resenting this category. We then continue this analysis for all crosscutting concerns in the list, making a decision about which of them affects a given use case context. Most of the time we make such decisions based on our common sense and prior experience. For example, we want to give the privilege to create a guest reservation only to some user roles. Hence, we make a decision that the User Roles concern affects the context of the UC.01.01 use case. Further, while creating a new reservation, the user should enter reservation data. Therefore, the system should implement field validation that can be invoked either when the user leaves a given data-entry field or when the user saves the entire reservation. This means that the Field Validation concern should be marked as affecting the UC.01.01 use case.

Another example is analysis of the Connectivity concern. One important requirement for the HMS application is that when the front end is disconnected, the application should inform the user about the disconnected state and then provide the user with some limited functionality. In particular, the front-desk clerk should be able to process guest reservations. This means that the Connectivity concern also affects the context of the UC.01.01 use case. We continue analyzing the first use case for all other crosscutting concerns. Once we have completed this analysis for all use cases, the composition table has all cells popu-

lated with either 1 or 0, thus representing a complete structure of functional requirements.

With step 4 we completed the first module's RCT that is now ready to be used for establishing traceability and measuring test coverage. Even though creating an RCT takes a few steps, in practice it doesn't take much time. Commonly, just a one-hour session with either a business analyst or developer who knows well the functionality of a given module is sufficient to complete the RCT for that module.

## STEP 5: Establish Traceability Between Requirements and Regression Tests

At this point in our analysis we already know the inventory and structure of functional requirements. We also know the inventory of the existing regression tests that we need to allocate to their related requirements. Once this step is completed, we will be able to measure test coverage and identify coverage gaps in step 6.

Commonly, regression tests are stored either in a test management tool, for example Mercury Quality Center (MQC), or in Excel or Word documents. If you use a test management tool, you first need to create in the tool a requirements repository following the same requirements structure as we created in the RCT. Alternatively, if you manage your

tests in Excel or Word documents, you can create the same requirements structure in an Excel file.

Once the requirements repository is created, we then use it as a frame of reference to establish traceability from the existing tests to related requirements. To perform this step, we take one regression test at a time and, based on our understanding of the test purpose, we make a decision about which software requirement this test is intended to validate. Then we link this requirement to the test. Depending on your test design, it is possible that a given test can be traced to more than one concern. We continue this activity until all regression tests are traced to their related requirements. After we have established traceability between regression tests and requirement, we can see exactly which concerns are covered and not covered by the regression test suite.

## STEP 6: Measure Test Coverage and Identify Gaps

At this point, I need to clarify that assessment of test coverage is not the same as evaluation of test-design completeness for individual requirements. For example, you can create tests for all application features and achieve complete test coverage, yet some of the individual requirements still can lack necessary test cases. Hence, this RCT technique is primarily intended to support our analysis from three perspectives:

1. Identifying test coverage gaps
2. Providing testers with visibility into which requirements have more and which ones have less test coverage
3. Helping testers understand which types of concerns they commonly miss in test designs

Based on the results of this analysis, testers can make better decisions about how to evolve their regression suite.

From step 5 we already know which requirements are not covered by tests. We again use the RCT to measure test coverage and we mark the cells that represent not covered requirements with a different color (yellow, as in table 1). Then we can derive coverage measure-

ments from two perspectives: test coverage for each use case context and test coverage for each concern type (across all use cases).

We measure test coverage for each use case context as follows:

- Calculate the sum of all applicable concerns for a given use case context. This number represents 100 percent of concerns to be covered by tests.
- Subtract from the sum the number of colored cells—i.e., concerns not covered by tests.
- Calculate the ratio of covered concerns to the total number of concerns and present it as a percentage of requirements covered by tests.

For example, the first use case context UC.1. Create Guest Reservation has a total of eight concerns. Two concerns, Data Dependency and Concurrency, are not covered by tests (see table 1); thus, the number of covered concerns is  $8 - 2 = 6$ , which is 75 percent. We continue this calculation for all other use case contexts to derive the test coverage measurements. In doing so, we take into account all applicable concerns identified in the RCT. Hence, our test coverage measurements represent the coverage of the entire inventory of requirements. After we calculate coverage for individual use case contexts, we can derive the average number for the entire module, which is 37 percent for the 01. Front Desk module. To better analyze and communicate the assessment results, we can present them as a bar chart, as shown in figure 2, from which we can easily see which use cases are better covered by the existing tests and which ones are not covered at all and present test coverage gaps. In our example the gaps are:

- UC.01.07. View, Update Folio Charges
- UC.01.09. View, Cancel Message
- UC.01.12. Manage Rooming List

Test coverage from the perspective of concern types is calculated in the same way. As we can see in table 1, the Data Dependency and Concurrency concerns are not covered by tests.

Finally, when we execute regression

testing we always have limited time. This means that a test team should decide which of the software requirements are most important to validate and agree on the regression test scope. Commonly, validating core functionality is a minimal requirement for regression test coverage. In addition, testers might decide that some of the identified crosscutting concerns should be included in the regression test scope. Thus, when assessing test coverage, we can present the results as a range between covering only the core functionality and covering a complete inventory of requirements, including all crosscutting concerns. Then it will be up to the test team to decide how much regression test coverage is really necessary for their application. For the 01. Front Desk module, the RCT shows the range of the test coverage— $37\% \div 75\%$ , where 37% is the coverage related to the entire inventory of requirements and 75% is the coverage of the core functionality only. When we have completed the assessment for all modules of the HMS application, the summary of results can be presented as shown in figure 3.

A requirements composition table discussed in this article is a requirements analysis technique that proved to be effective on many projects where testers had little or no requirements. In this paper I illustrated how the RCT can be used for assessing regression test coverage. The other RCT benefits for testers include effective knowledge transfer when they start a new engagement, better functional test planning, more effective exploratory testing, etc. In addition, this technique can help developers better analyze and evolve software requirements on traditional, use-case driven, and agile projects. **{end}**

### Sticky Notes

For more on the following topic go to [www.StickyMinds.com/bettersoftware](http://www.StickyMinds.com/bettersoftware).

- Aspect-oriented software development