LAB 2:
Requirements Prioritization & Release Planning
Preparations

Björn Regnell and Oskar Präntare

October 20, 2014

1 Introduction

1.1 Purpose

This document provides instructions on how to prepare for a computer lab session on requirements selection. The lab session illustrates how requirements prioritization and release planning can be supported by computer tools, and demonstrates the complexity in finding solutions to these problems. *The preparations in Section 2 should be completed before the actual lab is run.*

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.

1.3 Background

In this lab you will learn how to get started with requirements prioritization and release planning through the open source tool reqT, and reflect on how you could select requirements in your own project.

In real-world requirements engineering you are continuously faced with different types of trade-off problems. As we have limited development resources and normally would like our most important features to be ready as soon as possible, we need to make hard decisions on what to develop next and what to postpone. If we spend some time on assessing the cost and benefit of the things we have at hand, we can hopefully find a good balance in how we spend our effort wisely in

relation to the available lead time. Two main trade-off problems in requirements engineering are

- **requirements prioritization**, where (a subset of) requirements are traded off against each other according to the opinions of the stakeholders based on some criteria such as the *benefit*, e.g. with respect to the strengthening of our product's brand, or the *cost*, e.g. of lost sales in case a requirement is not implemented. There are many prioritisation methods that can be used to elicit the stakeholders' opinions. In this lab we use the $100 method and an ordinal scale ranking method, and
- **release planning**, where requirements are scheduled in time over several releases under trade-offs with respect to constraints of the available capacities of different resources, requirements priorities and requirements interdependencies.

It is also likely that you will need to do hard choices regarding how you spend your efforts in the requirements engineering process. Probably you will have to make trade-offs such as: Is it more important to do more stakeholder analysis at this point, or should we instead focus on validation of the quality requirements that we have elicited so far? Is it more urgent to reduce critical incompleteness issues or should we first improve the verifiability of our scheduled requirements?

The prioritization planning methods in this lab may also be good for prioritizing and planning the tasks of the requirements process itself, although we will exemplify the methods by using features under consideration for development.

2 Preparations

Before doing the lab session, complete all preparations in this document and bring requested items to the lab. In particular you need to make sure that you can access the text files you prepare below at your lab session computer.

2.1 Prioritization Preparations

2.1.1 Definitions

Here is one way to formalize the requirements prioritization problem:

S is a set of m stakeholders, $S = \{s_1, s_2..., s_m\}$

Q is set of n requirements $Q = \{q_1, q_2, ..., q_n\}$

$p(s_i, q_j)$ is a number representing the importance of requirement q_j assigned by stakeholder s_i

$w(s_i)$ is a number representing the importance of stakeholder s_i

$P(q_j)$ is the total priority of requirement q_j calculated by some function that maps all $p(s_i, q_j)$ and $w(s_i)$ to a single, numeric value.

A *prioritization method* defines a procedure that assigns numeric values to $p(s_i, q_j)$ for all stakeholders s_i and all requirements q_j, and to $w(s_i)$ for all stakeholders, according to some predefined priority criteria.

Before carrying out a prioritization method, a prioritization criteria needs to be defined. Examples of criteria are: market value, stakeholder benefit, risk of loss, cost of implementation and urgency of delivery. An example of a criteria applied in a pairwise comparison is: "does requirement X have a higher *benefit for stakeholder A* compared to requirement Y", and an example of a criteria used in an estimation is: "the *cost of implementing* requirement X will be €3 million".

Define a prioritization criteria. Choose a prioritization criteria relevant to your project. Define your criteria so that it is desirable to maximize the priority value.

Criteria def.: _____

Define requirements and stakeholders. Make a reqT model with 2 stakeholders and 15 requirements from your project, analogous to this template:

```
Model(
  Req("autoSave"),
  Req("exportGraph"),
  Req("exportTable"),
  Req("modelTemplates"),
  Req("releasePlanning"),
  Req("syntaxColoring"),
  Req("autoCompletion"),
  Stakeholder("modeler"),
  Stakeholder("tester"))
```

Save the model in a text file called `req.scala`

2.1.2 Methods

The **$100 method** gives each stakeholder a fictitious sum of money to "spend" on the requirements, where $p(s_i, q_j)$ is assigned to the amount of money "spent" for each requirement representing its importance according to some criteria. The combined priorities $P(q_j)$ are calculated as

$$P(q_j) = \sum_{s_i \in S} p(s_i, q_j) w(s_i) a_i$$

where the normalization constants a_i are selected so that the sum of all $P(q_i)$ is normalized to 100 units, thus

$$a_i = \frac{100}{w \sum\limits_{q_j \in Q} p(s_i, q_j)} \quad \text{where} \quad w = \sum_{s_i \in S} w(s_i)$$

Simplified $100 method. If all stakeholders are equally important, the formulas above can be simplified. Simplify $P(q_i)$ when $w(s_i) = a$ for all $s_i \in S$:

Use the $100 method. Put yourself in the shoes of each of your 2 stakeholders and use the $100 method to prioritize each of your 15 requirements according to your selected criteria.

Req	Id	Amount of dollars Stakeholder 1	Amount of dollars Stakeholder 2
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			

Transfer your priority data above into a model file of this form:

```
Model(
  Stakeholder("modeler") has (
    Prio(1),
    Req("autoSave") has Benefit(25),
    Req("exportGraph") has Benefit(10),
    Req("exportTable") has Benefit(8),
    //...
    Req("autoCompletion") has Benefit(28)),
  Stakeholder("tester") has (
    Prio(2),
    Req("autoSave") has Benefit(3),
    Req("exportGraph") has Benefit(25),
    Req("exportTable") has Benefit(14),
    //...
    Req("autoCompletion") has Benefit(2)))
```

Save the model code in a text file called `prio100.scala`

The **ordinal priority ranking method** assigns a positive integer number to each requirement q_j for each stakeholder s_i denoted $p(s_i, q_j) \in [1..n]$, where n is the total number of requirements and all $p(s_i, q_j)$ are different for each stakeholder s_i. The priority $p(s_i, q_j)$ represents an ordinal scale estimation of the preference order of the requirement q_j according to the views of stakeholder s_i. Estimations on an ordinal scale imply that the estimates only provide ordinal information and not ratio information, which means that it is not possible to tell if one priority is, say 33% or 50% of another priority, just because it has a lower ordinal value.

One way to assign ordinal priority values is to use **pairwise comparison** of the requirements and then by some algorithm (e.g. insertion sort[1]) sort the requirements in priority order, and when the sorting is ready, let $p(s_i, q_j) = n$ for the first requirement, $p(s_i, q_j) = n - 1$ for the second, etc. down to $p(s_i, q_j) = 1$ for the last requirement.

If there are n requirements, what is the total number of possible pairwise combinations, without considering order?

Try these lines in the reqT console to check your answer above:

```
reqT> def allPairs(n: Int) = (1 to n).combinations(2).toVector

reqT> allPairs(100).foreach(println)

reqT> allPairs(100).size    //size: _____
```

Consider a directed graph of comparisons, where a directed edge (a, b) represents a pair-wise comparison $a < b$. If there are n requirements nodes, what is the minimum number of comparison edges needed to connect all requirement with each other?

Try these lines in the reqT console to check your answer above:

```
reqT> def minPairs(n:Int) = (1 to n-1).map(i => (i,i+1)).toVector

reqT> minPairs(100).foreach(println)

reqT> minPairs(100).size //size: _____
```

[1] http://en.wikipedia.org/wiki/Insertion_sort

2.1.3 Compare methods

Define a cost criteria. Choose a cost criteria relevant to your project. Define your criteria so that it is desirable to minimize the priority value. A typical cost criteria is *"total effort of development including unit and system testing"*.

Criteria def.: _____

Use two methods and compare. Use first the ordinal-scale priority ranking method using insertion sort and then the ratio-scale $100 method, both with the same cost criteria applied to your 15 features.

When you do insertion sort to order your requirements in cost order on a ratio scale, you can e.g. put each requirement on a post-it note and then enact the algorithm physically, or you could enter your requirements in an editor and copy-paste them into the right order while building the final list.

Put yourself in the shoes of those stakeholders, e.g. the developers of your project, that are knowledgeable about your costs. While you carry out the prioritization, reflect on pros and cons of each method.

Req	Id	Cost (insertion sort) Ordinal scale	Cost ($100) Ratio scale
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			

Reflect on pros and cons of each method. Write down your reflections from comparing the ordinal-scale-priority-ranking-by-insertion-sort-method with the $100-method.

1. What are the pros and cons of each method?
2. How do the methods scale as the number of requirements increases?
3. Which method is easiest to carry out incrementally when a new requirement is elicited?
4. Which method may give most accurate/useful/honest estimations?
5. In which different contexts may the methods differ in usability?
6. Was the $100-method easier because of what you learned about the costs in your first round with the insertion sort method?
7. Any other reflections?

2.2 Release Planning Preparations

2.2.1 Definitions

An optimal release plan can be defined as an allocation of requirements to different releases that maximizes a benefit under some constraints, such as cost and precedence.

The definitions below represents one of many possible models of the release planning problem. Here we use a simplified model that does not take different types of resources and costs into account. In the model below, we also exclude precedence constraints. In subsequent assignments, different types of resources and precedence constraints are introduced.

S is a set of m stakeholders, $S = \{s_1, s_2..., r_m\}$

Q is set of n requirements $Q = \{q_1, q_2, ..., q_n\}$

R is a set of k releases, $R = \{r_1, r_2, ..., r_k\}$

r_i represents release i containing some subset of requirements $\{q_j\} \subseteq Q$, where every requirement q_j only belongs to one release

$p(s_i, q_j)$ is a number representing the priority of requirement q_j assigned by stakeholder s_i

$w(s_i)$ is a number representing the importance of stakeholder s_i

$P(q_i)$ is the weighted, normalized sum of all stakeholders' priorities of requirement q_i, defined as $P(q_i) = \sum_{s_j \in S} p(s_j, q_i) w(s_j) a_i$, where a_i are constants of normalization defined similar to the a_i constants in Section 2.1.2.

$c(q_i)$ is the cost of implementing requirement q_i

$C(r_i)$ is the capacity of release r_i

Optimization. The releases should be optimized by maximizing the total sum of all priorities of the requirements included in each release, for all releases:

$$max[\sum_{r_i \in R} \sum_{q_j \in r_i} P(q_j)]$$

subjected to the constraint that the sum of all requirements' costs in a release should be less than the capacity of the release:

$$\sum_{q_i \in r_i} c(q_i) \leq C(r_i), \text{ for all } r_i \in R$$

2.2.2 Simple release planning

The definitions in Section 2.2.1 represents a simplified model of the release planning problem, where there is only one type of cost and no special constraints on the ordering of requirements. Based on previous data that you have gathered about your 15 requirements, create a simplified release plan by following the steps below. We start by simplifying the model even further considering only one stakeholder.

Calculate a simple release plan manually by allocating some of your $n = 15$ requirements to two releases, denoted r_1 and r_2. In r_1 only a maximum of 35% of the total cost can be fitted, and in r_2 there is only room for max 15% of the total cost.

1. Fill in the priority data in the Priority column in the table below by transferring the data from one of your 2 stakeholders in Section 2.1.2.
2. Fill in the priority data in the Cost column in the table below by transferring the data from the \$100 method in Section 2.1.3.
3. Allocate some q_i to either r_1 or r_2 and transfer costs and priorities to the selected release respectively. The cost of a requirement is not allowed to be split between releases. Try to maximize the sum of priorities of scheduled requirements.

Req q_i	Prio $p(q_i)$	Cost $c(q_i)$	$p(q_i)$ if $q_i \in r_1$ else 0	$c(q_i)$ if $q_i \in r_1$ else 0	$p(q_i)$ if $q_i \in r_2$ else 0	$c(q_i)$ if $q_i \in r_2$ else 0
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
Sum:						

Total sum of allocated priorities $\sum_{q_i \in r_1} p(q_i) + \sum_{q_i \in r_2} p(q_i) =$ _____

Total sum of non-allocated priorities $\sum_{q_i \notin r_1, r_2} p(q_i) =$ _____

How long did it take to make a good manual release plan? _____

2.2.3 Advanced release planning

By involving different priorities from more than one stakeholder, adding multiple resources that each have their own costs per requirement, and introducing constraints on the ordering of requirements, the release planning problem becomes significantly more challenging.

As a preparation for the lab, you are requested to manually try to solve an advanced release planning problem for a small set of requirements. We will stick with just one stakeholder to make it a bit easier. [2]

As an example we will use 9 features listed in Table 1 below. We assume a hypothetical release planning problem with two releases "March" and "July" and two development resources "Team A" and "Team B". The teams have different skills (back-end & client dev) and the time to do their respective part differ, while both are needed. The teams have different available capacities to spend on the different releases. For the March release Team A can max spend 20 h and Team B can spend max 15 h. For the July release Team A has max 15 h and Team B has max 15 h.

1. Allocate some or all of the features in Table 1 by filling in its last column with a release name: March or July (or – if not allocated to any release), while abiding the releases' resource constraints. *Try to maximize the sum of all priorities of the allocated features.*
2. Fill in the requested results in the blank cells of Table 2.
3. We now introduce the following precedence constraint: `Feature("exportHtml") preceedes Feature("exportGraphViz")` implying that exportHtml must be in `Release("March")` release while exportGraphViz must be in `Release("July")` if both are allocated. Re-allocate some or all of the features in Table 3 now taking into account the above precedence constraint, while still abiding the same resource constraints as before. *Try to maximize the sum of all priorities of the allocated features.*
4. Fill in the requested results in the blank cells of Table 4.
5. How long did it take to make a good manual release plan? _____
6. How did you find your solution? Describe your methodology.
7. What are the difficulties of release planning?
8. How does the methodology to find the release plan change when precedence constraints are introduced?
9. Are the suggested solutions you have found optimal? Why or why not?

[2] Don't spend ages on trying to find an optimal solution manually; in practice we need tools for that as the release planning problem is similar to the so called knapsack problem which is NP-hard!

Table 1: Advanced release planning without precedence constraints.

Feature	Priority	Team A cost	Team B cost	Release
exportHtml	10	9	2	
exportGraphViz	10	7	8	
exportTabular	10	3	9	
exportLatex	7	6	4	
exportContextDiagramSvg	6	3	4	
syntaxColoring	3	6	2	
autoCompletion	4	3	3	
releasePlanning	7	4	5	
autoSave	9	6	7	

Table 2: Sums without precedence constraints.

Sum	Release March	Release July	Total
Team A's capacity	20	15	35
Team B's capacity	15	15	30
Sum of hours Team A worked			
Sum of hours Team B worked			
Sum of priorities			

Room for answering questions 6–7 in Section 2.2.3.

Table 3: Advanced release planning *with* a precedence constraint.

Feature	Priority	Team A cost	Team B cost	Release
exportHtml	10	9	2	
exportGraphViz	10	7	8	
exportTabular	10	3	9	
exportLatex	7	6	4	
exportContextDiagramSvg	6	3	4	
syntaxColoring	3	6	2	
autoCompletion	4	3	3	
releasePlanning	7	4	5	
autoSave	9	6	7	

Table 4: Sums *with* a precedence constraint.

Sum	Release March	Release July	Total
Team A's capacity	20	15	35
Team B's capacity	15	15	30
Sum of hours Team A worked			
Sum of hours Team B worked			
Sum of priorities			

Room for answering questions 8–9 in Section 2.2.3.

2.2.4 Constraint solving

The reqT tool includes an efficient constraint solver called JaCoP[3], enabling the formulation of prioritization and release planning problems using constraints over integer values in requirements models. After the problem has been formulated the constraint solver in reqT may automatically find a solution (if it exists), without the need for any further algorithm implementation.

1. Try these lines in the reqT console:

```
reqT> Var("a")        //constructs a variable named "a" that can be used in constraints

reqT> Var("a") > Var("b")     //constructs a XgtY constraint

reqT> Var("a") :: {1 to 10}   //constructs a Bounds constraint on values of Var("a")

reqT> val cs = Constraints(Var("a")>Var("b"), Var("a")::{1 to 10}, Var("b")::{5 to 15})

reqT> var m = Model(Stakeholder("s1") has cs)  //Constraints may be attributes in a model

reqT> cs.satisfy   //Constraints can be satisfied if a solution exists

reqT> cs.maximize(Var("b"))   //search for a solution that maximizes Var("b")

reqT> m = m + (Section("solution") has m.satisfy) //m.satisfy returns solution model

reqT> cs.satisfy   //By default, the solution search is initialized with random values
     //... thus if many solutions exists, each call to satisfy may pick a new solution

reqT> Req("q1")/Benefit >  Req("q2")/Benefit  //A constraint of integer attribute paths

reqT> m = m ++ Model(Req("q1")/Benefit > Req("q2")/Benefit).satisfy
                     //default bounds if no tighter bounds are given: {-1000 to 1000}
```

2. Construct a constraint problem with no solution:

3. Call `satisfy` on your unsatisfiable problem. What is the output?

4. Investigate available constraints in this reqT source file at line 261:
https://github.com/reqT/reqT/blob/3.0.x/src/reqT/constraints.scala#L261

5. Construct and solve some constraint problem using the `AllDifferent` constraint. You can construct a sequence of variables by typing e.g. `Vector(Var("x"), Var("y"))`, which is a subtype of `Seq[Var]`.

[3] See http://jacop.eu/ and http://cs.lth.se/edan01 if you want to learn more about CSP.