LAB 2:
Requirements Prioritization & Release Planning
Instructions

Björn Regnell

November 19, 2014

1 Introduction

1.1 Purpose

This document provides instructions on how to run a computer lab session on requirements prioritization and release planning with computer tools, and demonstrates the complexity in finding solutions to these problems.

1.2 Prerequisites

This lab assumes that you have installed the open source tool reqT.org and that you are familiar with basic requirements modeling using reqT. It is also assumed that you have completed Lab 1 Requirements Modeling. You should also complete the preparations for Lab 2, available at `https://github.com/reqT/reqT/raw/3.0.x/doc/lab2/lab2.pdf`

You should bring a file `prio100.scala` from the Lab 2 preparations to the lab computer. The file should include a reqT scala Model with at least two Stakeholder entities, each with a Prio attribute and a set of at least 15 Req entities each with a Benefit attribute. The integer Prio values should reflect the importance of each Stakeholder and the Benefit values should include your results from a $100-method prioritization session.

2 Instructions

2.1 Prioritization

2.1.1 Ratio scale prioritization

In this section you will use reqT to calculate resulting priorities based your prepared $100-method prioritization output.

Do the following steps:

1. Load your `prio100.scala` model into the tree in the reqT ModelTreeEditor.

2. Enter the following code in the reqT text editor. Similar code is available in the menu item: Templates -> Prioritization $100 Method.

```
m =>
val shs = m.entitiesOfType(Stakeholder)
val rs = m.entitiesOfType(Req)
val prioSum = shs.map(s => m/s/Prio).sum
val benefitSum = shs.map(s =>
  s -> (m/s).collect{ case Benefit(b) => b}.sum).toMap
val normalized = rs.map(r =>
  r has Benefit(
    math.round(shs.map(s =>
      (m/s/Prio)*(m/s/r/Benefit)*100.0 / (benefitSum(s)*
          prioSum)).sum).toInt)).toModel
println("\n--- Normalized, weighted priorities:\n" +
    normalized)
val sum = normalized.collect{ case Benefit(b) => b}.sum
println("\n--- Sum: " + sum)
println(normalized)
normalized.toString.save("prio100-result.scala")
m
```

3. Apply the above function to the tree containing your prio100 model. Check the console for output and the contents of the output file `prio100-result.scala`.

4. What 5 requirements have the highest total normalized priority?

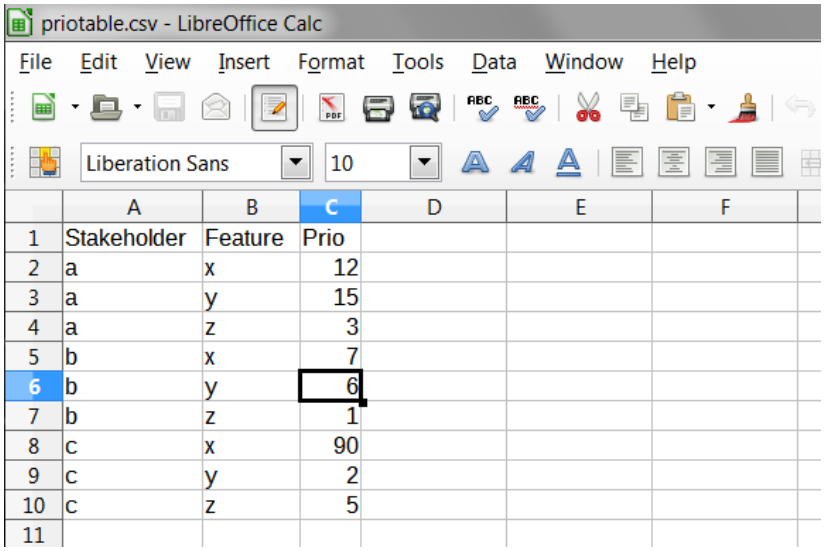
5. Change the priorities of the stakeholders. How did the normalized requirements benefit change?

6. Use a web browser to navigate to the reqT source code at GitHub and in the source file in `src/reqT` named `ModelBasicOps.scala` search for "`def entitiesOfType`" and explain what that method does.
   a) What is the result type?

   b) Can the collection include duplicate items? (Hint: search for "`lazy val entities`" and check if the method `distinct` is applied or not.)

7. Check the code ins step 2 above and match each code line with the calculations in the preparations Section 2.1.2. Explain to a friend what the code above does. You can insert `println` of interesting values to better understand what the contain, e.g. `println(benefitSum)`. Write down which `val` declarations in the above code that correspond to which sums in the formulas in the preparations?

8. Open a spread sheet program (e.g. LibreOffice Calc or MS Excel) and create the column headings Stakeholder;Feature;Prio and fill in the columns similar to this, using your own priorities:


priotable.csv - LibreOffice Calc

File Edit View Insert Format Tools Data Window Help

Liberation Sans 10

	A	B	C	D	E	F
1	Stakeholder	Feature	Prio			
2	a	x	12			
3	a	y	15			
4	a	z	3			
5	b	x	7			
6	b	y	6			
7	b	z	1			
8	c	x	90			
9	c	y	2			
10	c	z	5			
11						

9. Use Save As ... or Export and save your spread sheet in the .csv text file format, using semicolons as column separators (the default is depending on your locale). Open the file in a text editor and check that it has semicolons as column separators. Fix it if not, e.g. using your favorite editor's search-replace feature.

10. Select the Import -> From Prio Table menu item in the reqT gui, and import your spread sheet to the tree.

11. Add a Prio attribute to each stakeholder in the tree, using <Ctrl+E> and <Ctrl+R>, to model that stakeholders have different importance.

12. Calculate the normalized total priorities using the code from step 2 above. The code might be available in your undo history, check with <Ctrl+Z> in the text editor pane of the reqT gui.

13. Discuss with a friend how you could use the Import -> From Prio Table feature of reqT when you elicit priorities in your project. How would you prepare the data collection from stakeholders?

14. Extra if you are curious: Investigate the code in the reqT source file named `parse.scala` in the reqT repo at GitHub. How could you use the `load` method in `object Tab` to import a .csv file that have another character than semicolon as column separator?

2.1.2 Ordinal scale prioritization

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

2.2 Release Planning

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

2.2.1 Without coupling and precedence constraints

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

2.2.2 With coupling and precedence constraints

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.