https://www.ibm.com/developerworks/rational/library/04/r-3217/index.html Go JUN SEP APR
16 captures 09
15 May 2006 - 2 Apr 2019 2015 2016 2019 About this capture

developerWorks Premium An all-access pass to building your next great app!
Sign up

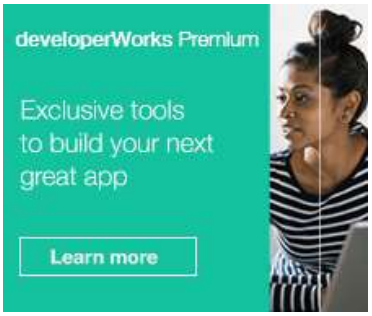
Traceability from Use Cases to Test Cases

The article illustrates a formal method of deriving functional test cases from use cases, including how to create a use case, derive all scenarios, and create reasonable test cases, as well as use IBM® Rational® RequisitePro for traceability from use cases to scenarios and test cases.

Director of Technology Solutions, The A Consulting Team, Inc.

04 May 2006 (First published 10 February 2006)
Also available in Chinese Japanese

Overview of the requirements types



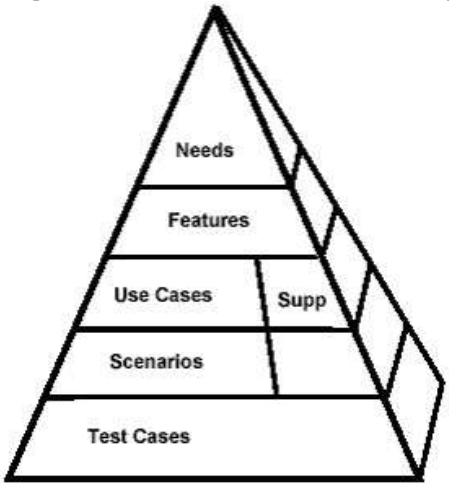
A requirement is defined as "a condition or capability to which a system must conform".

It can be:

- A capability needed by a customer or user to solve a problem or achieve an objective
- A capability that must be met or possessed by a system to satisfy a contract, standard, specification, regulation, or other formally imposed document
- A restriction imposed by a stakeholder

Figure 1 shows the requirements pyramid with the different levels of requirements.

Figure 1. The requirements pyramid



On the top level are stakeholder needs. Usually, a project contains five to fifteen of these high-level needs. On the lower levels are features, use cases, and supplementary specifications. On different levels of these requirements are different details. The lower the level, the more detailed the requirement is. For

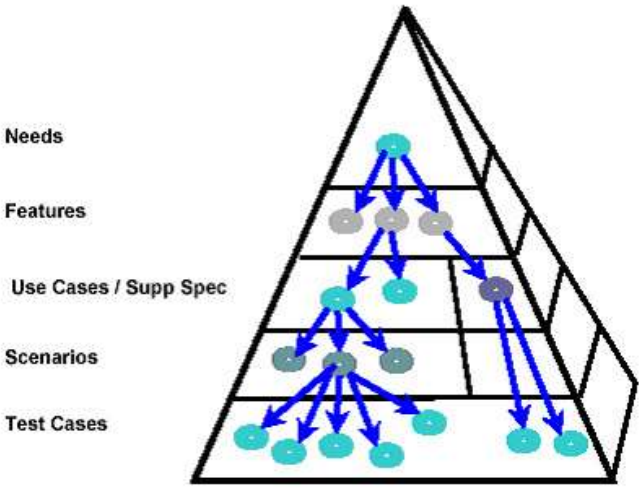
https://www.ibm.com/developerworks/rational/library/04/r-3217/index.html
Go
JUN SEP APR
09
2015 2016 2019
16 captures
15 May 2006 - 2 Apr 2019
About this capture

be even more specific: "System should use Oracle 9i database". The further down, the more detailed the requirement.

Traceability between requirements

Traceability is a technique that provides a relationship between different levels of requirements in the system. This technique helps you determine the origin of any requirement. Figure 2 illustrates how requirements are traced from the top level down. Every need usually maps to a couple of features, and then features map to use cases and supplementary requirements.

Figure 2. Traceability requirements pyramid



Use cases describe functional requirements, and supplementary specifications describe non-functional items. In addition, every use case maps to many scenarios. Mapping use cases to scenarios, then, is a one to many relationship. Scenarios map to test cases also in a one to many relationship. Between needs and features, on the other hand, there is many to many mapping.

Traceability plays several important roles:

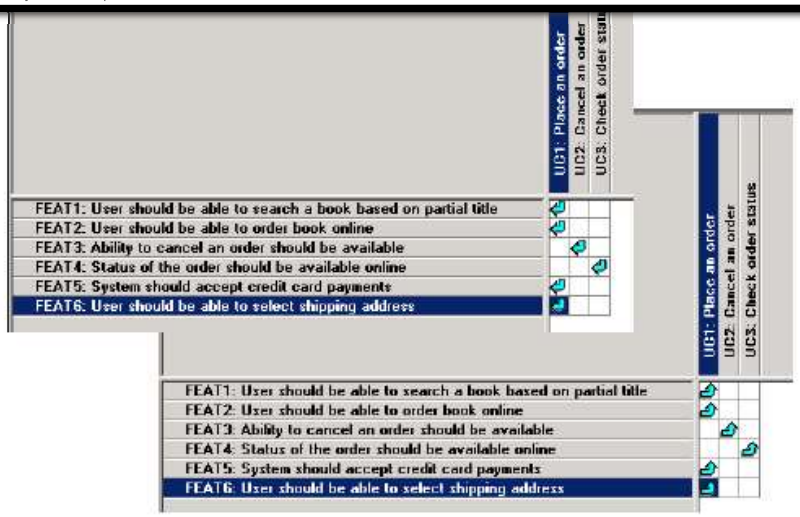
- Verify that an implementation fulfills all requirements: Everything that the customer requested was implemented
- Verify that the application does only what was requested: Don't implement something that the customer never asked for
- Help with change management: When some requirements change, we want to know which test cases should be redone to test this change

A traceability item is a project element that needs to be traced from another element. In terms of IBM Rational RequisitePro it's everything that is represented by an instance of the requirement type. Some examples of requirement types in RequisitePro are stakeholder needs, features, use cases, actors, and glossary terms.

In RequisitePro there is a convenient way of showing traceability in special views. Figure 3 shows an example of mapping features to use cases.

Figure 3. Traceability in RequisitePro

https://www.ibm.com/developerworks/rational/library/04/r-3217/index.html
Go
JUN SEP APR
16 captures
15 May 2006 - 2 Apr 2019
2015 2016 2019
About this capture



There is some question as to which direction the arrows should go: whether from lower level to higher level or from higher to lower level. Even the two examples in RequisitePro use different guidelines. The answer is that it doesn't matter, as long as you use them consistently across the project.

Actors and use cases

An actor is someone or something that interacts with the system. A use case is a description of a system in terms of a sequence of actions. It should yield an observable result or value for the actor. Following are some characteristics of use cases, which:

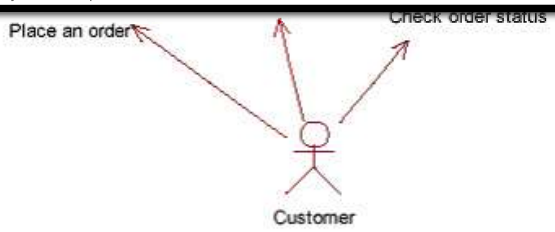
- Are initiated by an actor
- Model an interaction between an actor and the system
- Describe a sequence of actions
- Capture functional requirements
- Should provide some value to an actor
- Represent a complete and meaningful flow of events

The purpose of a use case is to facilitate agreement between developers, customers, and users about what the system should do. A use case becomes sort of a contract between developers and customers. It's also a basis for use-case realizations, which play a major role in design. In addition, you can produce sequence diagrams, collaboration diagrams, and class diagrams from use cases. Furthermore, you can derive user documentation from use cases. Use cases may also be useful in planning the technical content of iterations, and give system developers a better understanding of the purpose of the system. Finally, you can use them as an input for test cases.

Use case diagrams present relationships between actors and use cases. In this article we will use an online bookstore as an example of a project. Figure 4 shows a use case diagram for this project.

Figure 4. Use Case Diagram

https://www.ibm.com/developerworks/rational/library/04/r-3217/index.html
Go
JUN SEP APR
09
2015 2016 2019
16 captures
15 May 2006 - 2 Apr 2019
About this capture



The general format of a use case is:

1. Brief description
2. Flow of events
   - Basic flow
   - Alternative flow 1
   - Alternative flow 2
3. Special requirements
4. Preconditions
5. Post-conditions
6. Extension points
7. Context diagram
8. Activity diagram

The basic flow contains the most popular sequence of actions, the steps that happen when everything goes correctly. Alternative flows represent variations of the flow, including less usual cases and error conditions. A context diagram is a part of a use case diagram showing the relationships of this particular use case to actors and other use cases. An activity diagram is a flow chart that explains the use case. The context diagram and the activity diagram are not necessary, but help you visualize the use case and its position in the project.

In our Online Bookstore project, the basic flow of the use case place an order might look like this:

1. B1 User enters web site address in the browser.
   System displays login page.
2. B2 User enters an email address and a password.
   System confirms correct login, presents main page, and prompts for a search string.
3. B3 User enters search string – partial name of a book.
   System returns all books matching search criteria.
4. B4 User selects a book.
   System presents detailed information about a book.
5. B5 User adds the book to a shopping cart.
   Shopping cart contents is presented to the user.
6. B6 User selects "proceed to checkout" option.
   System asks for confirmation of a shipping address.
7. B7 User confirms shipping address.
   System presents shipping options.
8. B8 User selects shipping option.
   Systems asks which credit card will be used.

https://www.ibm.com/developerworks/rational/library/04/r-3217/index.html
Go
JUN SEP APR
09
2015 2016 2019
16 captures
15 May 2006 - 2 Apr 2019
About this capture

10. B10 User places the order.
    System returns a confirmation number.

Besides the basic flow, there are many alternative flows. The first alternative flow, for instance, describes what happens when the user is a new user (not yet registered with the online bookstore). In the basic flow, the user always has a user ID and password. In contrast, alternative flow 1 describes a case when a first-time user needs to register and provide customer data. Another example of an alternative flow is an invalid password. A user entering the wrong password gets an error message.

Table 1 shows the alternative flows that were included in the use case "place an order":

Table 1: Alternative flows

A1	Unregistered user
A2	Invalid password
A3	No books matching search criteria were found
A4	Decline a book
A5	Continue shopping after storing a book in the shopping cart
A6	Enter a new address
A7	Enter a new credit card
A8	Cancel order

The following convention is used for naming the flows:

Basic flow: B

Alternative flows: A1, A2, A3, ...

Steps in a basic flow: B1, B2, B3, ...

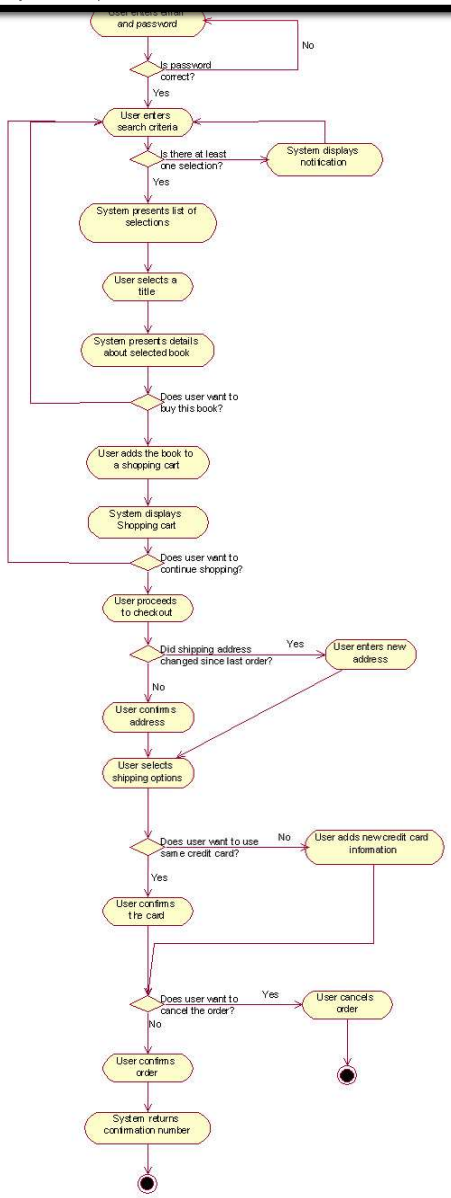
Steps in alternative flow 1: A1.1, A1.2, A1.3, ...

Steps in alternative flow 2: A2.1, A2.2, A2.3, ...

To derive alternative flows, use activity diagrams. Figure 5 displays an activity diagram describing this use case.

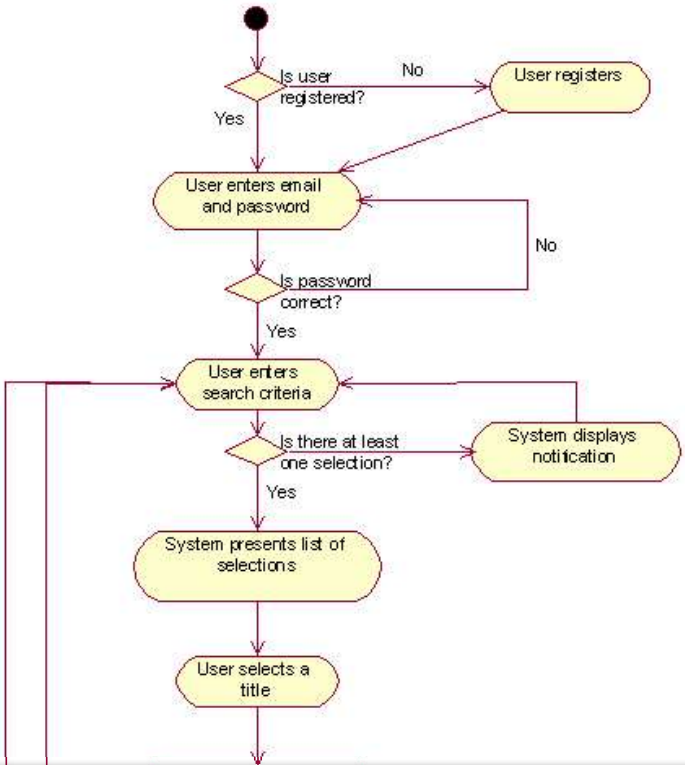
Figure 5. Activity diagram

https://www.ibm.com/developerworks/rational/library/04/r-3217/index.html
Go
JUN SEP APR
16 captures
15 May 2006 - 2 Apr 2019
2015 2016 2019
About this capture

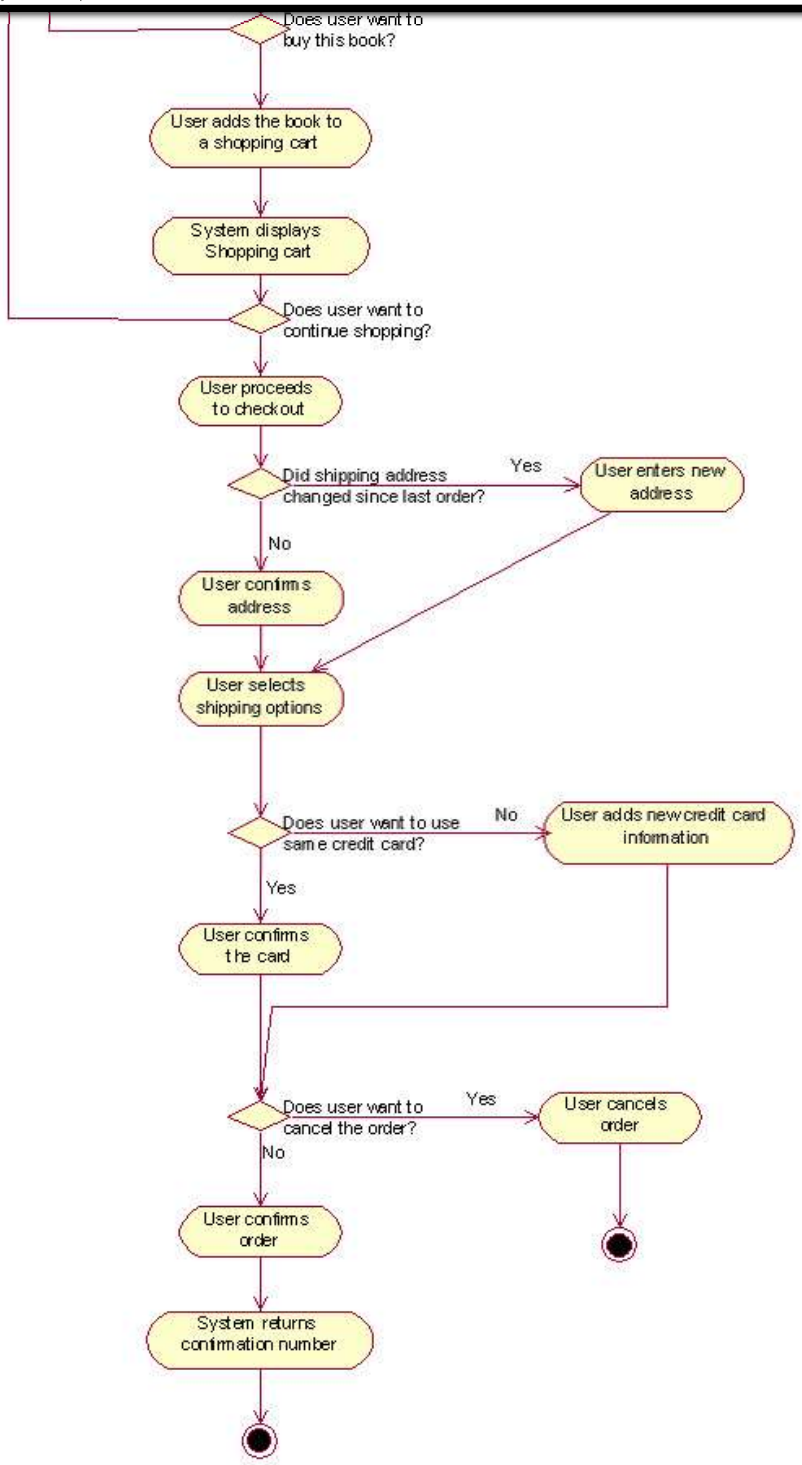


Click to see larger image

Figure 5. Activity diagram



https://www.ibm.com/developerworks/rational/library/04/r-3217/index.html
Go
JUN SEP APR
09
2015 2016 2019
16 captures
15 May 2006 - 2 Apr 2019
About this capture



View a larger version

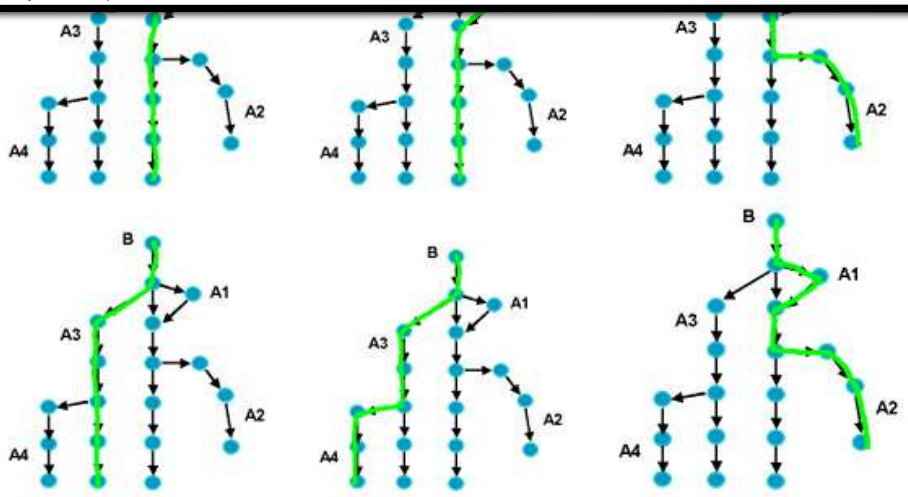
The basic flow is a straight line down, while alternative flows are usually the loops going either back or forth.

How to create test cases from the use cases

Before creating a test case, you need to identify all of the scenarios for the given use case. A scenario is an instance of the use case. It describes one specific path through the flow of events. Figure 6 is a hypothetical graph representing a use case with a basic flow B and alternative flows A1, A2, A3, and A4. To find all scenarios, we need to draw all possible lines through this graph.

Figure 6. Finding scenarios in a use case

https://www.ibm.com/developerworks/rational/library/04/r-3217/index.html
Go JUN SEP APR
16 captures
15 May 2006 - 2 Apr 2019
2015 2016 2019
About this capture



There is one scenario per alternative flow plus one scenario for each combination of alternative flows. There are definitely more scenarios than alternative flows, because there's one for A1, another one for A2, and one scenario which will be a combination of these two.

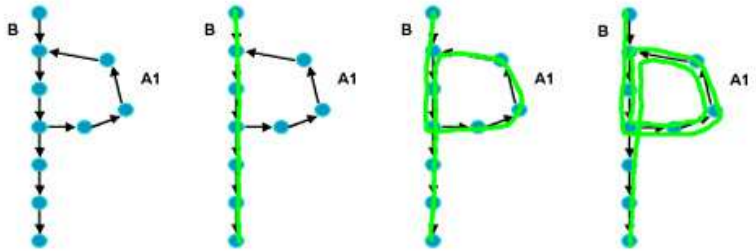
The easiest way to describe a scenario is to provide a sequence of alternative flows, for example, do flow A2 twice, and then do flow A6:

SC16: A2, A2, A6.

Another way to describe a scenario is to list all the steps in it, but this is both more difficult and unnecessarily detailed.

What should you do if you have infinite loops (loops going backwards)? Theoretically it would generate an infinite number of scenarios. Figure 7 shows an infinite loop going backwards.

Figure 7. Infinite loops



The reasonable approach is to do the basic flow once, do a loop once, and then do a loop a second time. If the program works for both instances of the loop, you can assume it will work for all of them.

The book ordering example has a basic flow and eight alternative flows. Four of them are going backwards, and the other four are going forward. If you want to describe all possible use case combinations, you will have over four thousand scenarios (there are eight alternative flows, four of which we may want to do twice because they are loops going backwards, so together it is 2 to the power of (8+4), which is equal to 4096. Obviously we don't need to do all of them.

Choose which ones represent a reasonable subset of these four thousand scenarios. Usually it is wise to select a basic flow, one scenario covering each alternative flow, and some reasonable combinations of alternative flows. Using the examples in Table 1, it probably won't make sense to do a scenario that contains both flows A1 and A7, because they are so far apart on the diagram that they don't have any influence on each other. But it makes sense to do A1 and A2, since they are immediately after each other and may be correlated.

https://www.ibm.com/developerworks/rational/library/04/r-3217/index.html Go JUN SEP APR
16 captures 09
15 May 2006 - 2 Apr 2019 2015 2016 2019
About this capture

going backwards close to each other).

The following 15 scenarios are worth testing:

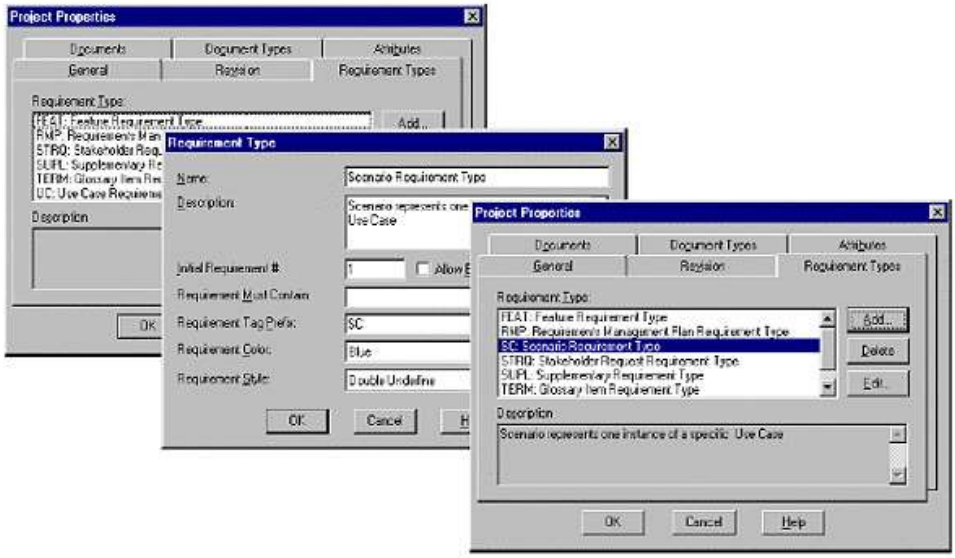
Table 2. Scenarios worth being tested

Table 2: Capturing selected scenarios	
Scenario 1 Basic Flow	Scenario 9 A8
Scenario 2 A1	Scenario 10 A1, A2
Scenario 3 A2	Scenario 11 A3, A4
Scenario 4 A3	Scenario 12 A4, A5
Scenario 5 A4	Scenario 13 A3, A5
Scenario 6 A5	Scenario 14 A6, A7
Scenario 7 A6	Scenario 15 A7, A8
Scenario 8 A7	

How to Create a Scenario in RequisitePro

Scenario is not a standard requirement type in RequisitePro, so you need to add it as a new requirement type. To do that, go to Project Properties, Select the Requirement Types tab, and click Add. Next, fill in the appropriate fields (as shown in Figure 8), and click OK.

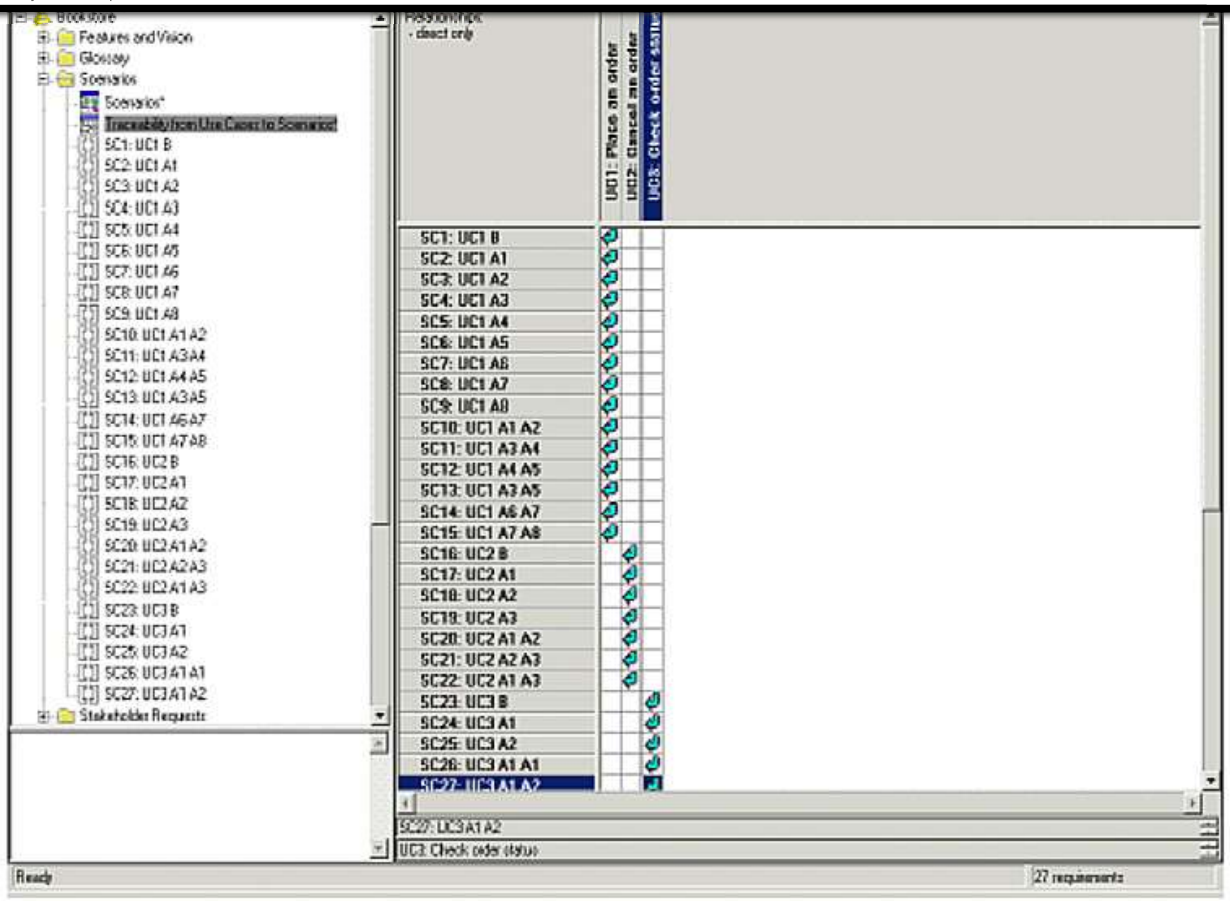
Figure 8: Adding a requirement type scenario



After creating the requirement type, we should enter all scenarios and set traceability from use cases to these scenarios, as shown in Figure 9.

Figure 9: Traceability from use cases to scenarios.

https://www.ibm.com/developerworks/rational/library/04/r-3217/index.html
Go
JUN SEP APR
09
2015 2016 2019
16 captures
15 May 2006 - 2 Apr 2019
About this capture



View a larger version

In RequisitePro, you can name scenarios with the name of the use case and a sequence of alternative flows (for example: UC1, A6, A7).

Now that you have all the scenarios, you need to get the test cases. There are four steps to do that:

1. Identify variables for each use case step
2. Identify significantly different options for each variable
3. Combine options to be tested into test cases
4. Assign values to variables

The following sections describe details of these steps.

Step 1: Identify variables for each use case step

You need to identify all input variables in all of the steps in the given scenario. For example, if in some step the user enters a user ID and password, there are two variables. One variable is the user ID, and the second variable is the password. The variable can also be a selection that the user can make (for instance, Save changes or Cancel).

Here are all of the variables from the book ordering example:

In step B2, there are two variables: e-mail and password. Both of them are strings. In step B3, search a book, the variable is a search string, so it is also a string. In step B4, we need to select a book from a list returned from the system. In step B8, we need to select a shipping option. Amazon.com provides four options.

Step 2: Identify significantly different options for each variable

Options are "significantly different" if they may trigger different system behavior. For example, if we select a user id, which is supposed to be from 6 to 10 characters long, the following entries are significantly

https://www.ibm.com/developerworks/rational/library/04/r-3217/index.html Go JUN SEP APR
16 captures 09
15 May 2006 - 2 Apr 2019 2015 2016 2019
About this capture

Alexandria -- because it is a valid user id

Alexandrena -- because it is too long, and we expect the system to prevent us from entering a user id that long

However, "Alexandria" and "JohnGordon" are not significantly different, because they are both valid user ids that should cause the system to react in the same way.

The following guidelines describe some specific cases.

An option can be considered significantly different if:

1. It triggers different flow of the process (usually an alternative flow)
   Example

   Entering invalid password will trigger Alternative Flow 2
2. It triggers different error message
   Example

   If email is to long, the message is "Email should have no more than 50 characters"

   If email does not contain @ sign, the message is: "Invalid email address"
3. It causes different appearance of the user interface
   Example

   If Method of Payment is a credit card, fields to enter credit card number, expiration date and cardholder name are shown
4. It causes different selection to be available in the drop-downs
   Example

   The customer registration screen may contain drop-downs "Country" and "State/Province". The drop-down "State/Province" is populated based on the country selected: for the US it contains all the states, for Canada all the provinces, and for other countries it is grayed-out. This creates three different options:

   US

   Canada

   Any other country
5. It is an input to some business rule
   Example

   Assuming there is a rule "If the order is placed after 6pm, and user select Overnight Shipment, the message should inform that the book will arrive after tomorrow", we may have two separate options:

   Overnight Shipment, order placed before 6pm

   Overnight Shipment, order placed before 6pm
6. It is a border condition
   Example

   Since password should have at least 6 characters we should test:

https://www.ibm.com/developerworks/rational/library/04/r-3217/index.html Go JUN SEP APR
16 captures 09
15 May 2006 - 2 Apr 2019 2015 2016 2019
About this capture

7. Something is changed vs. the default is used
   Example

   On the credit card payment screen the cardholder's name is populated with the name of a person placing the order. This creates two separate options:

   Keep default cardholder's name

   Change cardholder's name to a different one
8. The entry format is not clearly defined and may be differently interpreted by the user
   Example

   Phone numbers are written differently by different people:

   Using brackets (973) 123 4567

   Using dashes 973-123-4567

   Plain number with spaces 973 123 4567
9. When regular cases differ in different countries
   Example

   Credit card expiration date format may be different in the USA and in Europe

If we are testing numbers, we may consider the following options:

Regular number, reasonable from the application point of view

Zero

Negative number

A number with two decimals

The biggest number that can be entered (99999999999999 - as many nines as can fit)

How do you know what is the minimum and maximum allowed length of a field? This requirement can come from different sources. Sometimes it comes from the business analyst or a customer. For example, if we enter a Dun and Bradstreet number that identifies a company, it should always be a number containing 9 digits. It is a business requirement.

Quite often, however, it doesn't come from the customer or the user. If you ask the customer how big the last name field should be, they might say that they don't care and ask you to make it whatever is reasonable. In this case it is a design step rather than a requirement step to decide how long the variable should be.

In another situation, it may be suggested by the data analyst or database designer-- for example, if all other applications in the corporation store last names in 30-character long fields, your application should probably comply with this standard as well.

Regardless of the source of the requirement, it should always be agreed upon and documented before we do the test cases.

There is a question about where requirements like those just discussed should be documented. One place to add this kind of requirement is a paragraph called Special Requirements in the use case.

https://www.ibm.com/developerworks/rational/library/04/r-3217/index.html Go JUN SEP APR
16 captures 09
15 May 2006 - 2 Apr 2019 2015 2016 2019 About this capture

application. This makes sense especially if the same variable appears on many screens in many use cases, so you could say in one document that all the names are up to 30 characters and all the addresses are up to 100 characters. However, if they are specific to a use case, it is better to add them to special requirements in that use case.

Table 3 shows options that were identified for variables in the basic flow of the sample project:

Step	Variable	Options to be tested						
B1	Website	Actual URL						
B2	Email	Regular	Blank	Min allowed (1 char)	Max allowed (50 char)	One more than allowed (51 char)	Very long (257 char)	Invalid (no @ sign)
B2	Password	Regular	Blank	Too short (5 char)	Min allowed (6 char)	Max allowed (10 char)	One more than allowed (11 char)	Very long (257 char)
B3	Search string	Regular	Blank	Min allowed (1 char)	Max allowed (300 char)	One more than allowed (301 char)		
B4	Selection	First selection	Last selection					
B5	Action selection	Add to shopping cart						
B6	Action selection	Proceed to checkout						
B7	Shipping address	Confirm the address on file						
B8	Shipping method	5 days	3 days	2 days	Overnight			
B9	Payment method	Confirm the credit card on file						
B10	Action selection	Place an order						

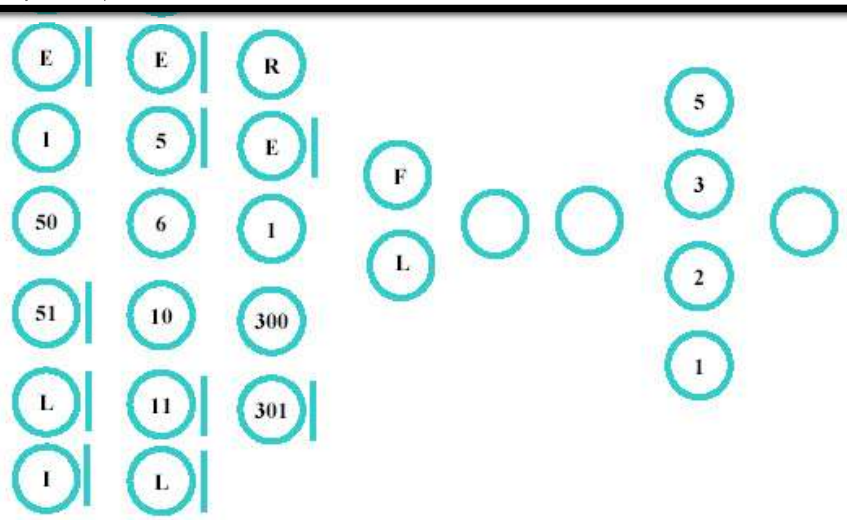
Step 3: Combine options to be tested into test cases

In the previous step you identified all the options. In this step, you need to combine them in the sequence of test case steps.

Figure 10 graphically illustrates the options to be tested. In each column, there is an input variable to be tested, and each row is one option: R is regular, E is empty, and then one character, 50 characters, 51, and so forth. "L" means very large, and "I" means illegal.

Figure 10: Options to be tested for each step

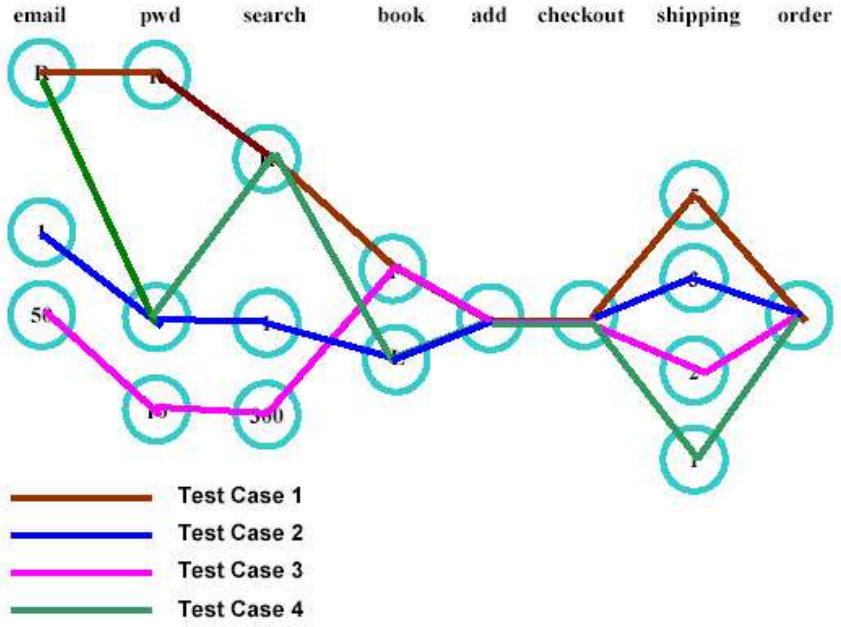
https://www.ibm.com/developerworks/rational/library/04/r-3217/index.html
Go
JUN SEP APR
09
2015 2016 2019
16 captures
15 May 2006 - 2 Apr 2019
About this capture



The options that have the bar after them throw the user out of the basic flow: they represent some errors that are described in alternative flows. Because you are currently designing test cases only for the first scenario, you can remove them (they will be tested in some other scenario). From whatever is left, you need to create a minimum number of test cases that cover all the conditions.

Create test cases by connecting circles, as shown in Figure 11.

Figure 11: Combine options to create test cases



To create the first test case, you can pick and connect any options. When you create the second test case, pick one of the options that was not used in the first one. Continue adding test cases until all nodes of the graph (as shown in Figure 11) are covered. Usually you'll need from 4 to 6 test cases to cover all the options that should be tested. However, some specific situations may require more.

Allocation of test cases can also be represented in the form of a test case allocation matrix, as shown in Table 4.

Step number	Variable or selection	TC1	TC2	TC3	TC4
B1	Website	Actual URL	Actual URL	Actual URL	Actual URL
B2	Email	Regular	Min allowed (1 char)	Max allowed (50 char)	Regular
B2	Password	Regular	Min allowed (6 char)	Max allowed (10 char)	Min allowed (6 char)

https://www.ibm.com/developerworks/rational/library/04/r-3217/index.html
Go JUN SEP APR
16 captures 09
15 May 2006 - 2 Apr 2019 2015 2016 2019
About this capture

B3	Search string	Regular	Min allowed (1 char)	Max allowed (300 char)	Regular
B4	Selection	First selection	Last selection	First selection	Last selection
B5	Action selection	Add to shopping cart	Add to shopping cart	Add to shopping cart	Add to shopping cart
B6	Action selection	Proceed to checkout	Proceed to checkout	Proceed to checkout	Proceed to checkout
B7	Shipping address	Confirm the address on file	Confirm the address on file	Confirm the address on file	Confirm the address on file
B8	Shipping method	5 days	3 days	2 days	Overnight
B9	Payment method	Confirm the credit card on file	Confirm the credit card on file	Confirm the credit card on file	Confirm the credit card on file
B10	Action selection	Place an order	Place an order	Place an order	Place an order

Table 4 describes the graph from Figure 11 in the form of a matrix where every column contains a different test case. Each row corresponds to one variable entered by a user.

Step 4: Assign values to variables

In this step, you replace placeholders like "a very long last name" or "a long phone number with extension" with actual values, like "Georgiamitsopolis" and "011-48 (242) 425-3456 ext. 1234" respectively.

In this step you also split all the test cases from the matrix shown in Table 4, creating a separate table for each test case.

For Test Case 1 of Book Order Use Case, you will have a table like that shown in Table 5. This will be a document that you give to a tester. The tester will follow the directions from columns 2 and 3, and record the results in columns 5, 6, and 7.

Table 5: Final test case

Step number	Variable or selection	Value	Expected result	Actual result	Pass/Fail	Comments
B1	Website	www.amazon.com	Logon Screen			
B2	Email	jsmith@hotmail.com				
B2	Password	Johnsm	Main Screen			
B3	Search string	“Rational”	List of books			
B4	Book selection	First selection	Book details			
B5	Action selection	Add to shopping cart	Cart contents			
B6	Action selection	Proceed to checkout	Prompt for address			
B7	Shipping address	Confirm the address on file	Prompt for shipping			
B8	Shipping method	5 days	Prompt for payment			

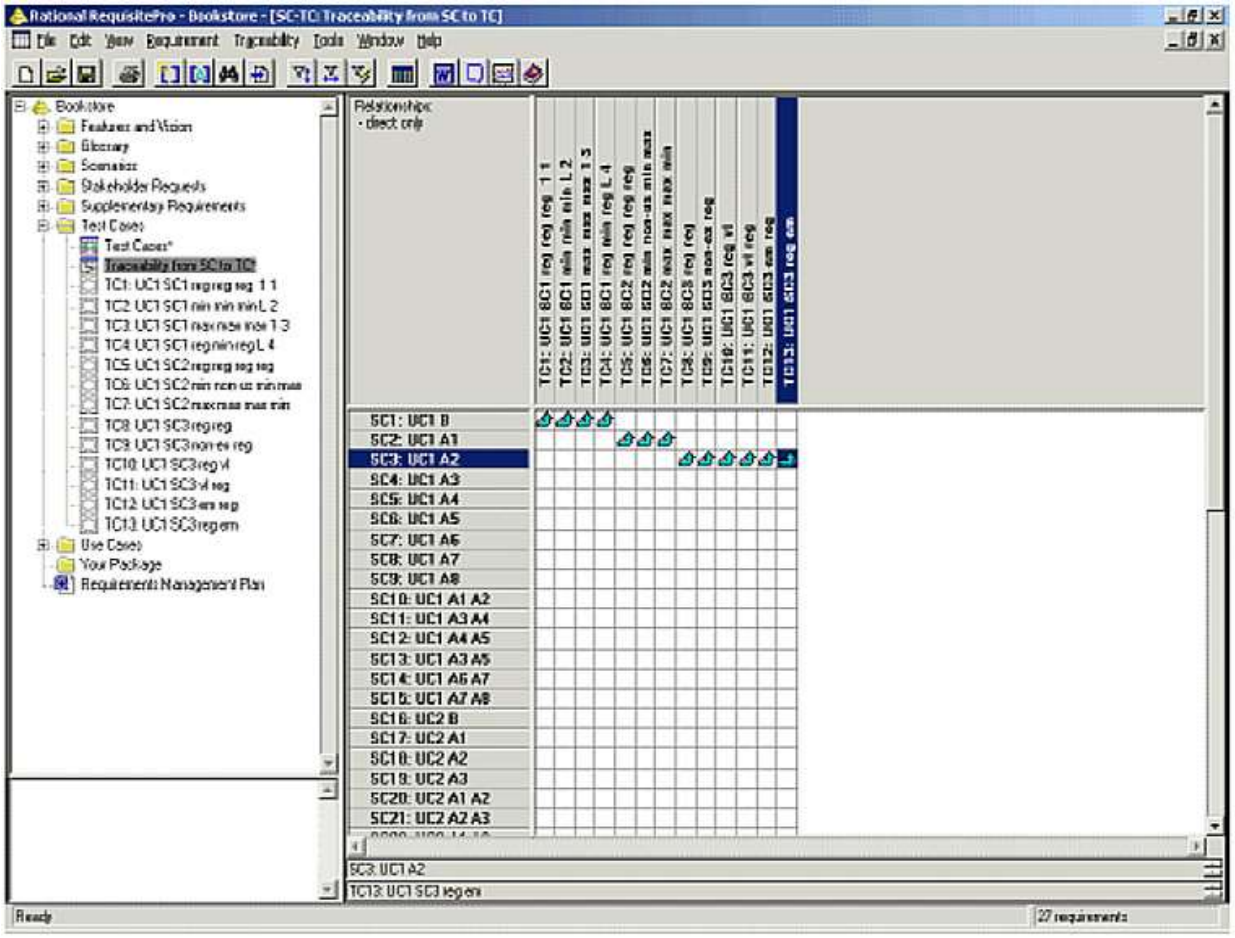
https://www.ibm.com/developerworks/rational/library/04/r-3217/index.html Go JUN SEP APR
16 captures 09
15 May 2006 - 2 Apr 2019 2015 2016 2019 About this capture

B9	Payment method	Confirm the credit card on file	Prompt for confirmation				
B10	Action selection	Place an order	Order number				

Once again, RequisitePro helps you to create traceability. After producing all your test cases, you can set traceability from scenarios to test cases.

Figure 12 shows all the scenarios: 21 scenarios derived from different combinations of alternative flows.

Figure 12: Traceability Matrix



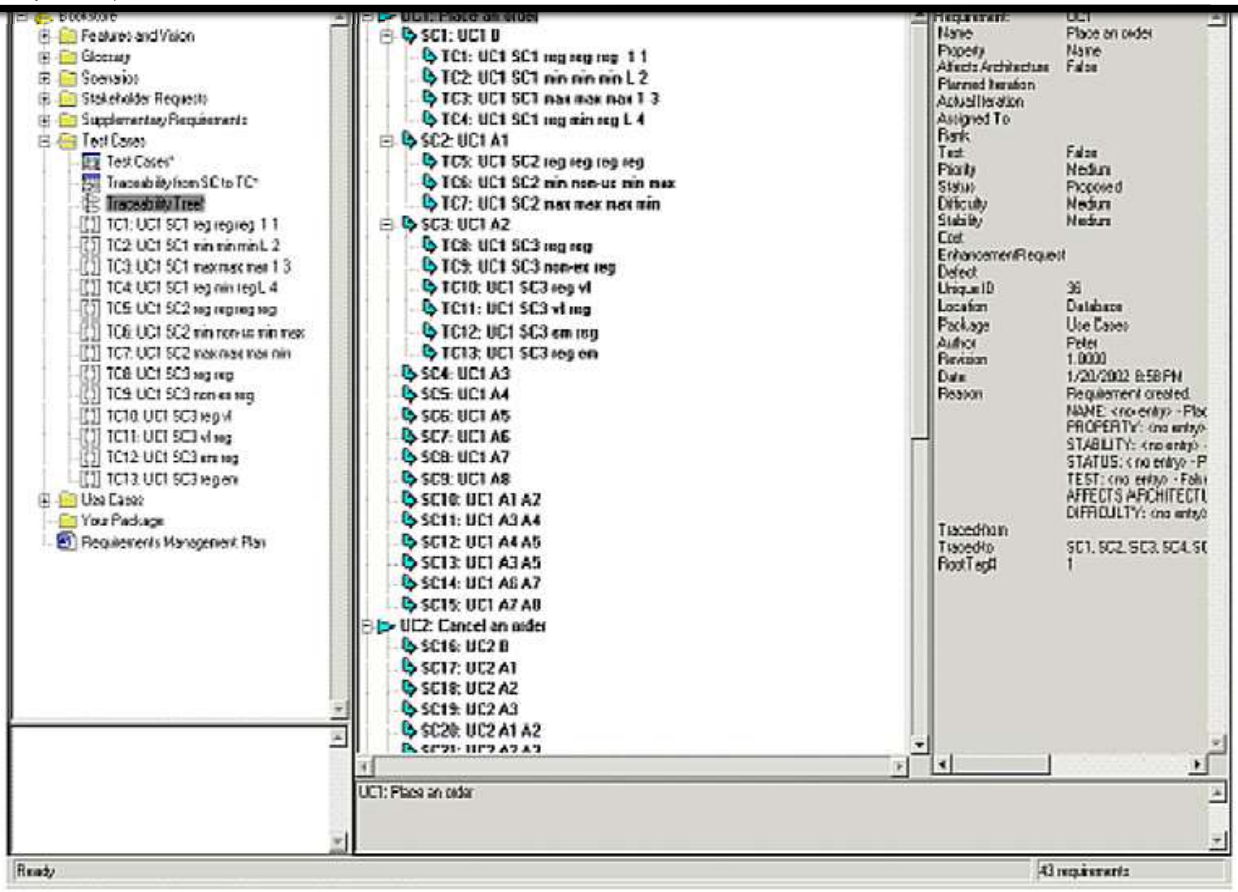
View a larger version

After setting the traceability between scenarios and test cases, we can create a traceability tree that shows traceability all the way from use cases to the test cases.

There are two options. The first option -- shown in Figure 13 -- is to trace out of the use case, which shows use cases on the top level and tracing to scenarios and test cases.

Figure 13: Traceability tree from use case

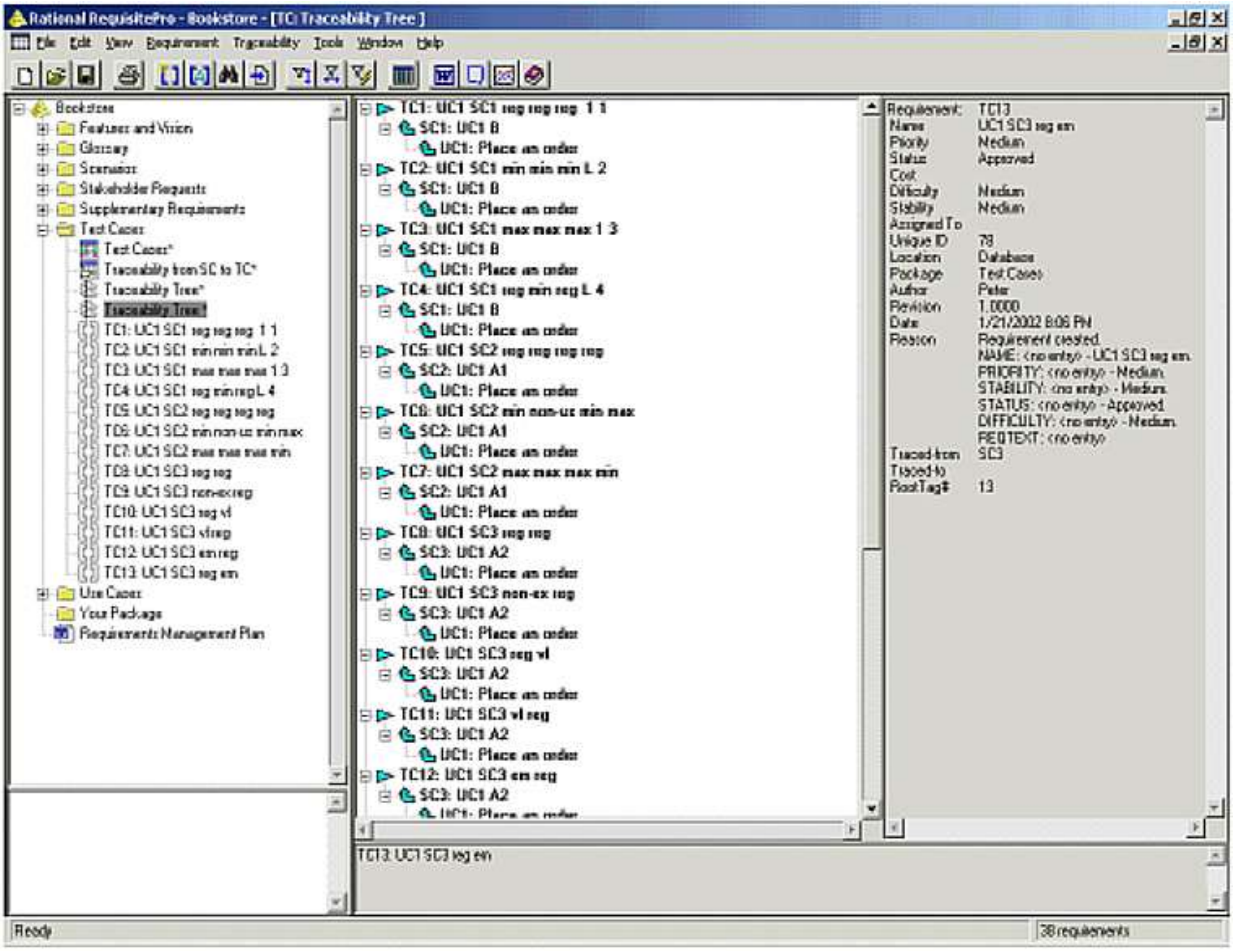
https://www.ibm.com/developerworks/rational/library/04/r-3217/index.html
Go
JUN SEP APR
09
2015 2016 2019
16 captures
15 May 2006 - 2 Apr 2019
About this capture



View a larger version

The second method is traceability into test cases, shown in Figure 14. In this case the tree looks different: you start with test cases, and then trace back from scenarios and use cases.

Figure 14: Traceability tree from test case

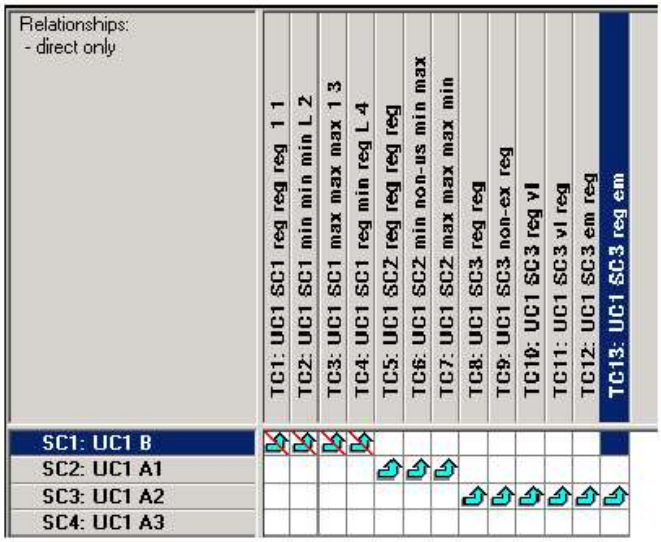


View a larger version

https://www.ibm.com/developerworks/rational/library/04/r-3217/index.html
Go JUN SEP APR
16 captures 09
15 May 2006 - 2 Apr 2019 2015 2016 2019
About this capture

Figure 15, show you which test cases might have been changed because a previous scenario and use case changed.

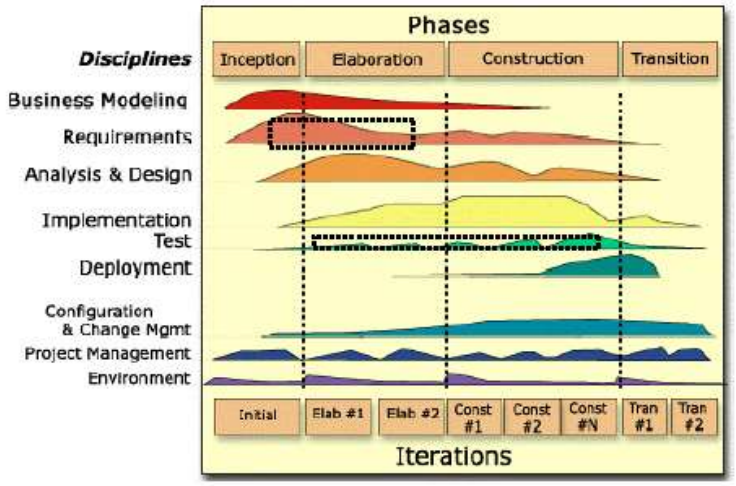
Figure 15: Suspect relationships



Mapping to the IBM Rational Unified Process

How do these activities map to the IBM Rational Unified Process (RUP)? Most of them take place in the Inception and Elaboration phases quite early in the process. Just after you have use cases, we can start doing scenarios and test cases. Figure 16 depicts where the activities fit in the RUP methodology.

Figure 16: Traceability activities mapped to RUP phases



While doing scenarios and test cases, you can give feedback to use case designers and refine requirements. This can help shift some tasks early on in the process, and eventually contribute to the team's ability to finish the project sooner. Test cases are used throughout Elaboration, and almost the whole Construction phase.

Conclusions

The article presented a method of deriving functional test cases from use cases. Here are some benefits of this approach:

- Test cases are derived in a more automatic way
- Avoids duplicate testing
- Better test coverage
- Easier monitoring of testing progress

https://www.ibm.com/developerworks/rational/library/04/r-3217/index.html
Go
JUN SEP APR
09
2015 2016 2019
16 captures
15 May 2006 - 2 Apr 2019
About this capture

Decreases project time by moving some tasks from Construction to Elaboration

Contributes to early discovery of missing requirements

The test cases that you create can be used for manual testing, as well as for automated testing using tools like IBM Rational Robot?. This method has been successfully used in multiple projects.

Resources:

1. Jim Heumann, "From Use Cases to Test Cases - Ensuring Quality from the Beginning." RUC 2001.
2. Jim Heumann, "Using Use Cases to Create Test Cases." The Rational Edge, June 2001.
3. Dean Leffingwell and Don Widrig, "Managing Software Requirements: A Unified Approach". Addison-Wesley, 1999.
4. Dean Leffingwell and Don Widrig, "The Role of Requirements Traceability in System Development", The Rational Edge, September 2002.
5. Rational Unified Process. Rational Software Corporation, 2001.

Click here to view original RUC presentation of this article.

Dig deeper into Rational software on developerWorks

Overview

Free online course: Getting started with IBM Bluemix

IBM Bluemix Garage Method

Technical library (tutorials and more)

Forums

Communities

developerWorks Premium
Exclusive tools to build your next great app. Learn more.

developerWorks Labs
Technical resources for innovators and early adopters to experiment with.

IBM evaluation software
Evaluate IBM software and solutions, and transform challenges into opportunities.