TRICENTIS

Test Design Specialist | Level One

LESSON TRANSCRIPT

SURVEY

Tricentis Test Design Specialist | Level One

Student Exercise Workbook

- Version 2019_04
- Designed to be used with Tricentis **Tosca version 12.x**

Lesson Transcript

This lesson Transcript provides the scripts used during the lesson videos for the Tricentis Test Design Specialist | Level 1 training.

Legal Notice

Tricentis GmbH
Leonard-Bernstein-Straße 10
1220 Vienna
Austria

Tel.: +43 (1) 263 24 09
Fax: +43 (1) 263 24 09-15
Email: academy@tricentis.com

CONTENTS

Survey **2**

PREFACE **2**

About this workbook 2

Introduction **4**

Lesson ONE - TEMPLATES **7**

Lesson 1 • Requirements 7

Lesson 2 • Attributes 11

Lesson 3 • Instances 13

Lesson 4 • Combinatorial Methods 15

Lesson 5 • TestCase Specifications 18

Lesson 6 • Classes 20

Lesson 7 • Link To Requirements 22

Lesson 8 • Integration of New Attributes 23

Lesson 9 • Best practices 24

Lesson 10 • Automatic Generation of Instances 26

PREFACE

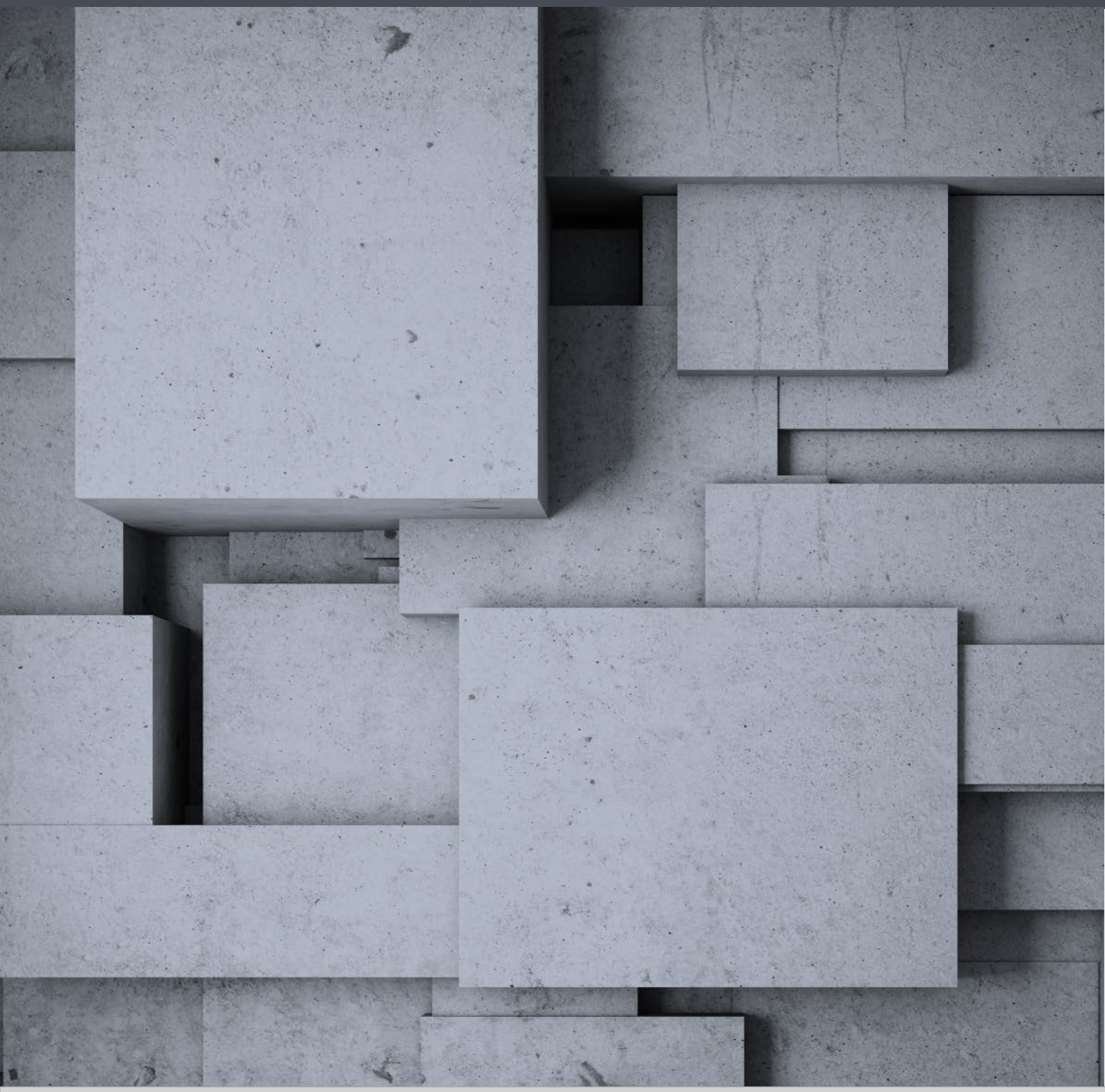
About this workbook

This transcript is specifically designed to supplement training of the Tricentis Test Design Specialist | Level 1

This transcript is divided into 10 Lesson sections and supplies the script used for the lesson videos.

This transcript is not aiming to be a complete manual.

TRICENTIS



GETTING STARTED

INTRODUCTION

Welcome to the Test Design Specialist Level 1 Course!

This course is meant to prepare you to become a Test Design Specialist, which means learning how to work in the yellow and red sections of Tricentis Tosca – that is Requirements and TestCase Design.
Before you begin, let's discuss why this is important and how it fits into the classic software testing life cycle (STLC). The cycle begins by analyzing the requirements for the software to be tested. Here we try to answer questions like:

- What are the requirements for my software?
- What areas need to be tested?
- What risk is attached to these requirements?

This will all be done in the Requirements section of Tricentis Tosca.

Once this stage has been completed, you will move on to the test planning stage. This involves estimating how many test cases are needed to cover the requirements you have defined, what these tests will look like later, and which data combinations need to be tested. This is done in the TestCase Design section.

This information is used in test case development, which you learned in the Automation Specialist Courses.

Once the Test Environment has been setup, the TestCases are executed and the execution results are analyzed, defect analysis is undertaken and finally reports of the test results are created. These results are then projected onto the requirements to determine what is left to be done as part of the cycle. And the software cycle beings again. We will focus here on the first two phases of this cycle.

Requirements

Defining Requirements is essential for effective test management. This definition is usually done at the beginning of the test process and can be done directly in Tricentis Tosca. It is possible in Tosca to track which Requirements have already been tested during the test process and which Requirements have not yet been tested fully. This helps you obtain an overview of the current state of the test progress.

Through risk priority assessment, you will be able to weight your requirements to determine where the greatest risk lies and where you should begin testing. With this, it is possible to determine what risks have already been covered and how high the risk would be to go into production at any given time.

Why is this important? First of all, you can ensure that the test focus can be placed on the most critical areas. Secondly, problem areas of your System Under Test are discovered early which means that preventive activities can be taken immediately.

And finally, risks can be continuously monitored to measure the status of the project and its quality

TestCase Design

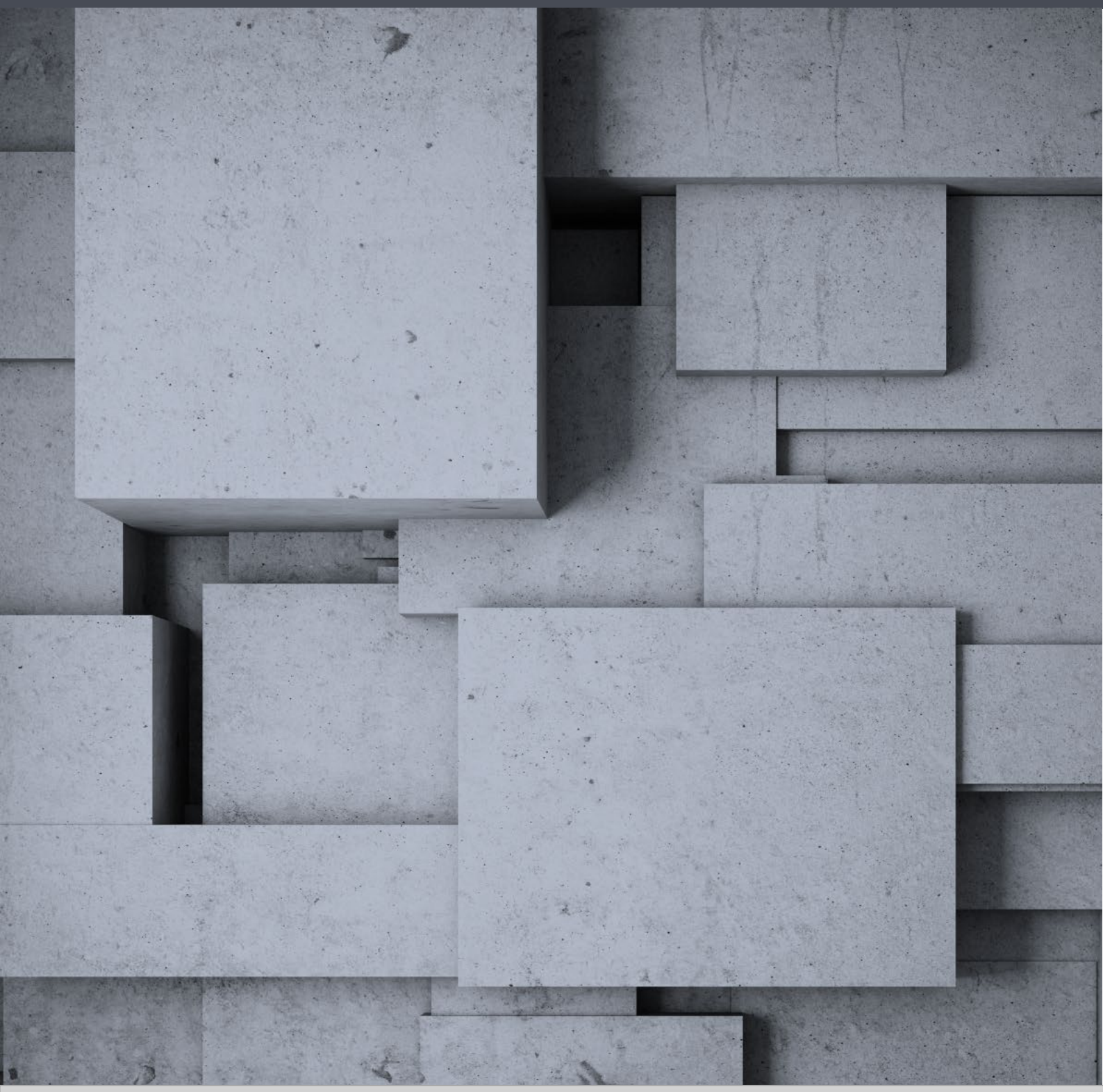
Once you have set up your Requirements, you would want to start planning or specifying your TestCases. This specification means receiving information about the application from developers and test data from business analysts, and translating this into a business readable format. During the testing process in many projects, only about a quarter of the time is spent specifying test cases, usually using an intuitive approach.

This often results in having redundant test cases that cover the most important requirements more than once and that do not cover Requirements with low risk. Then about three-quarters of the time is spent in execution. This large chunk of execution time is then repeated during regression tests over and over. If the time spent executing tests could be minimized, this would greatly affect the overall time spent testing.

The Tricentis approach to Test Case Design is based upon more time being invested up front in TestCase specification, which results in less time required for the actual execution. Investing time in the design wisely can save valuable time and money, while increasing the quality of the test. And it's not just about designing fewer TestCases, but also more precise TestCases, to avoid redundancies and maximize test coverage. With the Tricentis approach, each TestCase has a clear defined goal to make error analysis easier. These TestCases are then easier to maintain and adjust. Test case design can also help you organize your smoke tests, regression tests, and one time tests. TestCase Design in Tosca allows you to ensure a more efficient, and effective testing process.

Finally, the link back to Requirements ensures that you have a good overview of the risk contribution of each test case you have designed. The Requirements section then serves as a test management dashboard, to get a quick overview of the current state of your entire project.

We will explain the methodology in more detail as you proceed through the course. So with that, let's get started!



LESSON ONE - REQUIREMENTS

LESSON ONE - TEMPLATES

Lesson 1 · Requirements

In this lesson, you will learn how to create, structure and weight Requirements in Tricentis Tosca.
The classic testing life cycle begins with defining the Requirements of the software to be tested. This means identifying the criteria which the system is expected to fulfil. These criteria are the system Requirements.

A Requirement can be a functional need for the application - for example, a user should be able to order a product with a Visa credit card – or non-functional – for example, system security.

Defining Requirements allows both developers and testers to have an overview of what processes and functionalities are needed by the project. In a waterfall approach, this means defining the Requirements for the release of the entire application, whereas in an agile approach, this means defining the Requirements for a specific sprint or build release.

Once Requirements are defined, they can and should also be risk weighted. This means determining the level of risk, either financial or otherwise, associated with the failure of a particular Requirement. This is important because it assists testers in deciding where to start testing. By keeping track of which Requirements have already been tested and the results of the test executions, testers then have a good idea of the state of the project – in other words, how much of the business risk has already been covered and how much is left to be done. Developers then can also use the risk weighted Requirements to prioritize functionalities and bug fixes. This risk based approach to testing ensures that we have the minimal amount of risk that a release of our application will result in losses or damages.

All of this can be done in Tricentis Tosca, using the Requirements section. In the Requirements section it is possible to set up the Requirements, structure them, and to risk weight them. Once all of this is done, the Requirements can be linked with the other sections of Tosca, such as TestCase Design, TestCases and the ExecutionList section. This enables you to use it as a dashboard to have full oversight over the progress of the test portfolio and to create reports for the management.

Before you begin the physical set up in Tosca, you will need to determine the best way to gather your Requirement data. You may either start building the Requirements from scratch in workshops with the business department and other stakeholders from the project or you can use an existing structure given to you by a business analyst. Functional and technical description documents may also be used as sources for setting up the Requirements structure.

To begin in Tosca, it is highly recommended to first create a Folder structure in the Requirements section reflecting the system structure or the testing process. Creating Requirement Folders uses the same process as in every other section of Tosca – you simply right click and Create Folder. You can structure these with sub-folders as needed. This same Folder structure can then be used in other sections as an overall project folder structure.

Within these Folders, Requirements will be organized within **RequirementSets.** These RequirementSets contain groups of Requirements which represent different aspects of the application (for example, frontend, backend or non-functional in a waterfall project) or RequirementSets can be set up to reflect testing-process-related views, like regressions or sprints. These can be added inside a Folder by selecting “Create RequirementSet” or by using the shortcut “Ctrl+N, Ctrl+R”.

To create Requirements within the RequirementSet, you will select “Create Requirement” or the shortcut “Ctrl+N, Ctrl+R”.

Requirements can be layered, meaning that each Requirement can contain sub-Requirements. Requirements should be structured in a way which eliminates redundancies. The structure should be clear and the structure should not be too deep or complex. In other words, it should be easily readable.

The best way to begin organizing the Requirements is to define a requirement structure considering the highest-level functionalities (e.g. customer, account, products, order process) and continue by expanding these levels by adding sub-requirements.

The lowest level of sub-Requirements, or **Leaf Requirements**, should be modeled down to the level of use cases or user stories. Anything more specific is too detailed. Good structuring and naming upfront is important to maintain a good overview of the entire project via the Requirements section.

For example, to support an Agile structure and to reflect continuous delivery, the Requirements structure can be set up according to the testing team's sprint. Each Requirement is named after the User story and should be numbered.

To put this into perspective, we can look at an example from the DemoWebShop. In this example, we will begin in the Requirements with a folder called "Webshop" which will contain a RequirementSet that will be used as a container for all our frontend Requirements. Here now we can define all the needed Requirements for the release of the frontend of our application. The highest level of Requirements will be customer tasks, handling products, shopping cart related Requirements, and order process Requirements. Once these higher levels are defined, more detailed Requirements will be specified – for example, customer tasks consist of registering, logging in, modifying account data, and checking previous orders. This is specific enough, as the various ways to test the logging in process, for example, will be considered test cases used to test this Requirement.

Once the Requirements are defined and structured, we will move on to weighting each Requirement on all levels. The weighting process is meant to give an indication of which Requirements have the greatest level of risk associated with a failure of the functionality. Tricentis recommends that you always begin by testing those functionalities with the highest business risk.

There are two ways to weight the Requirements in Tosca – basic weighting and complex weighting.

Basic Weighting

To begin with basic weighting, ensure that the **weight** column is visible in the Requirements section. You will see that the default weight for each Requirement is 1, which means that all Requirements are weighted equally. Simple weighting allows you to assign values to each Requirement in your RequirementSet. They can then be compared to all other requirements on the same level as well as the RequirementSet as a whole. These values are linear, and can contain any value between 0 and 10 (10 being the highest weight). Best practice is to use a simple scale from 1 to 5.

The best way to do this is to close all sub-Requirements and begin on the highest level. Assign the values next to the respective Requirement, 1 being the lowest weight and 5 being the highest weight. You can then expand each Requirements and enter the Values for the Leaf Requirements underneath. The highest level Requirements can show us which area to begin looking in for testing, and the Leaf Requirements will indicate exactly which requirement to focus on.

Complex Weighting

In the assessment of Requirements, a linear scale may not provide you with distinctive values or it may be too difficult to distinguish between a simple weight of 3 or 4. To overcome this, Tosca offers a more sophisticated risk weighting methodology which uses 2 Values related to the business – the potential financial loss associated with the failure of a Requirements, also called the "Damage Class", and how heavily a function is used in the application, also called the "Frequency Class". Each of these is a column in the Requirements section, and can be added using the column chooser. Each Requirement can be assigned both a Frequency Class and a Damage Class.

1) for Frequency Class you would ask the question: How often is this functionality used?
2) and for Damage Class you would ask the question: How much financial damage would we incur if this functionality failed?

It is important to remember that these numbers are simply estimations and comparisons to the other Requirements.

Both can be assigned a Value in the range 0 to 10, similar to the Basic Weighting column. Again, best practice is to use a range of 1 (being the lowest level of financial damage risk or the least frequently used functionality) to 5 (the highest level of financial risk or the most frequently used functionality.

This weight is calculated as an exponential function using a base value of 2. So the formula would be: 2 to the power of Frequency Class times 2 to the power of Damage Class. When you add the Frequency Class and Damage Class, you will see that the Weight is calculated automatically by Tosca according to this formula. You can manually override the Weight if necessary. You can also adjust the base Values on the Properties tab for the RequirementSet.

Using an exponential weighting system allows for a wider and more clear spread between Requirements and helps to support the decision making when determining which Requirement to begin testing.

As with simple weighting, we will begin with the highest-level Requirements first, and then expand and weight the Leaf Requirements.

Use as a dashboard

With the Requirements all weighted, we can now compare the Requirements against other Requirements in the same level or with the overall RequirementSet. To do this, ensure that the columns "Contribution (%)" and "Relative Weight (%)" are visible using the Column Chooser. The column **Contribution (%)** allows you to compare over the whole RequirementSet. This shows the relative percentage of risk coverage for each Requirement in a RequirementSet according to its and its parent's weights. Values in the column **Contribution (%)** are multiplied with their parent in order to show their contribution in comparison to the whole set. The column **Relative Weight** shows the calculated weight in percentage compared on the same level. Values for elements on the same level always sum up 100 for **Relative Weight,** as they are compared to each other.

As mentioned before, other sections in Tosca can be linked to the Requirements section for a better overview of the entire project. TestSheets in TestCase Design will be linked to Requirements (which we will cover later in this course), Test Cases are linked to Requirements, and ExecutionLists are linked to RequirementSets. The column **ExecutionState** shows the result of the executed TestCases taking the risk into account leading to a risk-based coverage statement.

In Tosca versions 10.2 and before, your Requirements will auto-update as soon as an object is linked to them. This means the Dashboard will automatically recalculate all fields, including Execution State and Coverage. In Tosca 10.3 and later, the Requirements section does not auto-update by default. This is because in large repositories with many users and objects, every time a change is made which affects the Requirements, the recalculation can take some time. If you are working in a smaller project, this will not make a significant difference, therefore It may be nice to turn the auto-update setting on so that you always have a real-time view of the project. You will be alerted when a change has been made which has not yet been updated in the Requirements section. This will be indicated by an exclamation mark next to the RequirementSet.

To update, select the relevant RequirementSet and either press F6 or click the button "Update Outdated Values" in the Ribbon. To change this setting in Tosca 10.3 and later, go to *Project*>> *Settings* >> *Commander*>> *General*>> *Advanced*. The setting "AutoCalculateRequirements" will be set to OFF. Selecting "on" will enable the auto update as it was in previous versions. For the training, we have enabled autocalculate.

Lesson 2 · Attributes

In this lesson, you will learn how to create and structure Attributes in the TestCase Design section of Tricentis Tosca.

Theory

Let's start with a recap of some of the theory you covered in Automation Specialist Level 2.

The TestCase Design section of Tosca is where you specify and design the TestCases you will later want to build in the TestCase section. TestSheets are the highest-level components of the TestCase Design section, and it is within these TestSheets that you organize your test data and logical test structure. Each Requirement of your application should be covered by a TestSheet, and the relevant TestCase required to test this Requirement will be designed within this TestSheet.

The functionality of your application depends on the behavior of various business objects. The interactions with these business objects in your physical TestCases should be mapped in TestSheets using Attributes. Attributes are the various characteristics of the application, or representations of these business objects. An example of an Attribute in our sample application would be a credit card drop-down list. We would then add the variations of this Attribute later as Instances (for example, the types of credit cards we can choose from like Visa, Mastercard, and so on). We will discuss Instances in the next lesson, so we will stick to Attributes for now.

Attributes can be structured like Requirements. Every Attribute can have sub-Attributes, depending on how you have chosen to design the TestSheet. Each Attribute has a defined Attribute type property (written "AttrType") which can have the Value either "Logical" or "Physical". This is not user defined, but actually defined by Tosca. This means simply whether an Attribute has sub-Attributes (logical) or not (physical).

We recommend creating 4 logical Attributes in every TestSheet for structure. These are "Administration", "Precondition", "Process", and "Verification".

1. The **Administration** Attribute will contain information about the TestCases that will not contain actual test data, but may contain information that is relevant to someone who is reading the TestSheet. For example, who designed this TestCase, who is the contact person for this data, what is the test stage, or any additional comments.

2. The **Precondition** Attribute will contain prerequisites for your TestCase. That means data which is required to set up the Workspace or application in the right configuration for the TestCase, but is not the actual focus of the TestCase, for example data for logging into an application.

3. The **Process** Attribute will contain all of the data that will be tested in these TestCases. This information is both relevant for the TestCase to run and relates directly to the focus of the TestCase.

4. Finally, the **Verification** Attribute is where you should specify all data used for verifications or comparisons to test whether the process steps have functioned as expected.

Each Attribute also has the property "BusinessRelevant", and this property can be set by the user. There are three possible Values that can be selected for this property – either "yes", "no", or "result".

1. "Yes" is the default Value. This means that an Attribute is business relevant, meaning it will affect the TestCase and should be considered when deciding which combinations of data to use to create the test cases. This could be used for the Precondition and Process Attributes. All sub-Attributes of these Attributes will adopt the selected business relevance.
2. "No" means that the Attribute specified, and its sub-Attributes are not relevant to the TestCase being built and should not be considered when combining data. This could be chosen for the Administration Attribute.
3. "Result" indicates that the Attribute selected, and its sub-Attributes contain data that will be used for verification in the TestCase. This could be selected for the Verification Attribute.

Demonstration

To set all of this up in Tosca, you will go to the TestCase Design or the red section. Create structural Folders which will contain the TestSheets relevant to your project. Best practice is to set up the Folder structure in this section to reflect the structure of your Requirements section. This will not be the case in our current Workspace due to training purposes.

On the TestCase Design folder which should contain your TestSheet, you have 3 ways to add a TestSheet. In the Ribbon, you can select create object->TestSheet. You can also right click on the Folder and select "Create TestSheet" from the Mini-Toolbar. Finally, you can use the shortcut "Ctrl+N, Ctrl+T".

To add Attributes to a TestSheet, you have the same three options. Select Create object->Attribute from the ribbon, right click and select "Create Attribute" or use the shortcut "Ctrl+N, Ctrl+A".

On the properties tab of the Attribute itself, you can select from the dropdown box whether you want the Attribute to have Business Relevance "Yes", "No", or "Result". With the Attribute selected, this can also be toggled using Ctrl+F7 or using the button in the Ribbon. You see will the color of the Attribute changing to reflect this change.

Lesson 3 · Instances

In this lesson, you will learn how to create and specify Instances in the TestCase Design section of Tricentis Tosca.

Now that we have covered Attributes, the different features or characteristics of the application we are testing, we need to define the variations of those Attributes. In Tosca, these are called **Instances**.

Instances can be added to a particular Attribute by selecting the option to add an Instance from the Ribbon, choosing "Add Instance" from the Mini-Toolbar, or by using the shortcut "Ctrl+N, Ctrl+I".

For each Instance we can define different properties which help to identify how and when a particular Instance should be used and in which TestCase combinations. Instances cannot have sub-Instances.

The first property we can define is the "Character" of the Instance. The Character property tells us what kind of Instance we are dealing with – Valid, Invalid or "Straight Through".

Valid is the default character Value and is a white Instance icon with an "I" inside and is outlined in red. A Valid Instance means that, when this Value is applied, we do not expect an error to occur in the process.

An **Invalid** Instance is the same color as the Valid Instance but has an "X" instead of an "I". An Invalid Instance indicates data that, when applied, should result in an error. This does not mean a failed TestCase, but rather that we expect an error or error message in the system. A simple example would be a TestCase which is meant to check the log in process of an application. An invalid TestCase would use invalid data, so an invalid password, and should verify that the user is not logged in and that the appropriate error message appears.

A **Straight Through** is a special Instance which can be defined according to certain criteria. It should be a high-risk Instance, easy to implement, and have the fewest dependencies (meaning it can be combined with as many Instances from other Attributes as possible). This can often be referred to as the "happy path" Value and is usually part of the smoke test.

Defining which character to use for an Instance is simple – you can either right click on the Instance and select "Toggle Character" once for Invalid and twice for Straight Through or use the shortcut F7.

The second property we can define is the "**Position**" of the Instance. This means whether the Value is an Inner Value or a Boundary Value.

Boundaries are the Values at the end of a particular range. So I may have a range, age for example, which should only have Values between and including 1 and 99. That means the Values around 1 and around 99 need to be tested because incorrect operators may have been set at the Boundaries (for example the code may say >1 rather than >=1). So we could test 0 and 1 as lower Boundaries and 99 and 100 as upper Boundaries.

Boundary Values are important to consider, as they constitute a high probability of error. and this needs to be tested. These are one-time errors as long as the range on either side of the Boundaries do not change from one release to the next. Therefore, Boundary Values are often used in one-time tests. Defining whether an Instance is a Boundary Value can be done by pressing F8, or selecting "Toggle Position" from the context

menu. Inner Values are the default and look like normal Instances. Boundary Values have small red dashes around the "I" in the Instance icon. They can be either Valid or Invalid.

Inner Values are all other Values within the range that do not fall immediately around the Boundaries, in other words, all numbers between and including 2 to 98. Do we need to test each of these individually? Probably not. And remember, one of the goals of TestCase Design is to maximize risk coverage with the least number of TestCases. We can instead group these numbers into an "**Equivalence Class**". This would mean, we would create one Instance which represents all numbers between 2-98. We could then, for example, randomize the number being used using a Tosca RND expression and include this Instance in our regression tests.

It is not necessary to group Values into equivalence classes for all Attributes, of course. If you have an Attribute with specific values, not ranges, then you can define the specific Values to be used.

Lesson 4 · Combinatorial Methods

In this lesson, you will learn how to manually combine Attribute Instances and some of the possible combinatorial methods.

Up until this point in your TestCase Design, you have created individual Attributes and the Instances which reflect the variations of these Attributes. Using combinatorics, we will begin to combine these Attributes to reflect the actual data combinations needed to test the Requirement.

There are several methods for combining the data. The methods supported by Tosca include:

- All combinations
- Orthogonal
- Pairwise
- And Linear Expansion.

All combinations is a method which ensures that all possible Instance combinations are taken into account for testing. This method of combination results in the highest number of TestCases. Let's look at an example. Here you can see 3 Attributes with 3 Instances each. All possible combinations of this data would result in 27 possible data combinations.

The **Orthogonal** method will result in the lowest possible number of combinations. Each Instance Value must be used once but that is the only Requirement for the combinations. The highest number of Instances on one of the Attributes being combined will be the number of combinations. Using our last example, 3 Attributes with 3 Instances each will result in only 3 combinations. If one of the Attributes has more Instances than another, then the Instance values of the Attribute with fewer Instances will be repeated until each of the Instances with a higher quantity are used.

The **Pairwise** method combines each pair of possible Value combination for two Attributes. We will not focus on this method.

Finally, **Linear Expansion** is the method recommended by Tricentis. This method defines a "StraightThrough" or happy path. The first combination will use all StraightThrough instances. This could be considered a smoke test. As mentioned in previous lessons, the StraightThrough should be the TestCase with the highest risk if it were to fail because it is frequently used and has the fewest dependencies. All subsequent combinations switch out one StraightThrough Instance for a non-StraightThrough Instance. All non- StraightThrough Instances are combined individually in this way until all Instances are tested at least once.

This methodology provides for a clear test focus in each combination, because once the StraightThrough is tested, each of the combinations created using this methodology will have only one Instance Value which is different from the StraightThrough Instances.

Here you can see the difference between the methods as additional Instances are added for Attributes. All combinations grows quite quickly and results in the highest number of test cases. With a long and complex TestSheet, this could result in a very high number of TestCases which means longer time in execution. This may not be the most efficient testing method. Orthogonal has the least number of TestCases because it generates the minimum number of TestCases. This will result in the shortest possible execution time, but the TestCases have no clear focus in case of failure and will result in a longer time debugging. Finally, Linear

Expansion results in a moderate number of TestCases. For each additional Instance, an additional TestCase is created. This is because each Instance has one TestCase in which it is the primary test focus.

When would I use each method? Well this largely depends on your project, its requirements, the level of specification and granularity of those requirements, as well as any legal or regulatory specifications. For example, in some industries you may be required to test all combinations for certain requirements. Tricentis recommends Linear Expansion where possible or practical, but of course this depends on your project.

Now that we have defined the combinatorial methods, let's learn how to build them in Tosca. We will be using Linear Expansion. These combinations can be generated automatically and manually. For the time being, you will use only manual combination. In later lessons, you will learn how to combine automatically.

In Tosca, these combinations are called Instances and are represented in columns. These Instances can be created on the TestSheet level. On the TestSheet itself, you would add Instances. These Instances have the same properties as those we have created so far, meaning you can define the character and position. With the TestSheet in focus, you can see all generated Instance columns. Next to each Attribute on the TestSheet, you can then select the relevant Instance from a drop-down box. When using Linear Expansion, the first generated Instance will be the StraightThrough and will have all StraightThrough Instances. In each subsequent Instance, you should switch out one StraightThrough for a non- StraightThrough instance. In a real project, remember that some adjusting may be necessary, as this methodology does not take into account dependencies or relationships between Values and Attributes in a business context.

These Instances now represent the test cases which are needed to test the associated Requirement and the Values can be used in the TestCase section using the Template concept.

In addition to combination on the TestSheet level, Instances can also be combined on Attribute level. This means creating Instance combinations on lower levels. For example, one of my Attributes may be credit card information. Below this Attribute, I have details concerning this information. If I focus on the Attribute, I can create credit card Instances. Each one will contain the information which I select. Then, on the TestSheet level, I will only need to select the name of the credit card information Instance, and the associated Values for that Instance will be automatically populated. This serves to make combining easier and more accurate. You can see whether an Attribute has been combined at Attribute level because the Values are highlighted in light yellow at these levels.

You can organize your Instances by right clicking on the TestSheet or clicking on "Optimize Instances" from the Ribbon and selecting "Arrange Instances". This will group them by position and character – first the StraightThrough Instance, followed by Valid Inner Instances, Invalid Instances, Boundary Instances, and finally Invalid Boundary Instances. You can also merge Instance duplicates (both in the Ribbon and the Context Menu) which will find all duplicate Instances – in other words, Instances which are identical) and remove all but one.

You can see how many Instances are on the TestSheet level by hovering your mouse over the TestSheet Icon in the working pane.

Remember – In order to see the Instance columns on a certain level, you must be focused on that particular level (whether that be TestSheet level or Attribute level). Additionally, you may come across the problem in the TestCase Design section that you create Instances but then cannot see them. Try pressing F9, which is

the shortcut for hiding and revealing Instances in the TestCase Design section. This exists because with large, complex TestSheets it is sometimes easier to get an overall view of the data by hiding the Instance rows.

In this lesson, you will learn how to finish filling in your TestSheet with the Values to be used in the physical TestCases.

In the previous lesson, we filled in Values in the process portion of the TestCases, as all of these were relevant to determine the combinations for creating Instances or TestCases on the TestSheet level. We will still need to populate the TestSheet with more physical Values which can be used as Values in the TestCases we build in the blue section of Tricentis Tosca. These Values can be selected using the same method as we have previously seen – that is selecting the appropriate Instances Value for the Attribute. A perfect example of this was the credit card information. We chose a specific card number, expiry data, and card type. These we will be able to use later on when we actually build a TestCase Template by linking the Attribute Value to the TestStepValue.

We can add new Attributes and Instances now to the TestSheet which will not affect the number of combinations we already have. We can also start to fill in the non-business relevant and result Attribute Values – in our case that would be Administration and Verification. This means selecting the desired Instances.

At first glance, this looks like a tedious job, because it means adding each value manually 1 by 1. However, there are ways to help speed up the process. First, we have the option to fill all empty Values of a particular row. Let's look at the Administration Attribute. The Test Designer for all TestCases was Max Methodology. If we right click on the Attribute itself, the option to "**Fill Empty Values**" appears in the Context Menu. Select the option "Specific" and you can define the specific Value you want to add. When there are Instances defined already, you will simply select the desired Instance from the drop down box and all cells will be populated.

What if we want to fill empty Values, but not all Instances or TestCases should have the same value? Let's look at the test stage Attribute. We want the StraightThrough to be identified as a smoke test, the Inner Valid Instances to be defined as regression tests, and all Boundaries to be defined as one-time tests. We can actually filter for specific Instances or TestCases so that we can fill empty Values only for the filtered Instances. To do this, we would ensure that the Filter Column is visible using the Column Chooser. Then, on the level on which I want to filter, I will select the specifications. In this case, I want to filter the Instances on the TestSheet, I will filter on the TestSheet level. Here you will see a drop-down arrow and all possible filters are visible. By default, ALL is selected, but I can deselect this and choose individual Instances (for example the StraightThrough) or choose by Character and Position – Inner, Valid, or Boundary Instances. With this filter set, I can then enter my Values. I can then select ALL again from the filter or right click and select "Reset Instances Filter" from the Context Menu.

Notice that the row with the filter set becomes blue as well as the TestSheet itself. This is a quick way to verify whether you have any filters set on lower levels that you may have forgotten about.

There is also a new way of adding values without adding additional Instances to the TestSheet. These are called **Value Attributes**. They are not a separate Tosca object, but rather are simply Attributes without any Instances or sub-Attributes defined. These can be added wherever necessary, for example we will add a line for the user name under the type of user Attribute. Rather than adding instances and then selecting them, we will enter the Values themselves in the cells in the Value Attribute row on the TestSheet. In our case, actual email addresses. A very important note when entering Values into Value Attributes: they must be entered in

at the lowest level on which the Values were combined. So, with this example – we would enter the Value on the Type of User level. This is because the Value will be associated with the "Registered" user Instance. Everywhere that the registered user is selected as the type of user Instance, the email@company.com email address will be used. Once entered on the lower level, I can see this populating by then focusing again on the TestSheet. If I were to enter the email address on the TestSheet level and ignore the lower level combinations, it will instead think that I want to create a new type of user that is different from the registered user and this then affects my combinations.

Lesson 6 · Classes

In this lesson, you will learn how to create, modify, and use Classes in Tricentis Tosca.

A **Class** is a Tosca object which allows you to reuse an Attribute multiple times. This is similar to the Library concept in the TestCase section. You can define a Class, which would contain Attributes and, if applicable, their Instances and combinations, and reuse it as many times as necessary in different TestSheets. Think of a customer Attribute, for example. In a project where the application being tested is used by customers, there will be many TestSheets in a project which will require customer data. Rather than recreating this customer Attribute structure with its relevant Instances from scratch in every TestSheet, we can create a Class up front, and then reuse it in what is called a **Class Reference.** Using Classes will save time and effort reducing redundancies in the project. Additionally, once a Class is updated or changed, so are all of the references to this Class. This means maintenance is centralized and maintenance time is reduced.

Let's look at how this works in Tosca.

Classes can be created in 4 ways:

- First, you can create a Class by creating it manually within a TestCase Design Folder. Within this Class you can create Attributes and Instance combinations as required
- Second, you can drag & drop an existing Attribute or Attribute structure onto a TestCase Design Folder. A new Class is created and the original Attribute is converted to a ClassReference.
- Third, you can right click on a TestSheet and choose the option extract Class. This will take the entire TestSheet structure and convert it into a Class, and a Class Reference is created which contains all Attributes and Instances that were in that TestSheet.
- Fourth and finally, you can create a Class by dragging and dropping a Module onto a TestCase Design Folder. The Class adapts the name of the Module, the Module Attributes become TestCase Design Attributes, and the ValueRanges of the ModuleAttributes become Instances for each of their corresponding TestCase Design Attributes. In this course, we will only focus on the first two methods.

When you create a Class manually, it is similar to setting up a Test Sheet. You will right click on the Folder, select "Create Class", and then add Attributes, Instances, and combinations as needed.

As mentioned before, if you drag and drop an existing Attribute structure which contains Instances on the top level, meaning that there are combinations on that Attribute level, onto a TestCase Design Folder, a Class will be created and the existing Attribute will be converted into a Class Reference. If you drag and drop an existing Attribute structure which does not contain Instances or combinations at the highest level, you will receive two options: "Create Class from Attribute" and "Create Structure from Attribute".

Create Class from Attribute will copy the Attribute and Instance structure as well as automatically create **Instance** combinations on the basis of the existing combinations in the **TestSheet.**

Create Structure from Attribute will copy the Attribute and Instance structure from the original Attribute but will not create any combinations.

When you drag and drop an existing Attribute to create a Class, notice that the Attribute icon changes – a white arrow appears to indicate that this is now a Class Reference.

Remember: As with Libraries, you must select “Resolve Class Reference” on the reference itself if you want to make changes to or remove a Class without changes the Values in the reference.

Once the Classes are created using any of the above methods, you can then modify them as you would a TestSheet. That means you can add or delete Attributes, create or remove Instances and modify the Instance combinations. You can also assign characters and positions to the Instances.

The entire Class can now be used in multiple TestSheets as Class References and there is no limit to the number of Class References to a particular Class. This means you can drag and drop the class into a TestSheet, and the structure will be brought into the TestSheet, as well as the combinations you created in the Class, if any. When you have added new combinations to a Class, you can go back to the Class Reference in a TestSheet, and you will see that the newly created Instances are now listed.

In our scenario, you will be adding additional 3 new product Instances to the product Class. This will allow us to combine multiple products in one order, as well as reuse the product Class in later TestSheets. Once you have completed your product Class, you can now add a second product into your TestSheet and modify the TestSheet to reflect the combination of the two products. This means adding three new Instances to the TestCase to reflect the 3 new combinations added to the Class. Remember, when creating these new Instance combinations, you should consider that we have combined using the Linear Expansion methodology, so we should complete our list of Instances with this methodology in mind.

In this case, we use every Instance we created. However, not every Instance will be required in every Class Reference in a real project, as they will be reused in different contexts. This is not a problem, you need only use the relevant Instances for the context. Unused Instances will be **bolded** in the TestSheet.

It is also possible that you will not need all Attributes in every Class Reference. In this case, you can actually mark particular Attributes in a Class Reference as irrelevant for this particular TestSheet, and they can then be hidden from view. To do this, you will need to select the column “Relevant” from the Column Chooser. Each Attribute within a Class Reference will have a checkbox next to it. When the box is ticked, which is the default value, the Attribute is considered relevant. When you untick the box, the Attribute will be grayed out. You can toggle between hiding and revealing these irrelevant Attributes by using the shortcut “F11”.

Lesson 7 · Link To Requirements

In this lesson you will learn how to link your TestSheet to your Requirements.

Linking the TestCase Design Instances to the Requirements allows you to have an overview at this stage of how much of your project has been put into the planning process.

To link the Instances to the Requirements is simple: you drag and drop the Instances from the TestSheet onto their respective Requirements. If all of the Instances should be linked, you can drag and drop the entire TestSheet, but if only a certain set belong in a particular sprint, for example, you can select individual Instances to drag.

The linked Instances then become TestCase-Substitute links. Assuming you have properly defined your TestSheet, these TestCase Substitute links now demonstrate how many TestCases will later need to be built in the TestCases section to cover this Requirement. The rule of thumb for TestSheet creation is 1 TestSheet per Requirement. That TestSheet should contain all TestCases for that Requirement. Of course, if you work in sprints, this may differ in that you would have one TestSheet which will contain TestCases that will be created over multiple sprints.

When you create the TestCases, you will link them to the same Requirements and the substitute links will become TestCase links. Finally, when you create ExecutionLists, you will link the ExecutionList to the whole RequirementSet, and this will show you the progress of your entire project – in other words, you will knows which TestCases have been defined, which Requirements are being worked on and how far along you are, and if any executions have run, which have passed or failed.

This will be your project dashboard within the Tosca Commander.

Lesson 8 · Integration of New Attributes

In this lesson, you will learn how to integrate a new feature of your SUT into your existing TestSheet.

The addition of new features into an application is a common occurrence. When this new feature affects your TestSheets, you will find yourself in a situation in which you need to integrate this new feature and the corresponding Instances or data into your TestSheet.

To add the new Attribute, you will simply create an Attribute wherever it fits in logically in the TestSheet. This means it can be added into an already existing set of Attributes with existing combinations. In our case, an additional option has been added to our WebShop that allows the user to choose which type of shipping they would like. As this will obviously affect shipping costs, we will need to add this new "Shipping Method" Attribute to our TestSheet. Once the Attribute is added, you will need to create the Instances related to the Attribute. This means adding the Instances for the various options like ground shipping, next day air, second day air and N/A for the Instances which do not require or allow shipping.

When you have added the Attribute and its Instances, you must remember to complete the combinations on all levels from the lowest up to the TestSheet level. This may mean simply selecting the Instance Value for the new Attribute in the already existing combinations, or, if the addition of the Attribute requires entirely new combinations to be created according to the combinational method you have chosen (for example Linear Expansion), you will need to add these new Instances on the respective level and populate the Values. At each level you will need to ensure that there are no missing Values. These steps are valid for both the lower levels and the TestSheet level.

We have combined on the Checkout level and the TestSheet level. Because we have a new set of radio buttons with different shipping method options, we will actually need to test each of these methods. The "Standard" method or StraightThrough will be ground shipping. We will also need to add next day air, second day air and a fourth "not applicable" Instance which can be used when shipping methods are not applicable. On the Checkout level, we will need to add new Values to the existing Instances as well as check whether we are missing any Instances. In this case, we can add the Ground Instance (the StraightThrough) to the Standard shipping method. For In-Store Pickup and No Shipping, there is no shipping so we can add the Value N/A. We are still missing the Instances Next Day Air and Second Day Air. These will now be added and populated with the StraightThrough Instances everywhere except for the "Shipping Method" Attribute which will take the appropriate Values – Next Day Air and Second Day Air.

We would now check for any other levels that have been combined where this would be relevant. In our case, only the TestSheet level needs to be modified. We have already selected the correct checkout Instance for each of the existing 9 Instances. Again though we need to add 2 new Instances to reflect the new shipping methods. They should have all StraightThrough Values with the exception of the checkout Attribute where we will select Next Day air and Second Day Air respectively.

When you have created new Instances, you will need to manually link these new Instances to the Requirements which they are testing to create TestCase Substitute links.

Lesson 9 · Best practices

In this lesson, you will learn best practices for setting up TestSheets in the TestCase Design section of Tricentis Tosca.

When you begin to create multiple TestSheets in Tosca, you should consider reusability of data from TestSheets. This would mean converting any Attribute structures from TestSheets that will be used again into Classes. These Classes can then be amended, either immediately or during the course of the project, to include all necessary data variations.

We recommend that you use the core Attribute structure of Administration (with business relevance "no"), Precondition and Process (with business relevance "yes") and Verification (with business relevance "result"). Each of these Attributes can then be populated with sub-Attributes and Instances.

To also reflect an Agile methodology, you can use Test Case Design with the "Given-When-Then" structure. This is a template intended to guide the writing of acceptance tests for user stories.

An acceptance test for a User Story should contain:

- **Given** – which will describe context and precondition
- **When** - which will indicate actions which are carried out
- **Then** - which reflects a particular set of observable consequences / outputs

An acceptance test for our WebShop example would be:

- **Given** that a customer is registered and address is available,
- **When** the customer attempts to order a product with different shipping methods and product combinations
- **Then** the checkout should correctly calculate the shipping costs and add them to the total price of the order

You can also set up the structure of your TestSheet in advance in another tool – for example notepad or Microsoft Excel. This structure can then be copied from the Clipboard into the TestSheet.

In notepad, you simply write each Attribute on its own line, and tab indentations indicate when an Attribute should be copied as a sub-Attribute of the previous one. You then copy all. To add this structure to your Clipboard, right click on the TestSheet, and select "Create Element Structure from Clipboard" from the Mini-Toolbar or from the Ribbon at the top of the page. You will see the Attributes being created.

Copying from Microsoft Excel is similar – each Attribute has its own cell in its own row. Rather than tab indentations, you will use one column to the right to indicate when an Attribute should be copied as a sub-Attribute of the above one. You will then copy all and use the Create Element Structure from Clipboard option on the TestSheet level.

Other things to keep in mind when considering readability:

- Use short and comprehensible TestCase names and explicitly mention the test focus.
- Create Instances for Attributes, which describe the test focus.
- Use characters to mark instances as StraightThrough, Valid and Invalid.
- Expand Attributes for which you require more detailed information. Use comprehensible names for Attributes and Instances. Check the whole TestSheet for readability.
- If you use Concrete or Value Attributes, put them under a logical Attribute when possible, to make it clear what the Values are used for.
- Use Classes only if they are re-used at least twice.
- Expand Attributes for which you require more detailed information. Use comprehensible names for Attributes and Instances. Check the whole TestSheet for readability.
- If you use concrete or Value Attributes, put them under a logical Attribute when possible, to make it clear what the Values are used for.

- The Attributes in the TestSheet are described in business language (instead of the more technical language of modules and TestCases)
- Each TestCase or TestCase instance should be meaningful and understandable – it should test a business functionality and should be linked to a Requirement
- Each TestSheet instance should have a clear and defined Verification
- TestCase variations should be created based on equivalence classes to provide maximum coverage with minimum redundancy.
- Attributes (if there are more than 6-8 on one level) should be structured beneath other Attributes i.e. Sub attributes
- The final TestSheet is assigned to the TestCase template so that all required TestCases are Instantiated for execution.
- Boundary Value Instances are not needed in a regression test, as functionality around the Boundaries is not likely to change from release to release.

Lesson 10 · Automatic Generation of Instances

In this lesson, you will learn how to automatically generate Instances in your TestSheets.

The goal of the TestCase Design section in Tricentis Tosca is to generate entire test data sets using defined Attributes and Instances. Creating data combinations can be done automatically in Tosca. This means that if we focus on a particular level within a TestSheet, we can instantly combine all structural elements below that particular level to create Instances using the selected combination method.

So far in this course, you have learned how to manually combine Attributes on lower levels as well as the TestSheet level to create the necessary Instances according to Linear Expansion. This can easily all be done using automatic generation.

Let's begin on a lower level Attribute within the TestSheet. To combine at this level, we must focus on the Attribute on which the Instances should be created. To generate Instances automatically on this level, you need to create an empty Instance folder. This is because the lower Attributes being combined will automatically combine in the next highest Instance Folder. To create this, you will add an Instance which creates both the Instance itself and the Instance Folder which contains it. You will then delete the Instance, but keep the Folder. Now, you are ready to combine. Remember – if you are using Linear Expansion, you MUST indicate which Instances are StraightThroughs. This is because the combination method creates first a StraightThrough Instance, and then generates all additional Instances by switching out one of the StraightThrough Values for a non-StraightThrough Value. To combine, simply select the Attributes which you want to combine, and right click, and select Generate Instances >> and choose the combinatorics method.

You will see the combinations according to the selected method being created.

You can do the same thing on the TestSheet level. The only difference on this level is that you do not need to create an Instance Folder in advance, you just need to generate the Instances by right clicking on the TestSheet itself, rather than by selecting the lower level Attributes.

When you right click on the TestSheet and select generate Instances and select the method, you will see the Instance combinations appear in columns. If there have already been any combinations created on lower levels, this will be taken into account on the highest level. Attributes which are set to business relevant "No" or "Result" will not be taken into consideration in automatic generation.

Now, what happens if we want to add a new feature and reflect this in our TestSheet? If we were to add an Attribute and then Generate Instances again, the existing combinations will not be taken into account and may potentially duplicated. To avoid this, we have the option to "complete Instances", which will only create the new combinations required to test this new Attribute, and the existing combinations will not be recombined. Like Generate Instances, this can be done both on the TestSheet and Attribute level. The process is the same – you will right click on the TestSheet to create new combinations on the TestSheet level, or the sub-Attributes which you want to combine into the parent Attribute, and select Complete Instances and choose the combination method.