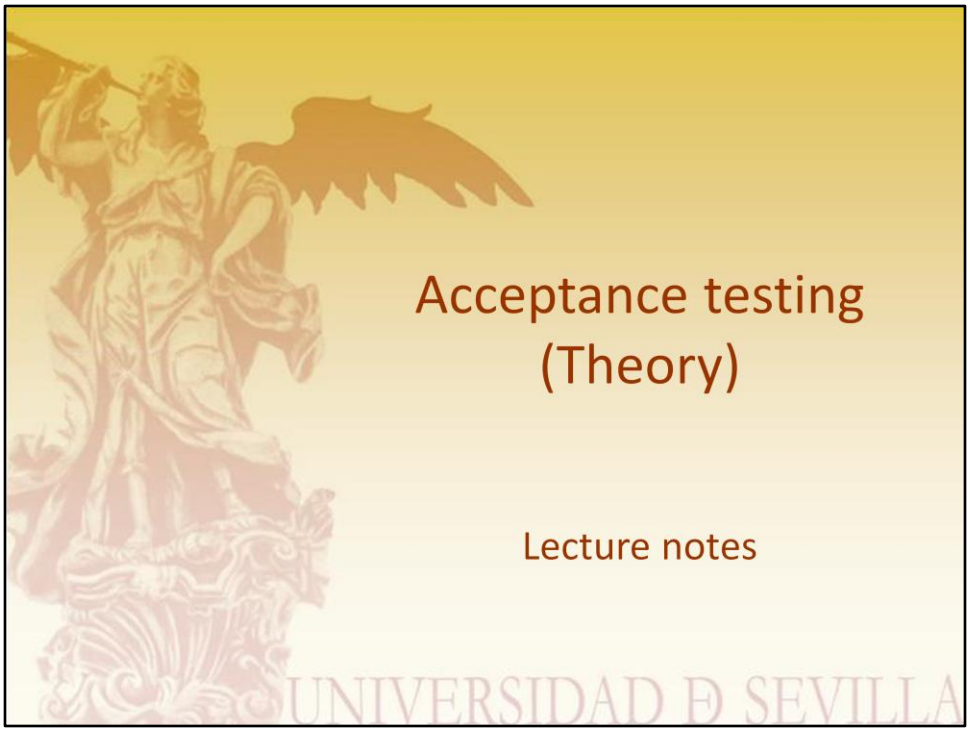


Welcome to this lecture! Today, you'll learn about acceptance testing.

--

What's acceptance testing about?

UNIVERSIDAD Ð SEVILLA

2

As usual, we start this lecture with a question: what's acceptance testing about? As usual, please, make a point of producing your own answer without casting a glance at the following slides.

This is a good definition

Acceptance testing refers to tests that are intended to check if a system satisfies your customer's expectations

UNIVERSIDAD Ð SEVILLA

3

Acceptance testing refers to tests that are intended to check if a system satisfies your customer's expectations. Acceptance tests are carried out by a tester on behalf of your customer and enable him or her to determine whether or not to accept the project. This kind of testing is very important because it helps demonstrate that the requirements are satisfied and their implementation's operating in a manner that's suited to real-world settings.

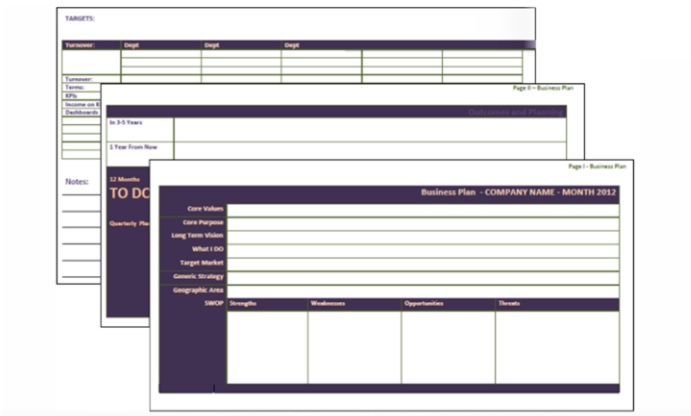
What do I have to know?

UNIVERSIDAD Ð SEVILLA

4

Let’s now report on what you have to know regarding acceptance testing.

Step 1: acceptance test templates



UNIVERSIDAD D SEVILLA

5

First, you need to command a couple of simple templates that we provide so that you can document your acceptance tests.

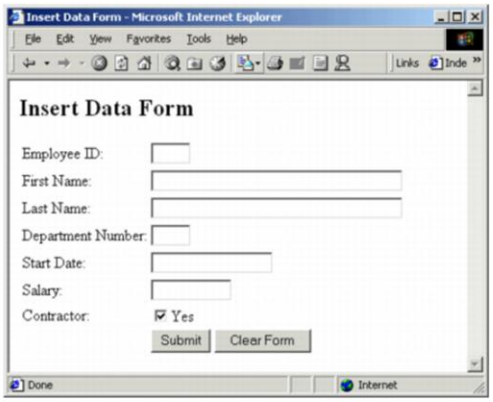
Step 2: listing tests

UNIVERSIDAD Ð SEVILLA

6

Then you need to know a little theory so that you can test your listing requirements.

Step 3: edition tests



UNIVERSIDAD Ð SEVILLA

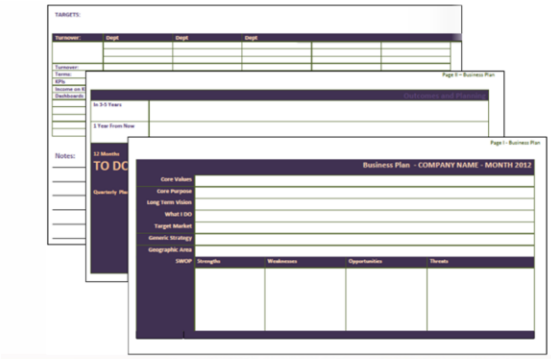
7

Then you need a little theory regarding edition requirements. That's all.

It’s then not surprising at all that this is the roadmap of this lecture.

Let’s start with the templates.

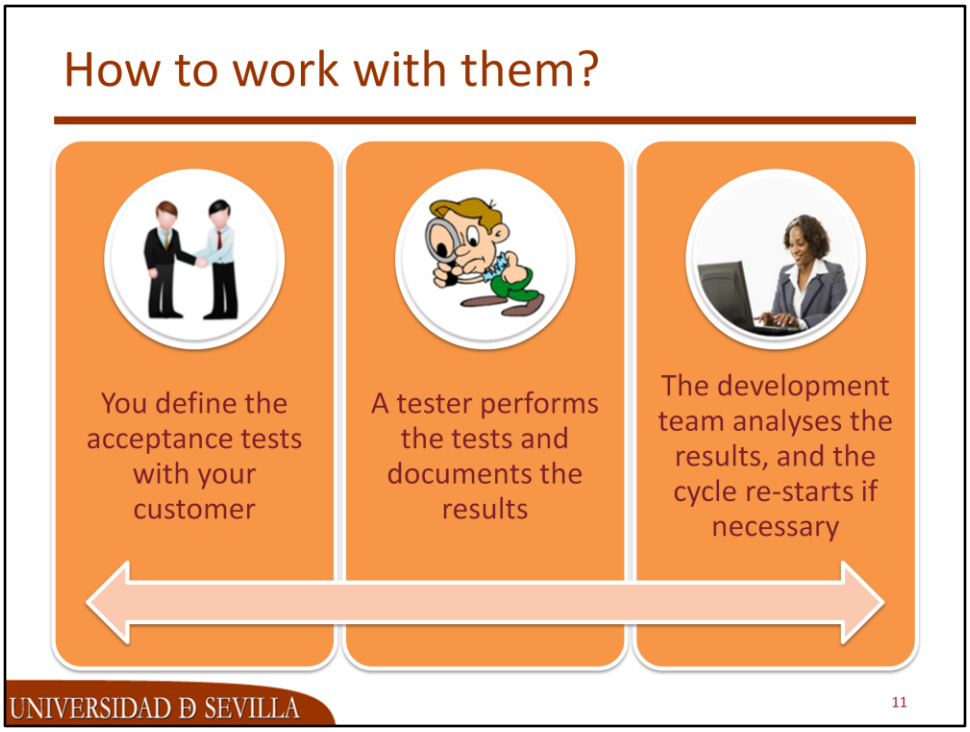
What are the templates?



They're documents that provide harnesses for your acceptance tests

UNIVERSIDAD Ð SEVILLA

They are documents that provide harnesses for your acceptance tests. Please, find them in the materials that accompany this lecture.

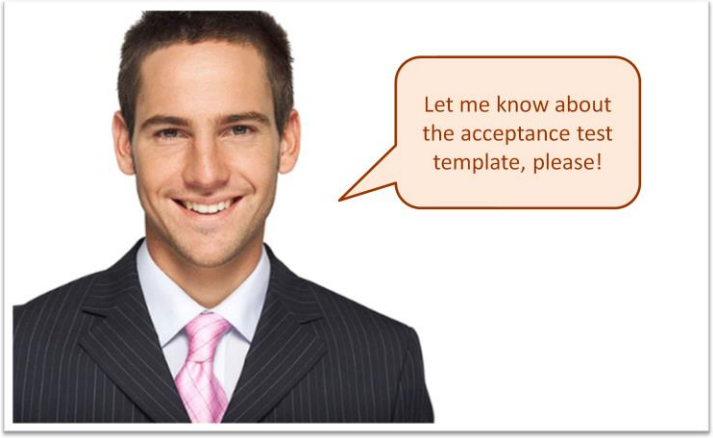


This is the typical workflow to work with the templates:

- First, you define the acceptance tests with your customer. Please, realise that these are the tests that your project must pass so that your customer pays the bill. It's then very important that you define them in co-operation with your customer.
- Then a tester performs the tests on behalf of your customer and documents the results.
- If the tester finds problems while performing the tests, then the development team must analyse and correct them. When the corrections are finished, the cycle is re-started.

Obviously, your customer won't be very happy if the cycle has to be re-started several times. You'd better correct everything on the first iteration!

Nice to hear that!



UNIVERSIDAD Ð SEVILLA

12

Ok, so it's now time to delve into the templates. Let's start with the template to document the acceptance tests themselves.

The cover

Acceptance tests
<PROJECT-TITLE> <VERSION>

Provide the title and version of the project to which this document applies.

Development team	
Identifier	*Write the identification of your development team.*
Members	*List the members of your group.*
Testing team	
Identifier	*Write the identification of your testing team.*
Members	*List the members of your testing team.*
Indexing data	
Test designers	*List the people who have designed the tests.*
Testers	*List the people who have conducted the tests.*
Notes	*Add notes if necessary.*

List person names using the following pattern: Surname, Name.

UNIVERSIDAD D SEVILLA

13

Open the template, and you'll find a cover page like the one in this slide. You only have to write the title of your project and the corresponding version. Then fill in the tables with information about your development team, the testing team, the test designers and testers. Add notes if necessary.

For every use case

Use case <CODE> <NAME>

Write the code and the name of your use case in the title.

Description

Provide a description of the use case. Note that the details must be described in your requirements elicitation document, so just provide a succinct description and use references where appropriate.

Access

Provide a description of how the tester can have access to the interfaces that implement this use case. Include screenshots if appropriate.

Tests

For every test that you wish the tester to perform, include a table with the following structure. Check your theory lecture notes to learn about some typical tests.

Test <#999>	
Description	*Describe the test that must be performed.*
Expected	*Describe what you expect from this test. Include screenshots if necessary.*
Outcome	*Describe what you've got when you performed this test. Add screenshots if necessary. Please, make sure that your description is enough for a developer to repeat what you've done, so that he or she can correct the problem.*
Notes	*Add notes if necessary.*

UNIVERSIDAD Ð SEVILLA

14

Now, for every use case, you must provide the following information:

- Description: it's a description of the use case that's going to be tested.
- Access: it's a description of how the tester can have access to this use case. Include screenshots if necessary.
- Tests: for every test that you wish the tester to perform, include a table with the following information: a description of the test, the expected results (including screenshots if necessary), the actual outcome, and notes, if necessary. Note that if the actual outcome is the expected result, then there's little more you can say; but if the actual outcome deviates from the expected result, the tester must document what's wrong as thoroughly as possible so that a developer can correct the problem.

Obviously, the document that you pass onto your testers must have the outcome field empty in every test; it's the duty of the testers to fill in these fields and return the document to you when they are done.

Additional tests

Additional tests

If necessary, design additional tests that cannot be easily associated with a particular use case. Describe them in this section.

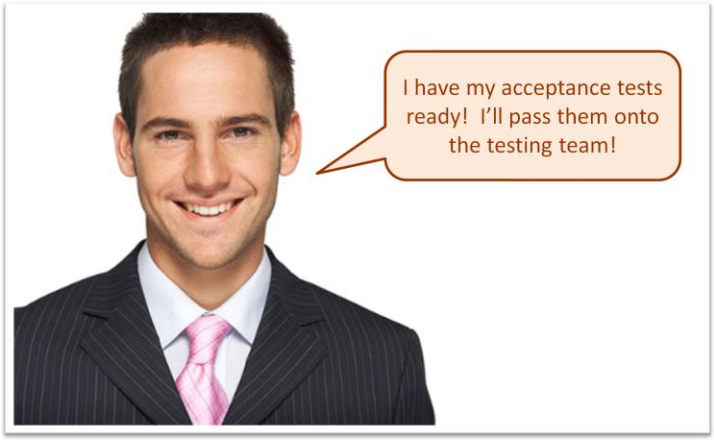
Test <#999>	
Description	*Describe the test that must be performed.*
Expected	*Describe what you expect from this test. Include screenshots if necessary.*
Outcome	*Describe what you've got when you performed this test. Add screenshots if necessary. Please, make sure that your description is enough for a developer to repeat what you've done, so that he or she can correct the problem.*
Notes	*Add notes if necessary.*

UNIVERSIDAD Ð SEVILLA

15

Finally, there's a section called "Additional tests". If necessary, please design additional tests that can't be easily associated with a particular use case and describe them in this section. Leave this section intentionally blank if you can't think of any such additional tests.

Nice to hear that!



UNIVERSIDAD Ð SEVILLA

16

When you have your acceptance tests ready, it's time to pass them onto your testing team.

Your testers – 08:00 h

UNIVERSIDAD D SEVILLA

17

They'll typically feel excited about finding bugs… that's when they start their working day.

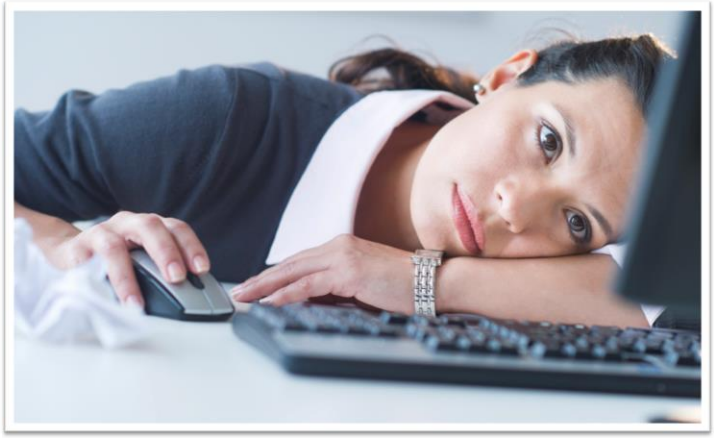
Your testers – 12:00 h

UNIVERSIDAD Ð SEVILLA

18

By the afternoon… they are very likely to feel a bit tired.

Your testers – 14:00 h



UNIVERSIDAD Ð SEVILLA

19

By the late afternoon, they won't be paying attention to every detail.

Your testers – 17:00 h

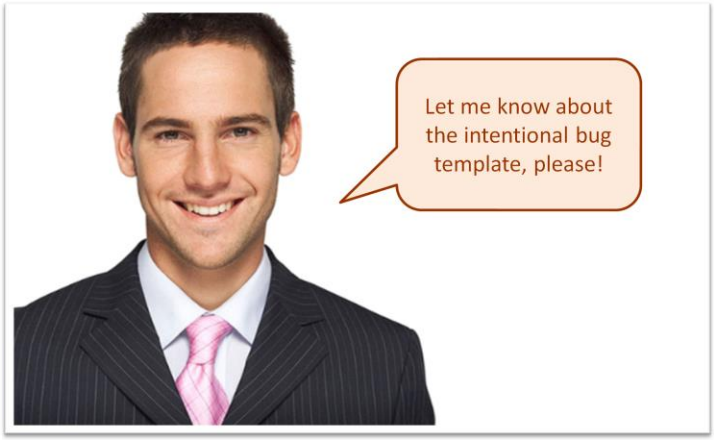
UNIVERSIDAD Ð SEVILLA

20

By the evening, they'll be so tired that they will not easily focus on bugs.

Note that testing is difficult. After several hours carrying out acceptance tests, a typical tester gets tired, bored, and usually wishes to get rid of this task. It's strongly recommend that you should include intentional bugs in the project that you release to them because this typically helps them stay focused and inquisitive. Unless your testers know that there are intentional bugs and that they have to find them, they'll soon get distracted and won't pay attention to the acceptance tests.

Nice to hear that!



UNIVERSIDAD Ð SEVILLA

22

So, it’s time to learn a bit about the report to describe the intentional bugs.

The cover

Intentional bugs
<PROJECT-TITLE> <VERSION>

Provide the title and version of the project to which this document applies.

Development team	
Identifier	*Write the identification of your development team.*
Members	*List the members of your group.*
Testing team	
Identifier	*Write the identification of your testing team.*
Members	*List the members of your testing team.*
Indexing data	
Designers	*List the people who have designed the tests.*
Testers	*List the people who have conducted the tests.*
Notes	*Add notes if necessary.*

List person names using the following pattern: Surname, Name.

Effectiveness
Document the effectiveness of your testers. Compute it as the ratio of intentional bugs that they've found to the total number of intentional bugs that you injected.

UNIVERSIDAD Ð SEVILLA

23

The cover's very similar to the cover of an acceptance test. It provides information on your development team, your testing team, and some indexing data. The difference's that there is an additional table to report on the effectiveness of your testers, which is measured as the ratio of intentional bugs that they've found to the total number of intentional bugs that you injected.

For every intentional bug

For every use case in which you have included an intentional bug, add a page with the following structure.

Bug in use case <CODE> <NAME>

Write the code and the name of your use case in the title.

Description

Provide a description of the bug that you have injected. Make it sure that the expected wrong results are made explicit, as well as the correct results. Include screenshots if appropriate.

Results

Comment on whether the tester found this bug or not.

UNIVERSIDAD Ð SEVILLA

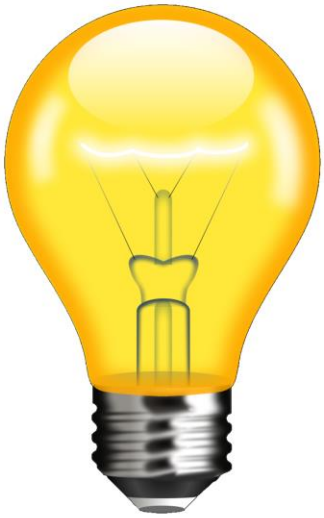
24

Now, for every intentional bug, you must provide the following information:

- Description: provide a description of the bug that you have injected. Make it sure that the expected wrong results are made explicit, as well as the correct results. Include screenshots if appropriate.
- Results: comment on whether the testers found this bug or not.

Let’s now report on a number of typical tests regarding listing requirements.

The key idea



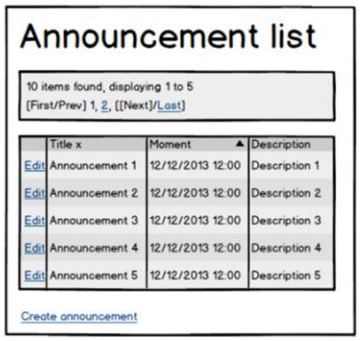
- Focus on the use cases
- For each listing one
  - Emulate normal behaviour
  - Emulate abnormal behaviour
  - Keep an eye on the following hints

UNIVERSIDAD Ð SEVILLA

26

The key idea is that your acceptance tests must be driven by your use cases. For each listing one, design some test cases in which you emulate normal behaviour (there must not be any problems and the system must be expected to work well), a few more in which you emulate abnormal behaviour (there must be some problems and the system must catch them all), and keep an eye on the hints that we provide in the following slides.

Test #1



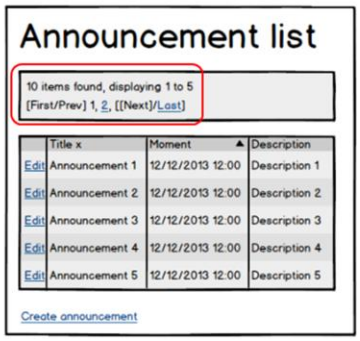
Are you displaying the right information?

UNIVERSIDAD Ð SEVILLA

27

The first and foremost important test is to make sure that the listing shows the information that it's expected to show. Review the information with your customer and make sure that it's correctly displayed.

Test #2

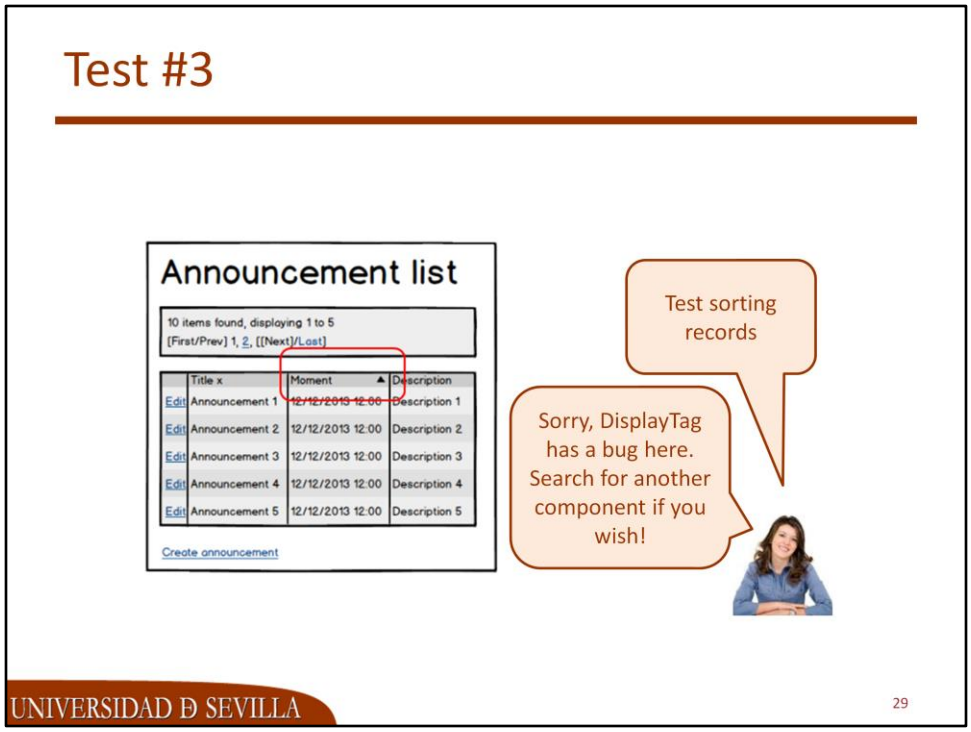


Test the pagination links

UNIVERSIDAD Ð SEVILLA

28

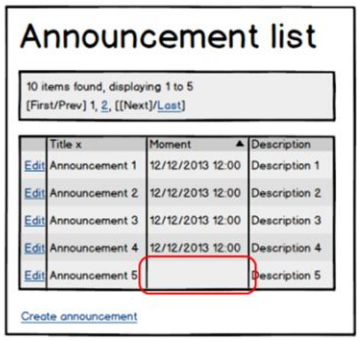
Then, test that the pagination links work correctly. You must have loaded enough data to have at least a couple of pages in every listing. Test every pagination link, that is: first page, previous page, next page, and last page, as well as the direct access links.



Test sorting records as well. Test that your customer can sort the columns that he or she requires to be sortable and that the sorting algorithm works well in every situation.

NOTE: please, note that we use a component called “DisplayTag” to implement listings. Unfortunately, it doesn’t sort dates correctly. We’re sorry, but this is real technology and real technology is far from perfect.

Test #4

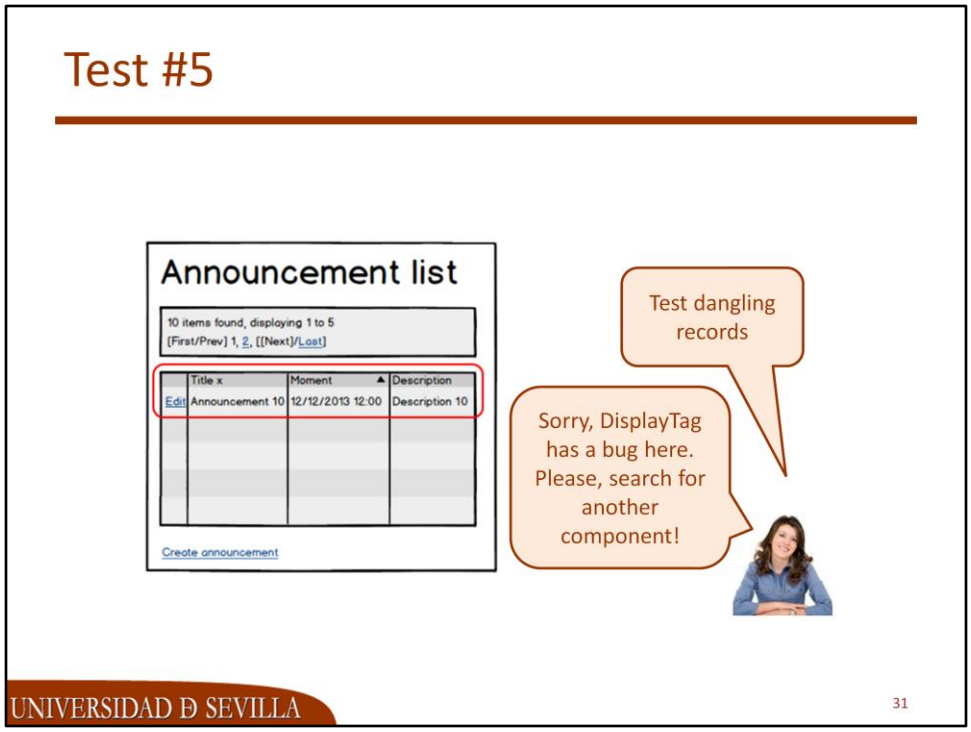


Test that optional attributes work well (not the case, but…)

UNIVERSIDAD Ð SEVILLA

30

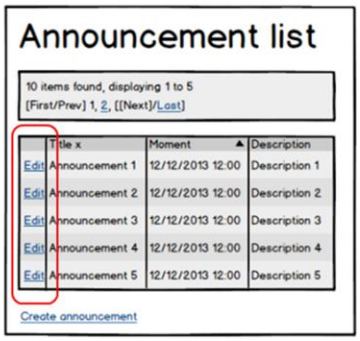
Test also that optional attributes are displayed as expected. Unfortunately, there's not a single example of an optional attribute in the Acme Certification project, so we've cheated a little in this slide. For a while, assume that there can be announcements that are expected to be held in future, but don't have a moment yet. In such cases, the listing should show an appropriate message or nothing at all, but not "null" or something like that.



Test dangling records, as well. A dangling record's a single record in the last page of a listing. It should display well. Edit that record and remove it. The listing should remove the last page and work well in this case.

NOTE: unfortunately, DisplayTag has another bug; it doesn't work well in the situation that we presented before, chiefly if the "keepstatus" attribute is involved. Isn't it time to search for another listing component? That might be a good idea for your Hackathon, right?

Test #6



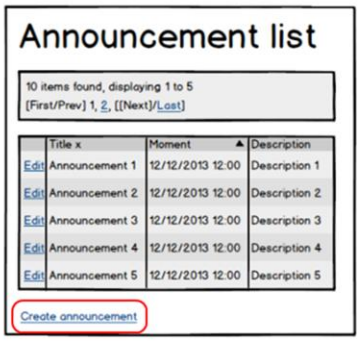
Test the edit links in your listings

UNIVERSIDAD Ð SEVILLA

32

Test that the edit link of your listings works well.

Test #7



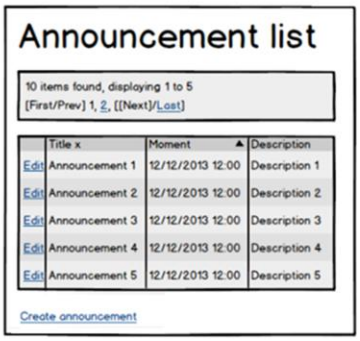
Test the create link in your listings

UNIVERSIDAD Ð SEVILLA

33

Furthermore, test that the create link works well.

Test #8



Test your data formats: dates, pictures, multi-value fields, and the like (not the case, but...)

UNIVERSIDAD Ð SEVILLA

34

Test your data formats, too. Needless to say: your system must not render any object using the default "toString" method. That results in objects that are rendered as "Customer@1234" or "Certification@5678", which simply doesn't make sense. Apart from that trivial test, please, pay special attention to the following ones:

- Dates: test that they are displayed in a format that is appropriate. Internationalise it so that they adapt automatically to the language in which the system's working, e.g., "day/month/year" in Spanish and "month/day/year" in English.
- Pictures: displaying a link to a picture is useless; render pictures properly using "img" tags. In listings, it's common that pictures are shown as thumbnails, which are tiny versions of the original ones.
- Lists: if an attribute is multi-valued, i.e., it stores collection of objects, then you must write some JSP code to iterate through the list and display it properly. By default, the components that we're using render lists within brackets and separate the objects by commas. That might work in some simple cases, e.g., a list of tags, but it's not generally a solution.

Test #9

Repeat your tests in every language for which your system is available

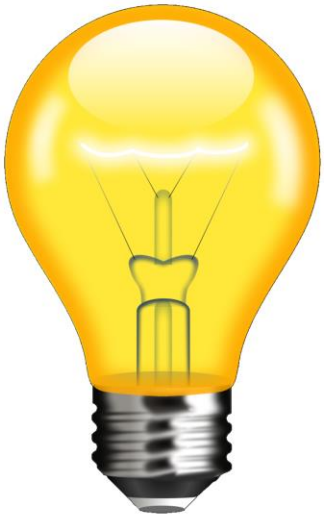
UNIVERSIDAD Ð SEVILLA

35

Finally, recall that your systems are typically expected to work in several languages. You must repeat your tests for every such language.

Let’s now report on a number of typical tests regarding edition requirements.

The key idea



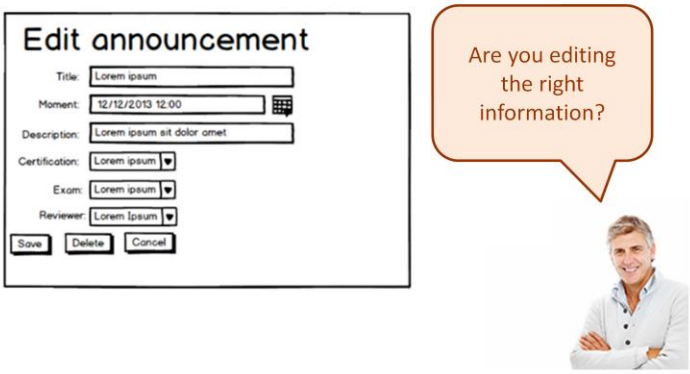
- Focus on the use cases
- For each edition one
  - Emulate normal behaviour
  - Emulate abnormal behaviour
  - Keep an eye on the following hints

UNIVERSIDAD Ð SEVILLA

37

The key idea is that your acceptance tests must be driven by your use cases. For each listing one, design some test cases in which you emulate normal behaviour (there must not be any problems and the system must be expected to work well), a few more in which you emulate abnormal behaviour (there must be some problems and the system must catch them all), and keep an eye on the hints that we provide in the following slides.

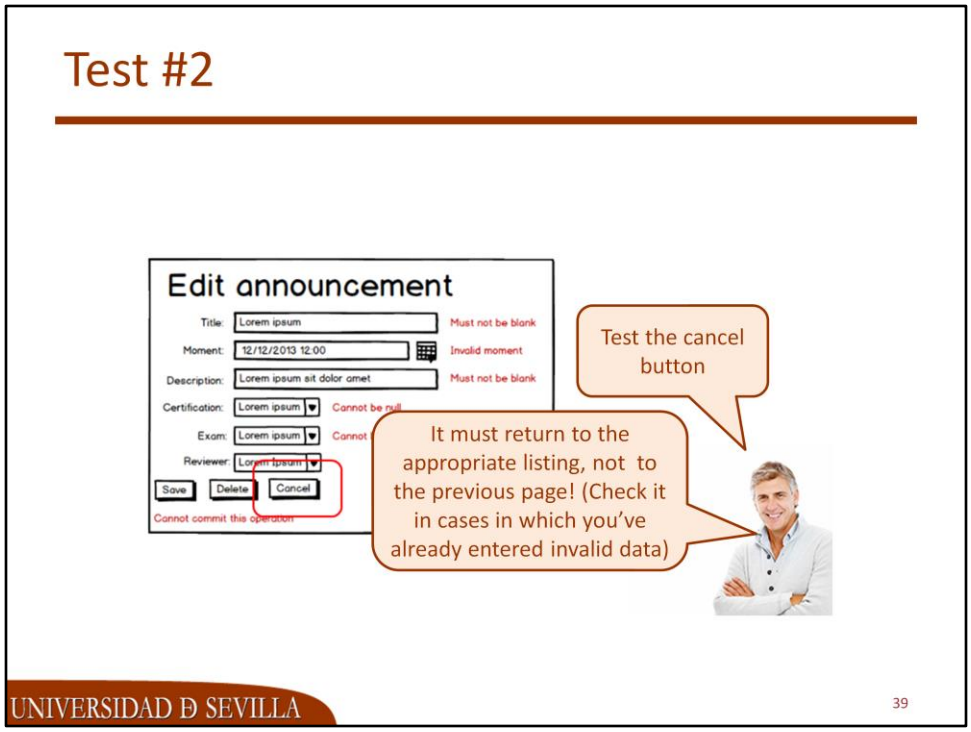
Test #1



UNIVERSIDAD Ð SEVILLA

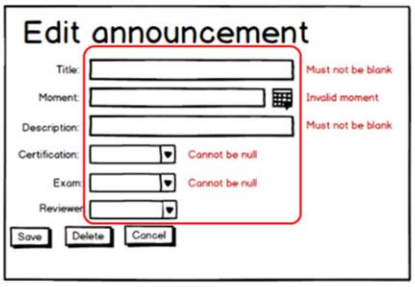
38

The first and foremost important test is to make sure that the form shows the information that it's expected to show. Review the information that is shown with your customer and make sure that it's correctly displayed.



Then you should test the cancel button and make sure that it gets you back to the appropriate listing… even in cases in which the form is filled with wrong data and then the user presses the cancel button. Please, note that you can't implement a cancel button using your browser's built-in mechanism to browse back. Think of the following scenario: (1) you display a listing, (2) you click on an edit button, (3) the system displays the edition form, (4) you enter some invalid data, (5) you submit the form, (6) the server returns a form with errors, and (7) you press the cancel button. If you implement the cancel button with "window.history.back()", it'll return to step (4) and will show the form with the invalid data. The best way to implement the cancel button is to return to the corresponding listing form. The project template provides you with a custom tag called "cancel.tag"; please, explore it!

Test #3



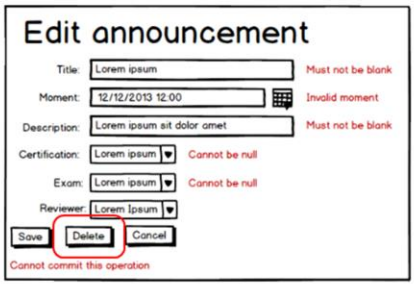
Test what happens when you leave every field empty

UNIVERSIDAD Ð SEVILLA

40

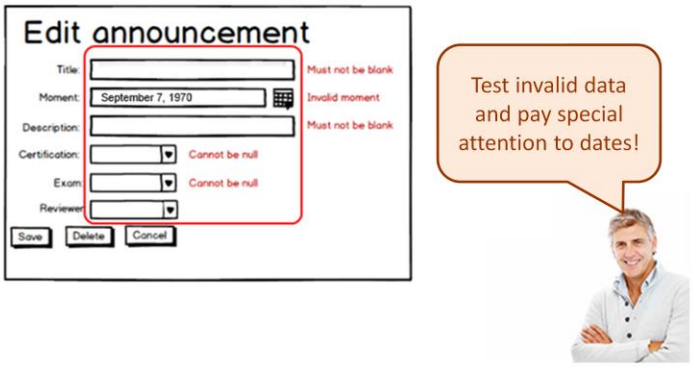
Now test what happens when you click on the “Save” button in an empty form. It should display appropriate error messages, but it shouldn’t get corrupted.

Test #4



Try deleting some data. Make a clear distinction between data that can be removed and data that cannot be removed because that would break a business rule.

Test #5



UNIVERSIDAD Ð SEVILLA

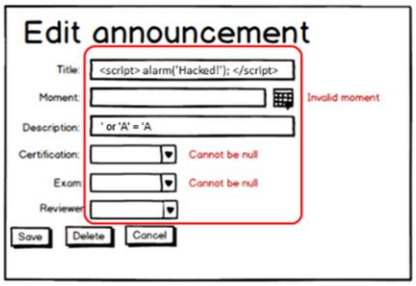
42

Now test what happens if you enter invalid data. Please, note that data may be invalid because it's not in the proper format or because it breaks some business rules. For instance, if you enter "September 7, 1970" as a date and the system expects dates to be formatted as "day/month/year" then an error would be reported. But there are other common tests beyond format issues, namely:

- Test that dates that must be in the past are actually in the past.
- Test that dates that must be in future when they're entered are actually in future.
- Test that the starting date of a period's actually before the closing date of that period.
- Test that expiration dates are actually in future.
- Test that periods are not in the past (unless this is allowed).

Please, note that the previous listing provides just a few examples; the list can be a lot longer in a real-world project.

Test #6



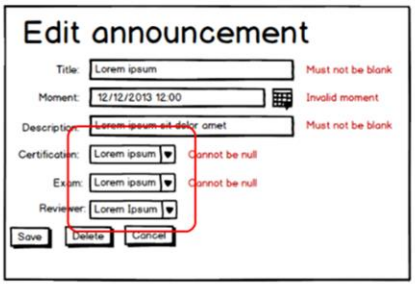
Test XSS and SQL injection

UNIVERSIDAD Ð SEVILLA

43

Now test what happens if you enter data that is intended to hack your system, that is, try both XSS and SQL injections. Test that your system behaves properly when you show that data in a listing.

Test #7



Test every possible combination of options in your dropdown lists

UNIVERSIDAD Ð SEVILLA

44

In cases in which you have dropdown lists, please, test every possible combination of options.

Test #7

```
<form id="announcement" action="announcement/admi:

        <input id="id" name="id" type="hidden" va
        <input id="version" name="version" type="

        <label for="title">
                Title:
        </label>
        <input id="title" name="title" type="text

        <br />

        <label for="moment">
                Moment:
        </label>
        <input id="moment" name="moment" type="te:
```

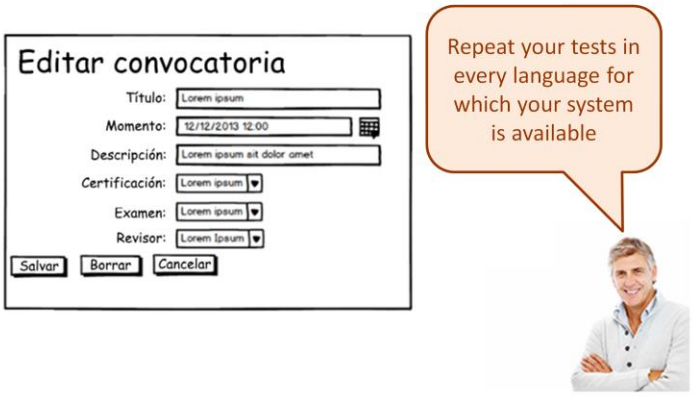
Test “hidden” data in your forms

UNIVERSIDAD Ð SEVILLA

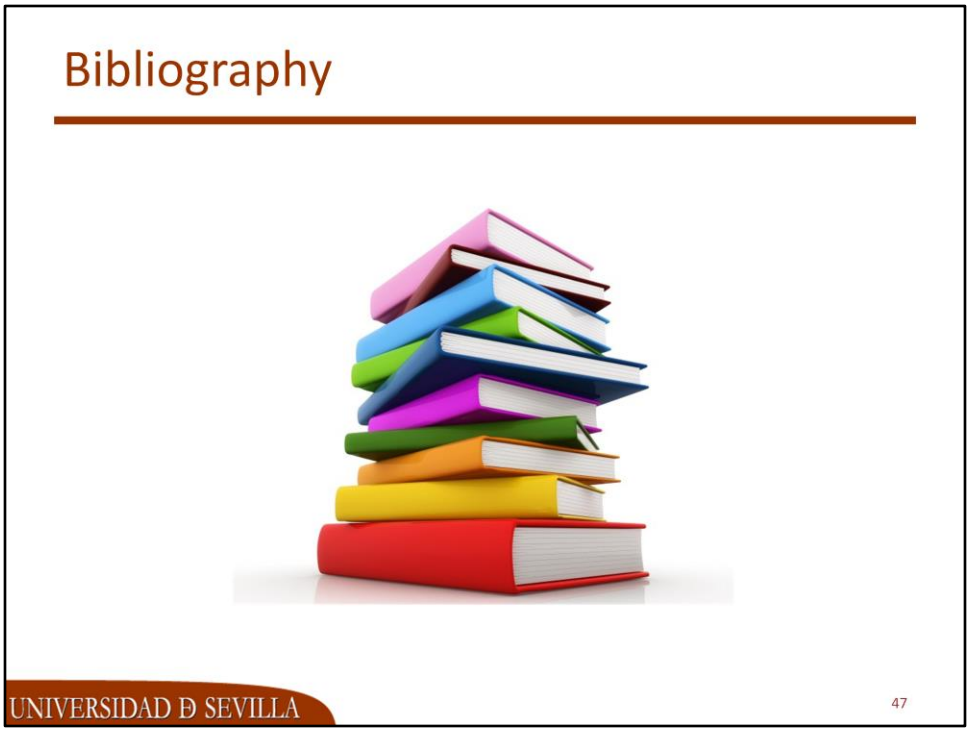
45

Finally, test your HTML forms and make sure that no “hidden” information is serialised to them. Please, recall that you must make an effort to prevent POST hacking and that you have to use form objects and pruned domain objects where appropriate.

Test #8



Finally, recall that your systems are typically expected to work in several languages. You must repeat your tests for every such language.



If you are interested in acceptance testing, we then recommend that you should take a look at the following book:

Bridging the communication gap: specification by example and agile acceptance testing
Gojko Adzic
London : Neuri, 2009

This bibliography is available in electronic format for our students at the USE's virtual library. If you don't know how to have access to the USE's virtual library, please, ask our librarians for help.

Thanks for attending this lecture!