IT Demand Management
A Request Prioritization Approach

João Gonçalo Soares Campos

Dissertação para a obtenção do Grau de Mestre em Engenharia Informática e de Computadores

Júri:
Presidente: Prof. José Manuel Nunes Salvador Tribolet
Orientador: Prof. Miguel Leitão Mignolas Mira da Silva
Vogais: Prof. Maria do Rosário Gomes Osório Bernardo Ponces de Carvalho

Julho de 2009
Abstract: Demand management in IT has been, for many years, a challenge hard to cope. With the democratization of agile methodologies we are entering an era where all applications stay permanently in a Beta state. Companies start to deploy their applications earlier with less functionality and features, leaving a lot of development work to be done after delivery. Because of that, demand management is starting to be more and more a problem that relates directly to software maintenance and requirement prioritization. We propose a solution to this problem based on agile methodologies, with innovative approaches to classify and prioritize requirements, by enabling a broader range of participation by users and customers through a negotiation process transparent to everyone involved in the process.

Keywords: Demand management, software maintenance, product backlog, requirements prioritization, agile methodologies.

Acknowledgements

Gostaria de agradecer ao professor Miguel Mira da Silva por toda a orientação e apoio prestado ao longo deste trabalho, por ter acreditado em mim, dado o espaço que necessitava e por ter estado presente sempre que necessário. Queria deixar um agradecimento à professora Rosário Ponces de Carvalho pelas suas críticas construtivas e pelas sugestões pertinentes.

Queria demonstrar a minha gratidão ao Nuno Luís, Catarina Galhordas e Samuel Sousa por estarem disponíveis para participar na minha investigação e por facilitarem todo o trabalho realizado na empresa em que trabalham, sem eles esta tese dificilmente teria sido acabada.

Gostaria de agradecer, também, à Rita Carvalho, pela ajuda que deu na pesquisa que realizei e pelo apoio que sempre procurou dar, e à Patricia Lopes, pela conversa decisiva que me ajudou a concluir a última parte da tese.

Por fim, queria dedicar esta tese aos meus pais e à minha tia Fátima pelo apoio incondicional que demonstraram ao longo de todo o meu percurso académico e por nunca terem duvidado das minhas capacidades. Não teria chegado ao fim se não tivessem acreditado em mim.
## Contents

1. Introduction ................................................................................................ 1
2. Problem ..................................................................................................... 3
3. Related Work ............................................................................................. 6
   3.1. IEEE Standard 1219 ........................................................................... 6
   3.2. ITIL .................................................................................................... 7
   3.3. Requirements Prioritization ............................................................... 11
      3.3.1. Joint Application Development .................................................. 11
      3.3.2. Requirements Triage ................................................................. 12
      3.3.3. 100-Point Estimation ............................................................... 13
      3.3.4. Theory-W .................................................................................. 14
      3.3.5. Kano Analysis ........................................................................... 15
4. First Iteration ............................................................................................ 19
   4.1. Environment Characterization ........................................................... 20
   4.2. Proposal ........................................................................................... 22
      4.2.1. Request Registry ...................................................................... 23
      4.2.2. Voting ....................................................................................... 24
      4.2.3. Implementation ......................................................................... 25
      4.2.4. Testing ..................................................................................... 25
   4.3. Action Taking ................................................................................... 25
   4.4. Evaluating ........................................................................................ 26
      4.4.1. Observation .............................................................................. 26
      4.4.2. Interview ................................................................................... 27
   4.5. Specifying Learning .......................................................................... 28
5. Second Iteration ........................................................................................ 30
   5.1. Proposal ........................................................................................... 30
   5.2. Action Taking ................................................................................... 31
   5.3. Evaluating ........................................................................................ 34
   5.4. Specifying Learning .......................................................................... 35
      5.4.1. Process Impact ......................................................................... 37
6. Result ...................................................................................................... 41
   6.1. Request registry ............................................................................... 44
      6.1.1. Artifacts .................................................................................... 45
List of Figures
Figure 1 - Percentage of overall software costs spent on maintenance .............3
Figure 2: The ITIL Framework ........................................................................7
Figure 3: The ITIL Application Management Lifecycle ....................................8
Figure 4 - Kano Analysis ...............................................................................16
Figure 5 - The Action Research Cycle (29) ...................................................19
Figure 6 - User distribution by usage frequency ...........................................20
Figure 7 - Application’s Types of Users .......................................................20
Figure 8 - User perception of how long it took to implement a request ..........21
Figure 9 - User perception of application improvements ..............................38
Figure 10 - User that have requests that have not been yet implemented ......38
Figure 11 - Perceived user participation on prioritization decisions .............39
Figure 12 - Prioritization Process View .......................................................42
Figure 13 - Request registry activity ............................................................45
Figure 14 - The grouping activity .................................................................47
Figure 15 - Tactical Decisions Activity ........................................................50
Figure 16 - Voting Activity ..........................................................................53
Figure 17 - Testing Activity .........................................................................59
Figure 18: The SCRUM Process ..................................................................III

List of Tables
Table 1 - What Stakeholders want and what they offer ..................................4
Table 2: Theory-W Win-Win Steps .................................................................14
Table 3: Categorizing an answer pair ............................................................17
Table 4: Aggregating Results .......................................................................18
Table 5: SCRUM Lifecycle ..........................................................................II
1. Introduction

Demand management is the ability to balance orders for products and services while planning production and delivery (1). In IT, demand management can be viewed as the process that captures needs and requests turning them into a planned work schedule. These requests will be put into application features, either by creating new projects or by adding them into applications that are already running.

Although the need for new applications will never end, the reality is that the ones that already exist need to be maintained and improved. The tendency, today, is to build applications with an initial smaller set of functionality and features, deploy it, and keep improving it while already in production (2) (3).

The phenomenon described above will make tasks like adding functionality, correcting bugs, improving performance and other, which were usually performed before delivery, to be done with the application already being used. Application maintenance is the “modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment” (4), so developing after delivery is nothing more than software maintenance.

Software maintenance is frequently in second place to all other stages of software development (5). This choice may cause dismay since over 60% (6) (7) (8) of the resources are consumed in this stage. Software maintenance is therefore the single activity where more than half of the investment in applications is done.

We can subdivide maintenance into four different types (9):

a) **Corrective maintenance** deals with the repair of faults found;

b) **Adaptive maintenance** deals with adapting the software to changes in the environment;

c) **Perfective maintenance** mainly deals with accommodating new or changed user requirements. In concerns functional enhancements to the system;

d) **Preventive maintenance** concerns activities aimed at increasing the system’s maintainability.

Despite the obvious economic reasons, software maintenance is also needed to keep the application alive. When applications start to be used the old saying “I’ll know it when I see it” comes to the surface. When people see what was done they start to know what needs to be done. This practical issue creates problems with application maintenance. The flooding of requests that follows the first few days of utilization originates a huge amount of information that needs to be processed, evaluated, prioritized and, only then, implemented, tested and deployed. This is known as product backlog.
Dealing with software maintenance and requests backlog has, therefore, some limitations and constraints that should be kept in mind (10):

a) Differences in importance. Importance is subjective. Different people that use an application will have different needs and different viewpoints of what needs to be done and what should be done.

b) Limited project resources. Programmers, time and money are limited, requirements usually are not.

c) Long schedule. Software maintenance aims at increasing the longevity of a software application by improving it, by adding new features, and correcting deficiencies.

The Standish Group (2) states that 37% of the factors that challenge projects can be related to requirements management. Not having these factors fully addressed can create situations where requirements are implemented in a first in first out order, ad-hoc, arbitrarily or some other random factor.

When trying to minimize spending and maximize profits these approaches are counterproductive, since there is no kind of value estimation to each requirement. Many propose different approaches to solve this problem.
2. Problem

It is widely recognized that software maintenance is the most expensive stage of the application lifecycle. The estimated cost for this phase can vary from 60% to 80% (8) of the overall of the software budget and 50% of programmer effort (11). Figure 1 depicts the reality over the years of costs associated with maintenance relatively to the overall software cost (12).

![Proportional Costs in Software Development](image)

**Figure 1 - Percentage of overall software costs spent on maintenance**

So software application projects outcomes have to start to be evaluated based on different metrics, because on the change of view of application development (3) (2), outcomes have to start to be measured also based on application management metrics.

The Standish Group Chaos Report (13) study categorizes projects outcome into three different categories:

- a) **Succeeded**: delivered on time, on budget, with required features and functions;
- b) **Challenged**: delivered late, over budget and/or with less than the required features and functions;
- c) **Failed**: cancelled prior to completion or delivered and never used.

Another study indicates that deficient requirements are the single biggest cause of software project failure. From studying hundred of organizations, Capers Jones discovered that requirements engineering is deficient in more than 75% of all enterprises (14). Some even state that getting requirements right might be the single most important and difficult part of a software project (15).
The traditional view of stakeholders defines them as individuals or organizations that are actively involved in a software project or whose interests the project affects. This means that any given software project can include customers, users, project managers, analysts, developers, senior management, and quality assurance staff (15). In Table 1 we can see the wide range of expertise and motivations that stakeholders typically exhibit (16).

<table>
<thead>
<tr>
<th>Stakeholder</th>
<th>Motivation</th>
<th>Expertise Areas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>Introduce change with maximum benefit</td>
<td>Business and information system strategies, industry trends</td>
</tr>
<tr>
<td>User</td>
<td>Introduce change with minimum disruption</td>
<td>Business process, operating procedures</td>
</tr>
<tr>
<td>Project Manager</td>
<td>Successfully complete the project with the given resources</td>
<td>Project management, software development and delivery process</td>
</tr>
<tr>
<td>Analyst</td>
<td>Specify requirements on time and within budget</td>
<td>Requirements Engineering methods and tools</td>
</tr>
<tr>
<td>Developer</td>
<td>Produce technically excellent system, use latest technologies</td>
<td>Latest technologies, design methods, programming environments and languages</td>
</tr>
<tr>
<td>Quality Assurance</td>
<td>Ensure compliance to process and product standards</td>
<td>Software process, methods and standards</td>
</tr>
</tbody>
</table>

Table 1 - What Stakeholders want and what they offer

As we can see in Table 1 only the first two types of stakeholders relate to business. Beside this two types project owner and business project manager can be included. The business project manager can be defined as the interface between technical staff and business staff. For the purpose of this thesis we will only consider stakeholders, in prioritization processes, customers, users, project owners and business project managers.

Since requirements derive from stakeholders requests, stakeholders believe that all requirements (suggested by them) should have the highest priority (10). Addressing the factors that challenge projects means addressing request prioritization. The nature of request creates some roadblocks that cause, most of the times, innumerous problems to project management and application maintenance scheduling.
The associated challenges and risks that must be addressed when evaluating, prioritizing and implementing application maintenance processes are (10):

a) Nature of requirements;
b) Number of requirements;
c) Managing demand accordingly to resources available;
d) Changing requirements priorities;
e) Incompatibility in requirements prioritization;
f) Collaboration between process owners and process users;
g) Subjective prioritization.

Knowing these facts leads us to conclude that an efficient and value-adding prioritization can be challenged by stakeholders’ view of application and their own personal agendas. This means that different perspectives, beside the tactical one, have to be put into the requirement prioritization process. Considering everyone that uses the application as a stakeholder, in these situations, can bring a fresh perspective to the problem and, combined with the stakeholders’ perspective; help generate a weighted prioritization based on usage issues and tactical orientation.

The standards, best practices and methods that exist in nowadays do not fully address the problems and challenges identified here. We will describe, in the following section, some of them.
3. Related Work

In this section we will describe the most relevant standards, frameworks and processes related to application management, software maintenance and requirement prioritization. We will point out, for each of them, their strengths and weaknesses. The work here presented represents the research done for the elaboration of this thesis proposal.

3.1. IEEE Standard 1219

The IEEE Standard 1219 for software maintenance defines a process that includes the following phases:

- a) Problem/modification identification, classification and prioritization
- b) Analysis
- c) Design
- d) Implementation
- e) Regression/system testing
- f) Acceptance testing
- g) Delivery

Each phase has inputs, controls outputs and process specification (4). Because of the scope of this document some phases will not be considered.

*Problem/modification identification, classification and prioritization*

In this phase we identify software modifications, classify them and assign an initial priority. Each modification request should be evaluated to determine its classification and handling priority. Classification shall be identified from the four types of maintenance (9):

- a) Corrective;
- b) Adaptive;
- c) Perfective;
- d) Emergency.

For each modification requirement in software applications, IEEE defines that the following activities shall occur:

- a) Assign an identification number;
- b) Classify the type of maintenance;
- c) Analyze the modification to determine whether to accept, reject or further evaluate;
- d) Make preliminary estimate of the modification size/magnitude;
- e) Prioritize the modification;
- f) Assign a modification request to a block of modifications scheduled for implementation.
After the completion of these activities modification request shall be uniquely identified and entered into a repository. The output of this process shall be validated modification requests and process determinations that were stored into the repository.

This standard has some interesting suggestions to be used in the requirement prioritization process. The use of blocks of modification, enclosing multiple requirements, can be useful to focus prioritization process in a specific area of project intervention, and early effort estimations can help the elucidating stakeholders about the cost of the requirements and prioritize them accordingly.

One of the problems with this approach is the preliminary acceptance or refusal of a modification request. This decision should be taken only by stakeholders and users through the prioritization process. A modification request is translated to requirements and put into the backlog. When the next prioritization process starts, stakeholders will decide, by prioritizing requirements, which should be included in the next block of requirements to be implemented. The ones left out are returned to product backlog to be reconsidered.

3.2. ITIL

ITIL (the IT Infrastructure Library) is a comprehensive, consistent and coherent set of best practices for IT service management processes, promoting a quality approach achieving business effectiveness and efficiency in the use of information systems (17).

![Figure 2: The ITIL Framework](image)

The ITIL Framework is divided in different areas (see Figure 2). The core of the discipline is the service management, divided into service delivery and service support. The rest of the framework completes the field of studies with the business perspective, ICT infrastructure management, security management, planning to implement service management and applications management.
Due to the scope of this thesis, we will focus on applications management.

The ITIL approach to applications management derives from the traditional approach. The application lifecycle defined in this framework comprises traditional application development phases and service management phases, from ITIL, combined into a single lifecycle (18). Figure 3 represents the application management lifecycle as defined by ITIL.

The framework defends that adding new features into an application does not constitute a new application lifecycle, since it does not create a new application, in fact, the lifecycle of any application continues until is formally ended by retirement (18). The only particularity is in the optimization phase, where it is stated that in case of optimizations implementation, the application is to be kept, temporarily, in two different states. Since the application is still “alive” it will continue to require technical maintenance and bug correction. And since it becomes necessary to implement optimizations, approved in the optimization phase, a new cycle of application management is started concurrently.

It is important to understand that only one cycle exists, despite this fact, improvements put into the application will be done following the cycle here describe but do not constitute a new application management lifecycle. The lifecycle that results from optimizations decisions will, eventually, be merged with the “official” application management lifecycle.

We will now describe in more detail the most relevant stages of the application management lifecycle, accordingly with the thesis scope, as proposed by the ITIL framework.

**Requirements**

The requirements phase identifies the functionality, performance levels, and other characteristics that the application must satisfy. The requirements produce on this phase set the foundation of all other
phases and acceptance criteria. ITIL states that requirements for modification of an existing application may originate in the optimize phase in the form of a request for change (18).

The framework defines three different types of requirements that decision-maker have to consider:

a) Functional Requirements: describe things that the application should do.
b) Non-functional Requirements: define requirements and constraints on the IT system.
c) Usability Requirements: ensure that the system meets the expectations of its users with regard to its ease of use.

This stage accommodates some preparation for future changes in the application. Change cases are used to clarify the scope and direction with the sponsor. These cases specify the flexibility that is required to support likely future demands. Because of this extra architecture and design work will be needed at this point to ensure the change cases can be met in the future at reasonable cost (18).

Operate

This process step covers the day-to-day operations. Changes in regulatory requirements would be addressed in this step. If end users uncover flaws in the application, they are addressed in this step. In addition, service levels are monitored and application performance is measured and reported. When service levels are missed, this process step outlines the step-by-step actions required to address the problem.

As in other process steps, a number of activities, roles and artifacts should be addressed to correctly address the operational stage (18):

a) Day-to-day maintenance activities to maintain service levels: each application will have a number of tasks performed on it to ensure that it remains healthy.
b) Application state: it is important to understand and practice how to restore application state. This may be needed as result of an unplanned activity, such as hardware or system failure, or following a planned activity, such as moving or maintaining a server.
c) Benefits of an application: an assessment of benefits provided by the application compared with those that it was designed to deliver should be conducted in this phase.
d) Operation management checklist: operating the application will also follow a process that allows it to be run against high-level manageability requirements for the organization and any special management requirements that have been identified for the application.
e) Organization of operations team roles
Optimize

This is the last step in the iterative Application Management process. Both functional and performance optimization are addressed in this step. Performance Management teams analyze applications to identify any performance improvement opportunities.

ITIL framework defines an application review process that evaluates if the application is continuing to meet the ongoing needs of the business of the technical requirements of the infrastructure that is operating within. This process can be triggered by a number of reasons (18):

a) A fixed period of time has passed since the last review;
b) Problem management has identified issues with the current version that require modifications;
c) Business requirements are changing and the use of the current application needs to be reviewed against these new requirements;
d) The technical infrastructure that the application relies on has changed and the application needs to be reviewed against this.

The review process should be addressed considering different areas related to the application, where change will have some kind of impact. People, business processes and technology are areas that should be assess against the review process (18).

Whenever a change is required in the application a request for change should be prepared. Information should be gathered considering the request and include business need and business justification, who is responsible for the change, the amount of effort needed to complete the change, the prerequisites for the change, when the change needs to be completed, who the stakeholders are and the success criterion for the change. The modification proposal is then used to drive a new iteration, for the same application, of the application lifecycle (18).

The ITIL Application Management Lifecycle does a distinction between bugs and modification requests and defines different processes to address each of them. The urgency of problem solving, when compared with optimization implementation, can be argued. Solving a bug issue may not be more important or valuable that implementing a new feature into the application. For example a low impact bug with a high consumption of resources for resolution should, in some cases, be left aside to make possible a new feature implementation. Therefore the effort put into the resolution of a problem in the operate phase should be considered against the benefits that outcome of this task and not only because of service level issues.
3.3. **Requirements Prioritization**

There is, in the IT industry, the conscience that requirements have different degrees of importance. Prioritizing methods guide decision makers in their task of analyzing requirements in order to assign them members or symbols reflecting their importance (19). In the following subsections we will describe some of the most relevant methods that exist nowadays.

### 3.3.1. Joint Application Development

Joint Application Development (JAD) is intended to accelerate the design of information systems and promote comprehensive, high-quality results (20). The JAD session is a highly facilitated multi-day workshop where users and builders of a software system intricately define detailed requirements (21).

A group of experts in different functional areas empowered to make technical and, sometimes, contractual decisions, are gathered with the goal to finish a stated agenda with no open issues (21).

The JAD process begins after high level requirements have been developed and documented. The process can be broken into three different stages (21):

a) Preparation of JAD sessions;
b) Holding the JAD session;
c) Post-JAD review.

A number of roles were defined for these meetings. Every meeting should include a session leader, a facilitator and a scribe. Technical specialists in the areas of database and graphical user interface are required, along with domain expert in the target application (21).

There are four necessary building blocks for a well-run JAD meeting (20):

a) Facilitation: A designated leader (or leaders) manages the meeting.
b) Agenda setting/structure: The meeting must have a plan of action.
c) Documentation: One or more designated scribes carefully document everything in the meeting. Various lists are rigorously maintained.
d) Group Dynamics: Group dynamics techniques are used for inspiring creativity, resolving disagreements and handling speaking protocols.

The JAD meeting objectives vary accordingly with system development phase. Meetings in early stages of development focus on higher-level issues: defining objectives, decomposing the domain into smaller functions, defining boundaries and scope. In these meetings,
participants begin to compile a list of assumptions, constraints and open issues; to target specific people and organizations for tasks; and construct timelines. Once the JAD meetings get into the latter phase the main focus becomes more specific. Users are asked to give increasingly more detail. These meetings are often longer, three to five days, compared to one or two days in early stages.

This methodology is mainly used to plan project implementations. It considers many aspects of project planning including, but not only, requirement prioritization. JAD can be used in application maintenance since it does not state that it is a single event in applications lifecycle. However, due to the duration of the workshops and the requirement of onsite participation of each stakeholder, JAD sessions can turn a meeting into an agenda nightmare, turning the planning of these meetings very hard and much spaced in time. This process does not grant the necessary agility and flexibility that a dynamic process, such as application maintenance and requirement prioritization, needs.

3.3.2. Requirements Triage

Requirements Triage (22) and it is a multistep process that includes establishing relative priorities for requirements, estimating resources necessary to satisfy each requirement, and selecting a subset of requirements to optimize probability of the product's market success in the intended market (23).

Davis (22) states that determining what requirements a product will satisfy follows a process similar to triage. He divides requirements into three groups, as in medicine triage, and classifies each requirement into one of those groups.

a) Requirements that the next baseline clearly must satisfy.
b) Requirements that the next baseline need not satisfy.
c) Requirements that the product could incorporate, but that the development team must first carefully weight against available resources.

Each group will require different strategies. The first group of requirements are put into the product as features, the second group should be put into a bin for consideration in future releases and the last group should be considered for implementation according to resources available, a second prioritization for the last group can be used to determine which requirements, that are nice to have, should be included first.

Triage is the process of determining which requirements a product should satisfy given the time and resources available. Requirements triage is somewhat similar to Kano Analysis, discussed in section 3.3.5. This process sets a way of defining very well what requirements should be put into next phases of development. It uses as input stakeholders information to define what should be done, what should not be done and what is desirable to be done if the opportunity arises. Nevertheless, it does not take into consideration that requirements may
have many different sources and may have different urgencies, because it makes the assumption that requirements are constructed based on users complaints. For example if the process supported by an application has to be changed, the application should accompany that change as soon as possible.

On the other side, a good characteristic of this process is the understanding that resources are limited and that there will exist requirements that should be implemented but may not be implemented. Not using understanding that requests may have different sources and degrees of importance can turn this process into a beautification project and leaving strategical features out. This can happen because users lack the long term perspective, do not fully understand processes underlined by applications and are more focused on usability issues that on process management and improvement.

3.3.3. 100-Point Estimation

The 100-Point Estimation technique is probably one of the oldest and simplest forms of prioritization (24). Leffingwell and Widrig (25) state that “This simple test is fun, fair and easy to do”. In this technique each stakeholder gets a hundred points to “buy” requirements. For each requirement, each person writes down in a piece of paper how much they think that one requirement is worth. After this process is concluded one person gathers all the “bids” and sums the cumulative result, for each requirement, the one with the highest score is considered the most important requirement (24).

This kind of prioritization method may take a while. Since each person has a small amount of points, careful consideration should be put into the voting.

The 100-Point Estimation can be applied as follows (24):

a) Put all requirements in a row;

b) Divide all the points among the requirements, according to which requirements are most important to the system.

Leffingwell and Widrig (25) discuss that the problem with this method is that it can be done only once in every project. Some reasons can be pointed to justify this fact. One of them is that after the first prioritization everyone is conscious of the voting results. Therefore, all subsequent voting sessions will be conditioned. People, after the first voting will try to push their agenda by placing more of their own requirements on the top requirements list (24). There should also be established a limit to how much one person could spend at one single requirement. If no limit exists, stakeholders could predict that which requirements will be voted as high priority and not vote on those ones (since they are going to be elected anyway) and vote all their points on a single requirement (25).

Since maintenance is an iterative process that will require several iterations and, because of that, many requirements prioritizations, it becomes obvious that 100-Point Estimation is not suited for application maintenance requirements prioritization. The only way it can be applied
is if it is used only by the development team where, usually, exists less conflict of interests or personal agendas. SCRUM methodology uses a similar technique to decide which requirements of the sprint backlog should be implemented first (see Appendix 1).

3.3.4. Theory-W

This theory holds that the primary job of the software project manager is to make winners of each of the parties involved in the software process (26).

To better understand this theory becomes necessary to understand the win-win concept behind it. According to Boehm and Ross (26) win-lose situations are those in which only one of the participants in the process gets any kind of profit from it. These types of situations are, for example, the implementation of a sloppy software system where only some functionality is correctly implemented. In this case the users loose but developers and customers win, due to the low cost of the project.

Creating a profit sharing or win-win situations was done on the field of negotiation (26). This theory defines a set of steps to be performed in requirement negotiation. Theory-W has four steps (23):

a) Separate the people from the problem.
b) Focus on interests, not positions.
c) Invest options for mutual gain.
d) Insist on using objective criteria

These steps were expanded to form a set of win-win pre-conditions, and some further conditions for structuring the software process and resulting products. The theory-W steps are depicted in Table 2 (26).

<table>
<thead>
<tr>
<th>1. Establish a set of win-win</th>
</tr>
</thead>
<tbody>
<tr>
<td>a) Understand how people want to win;</td>
</tr>
<tr>
<td>b) Establish reasonable expectations;</td>
</tr>
<tr>
<td>c) Match people’s tasks to their win conditions;</td>
</tr>
<tr>
<td>d) Provide a supportive environment;</td>
</tr>
<tr>
<td>2. Structure a win-win software process</td>
</tr>
<tr>
<td>a) Establish a realistic process plan;</td>
</tr>
<tr>
<td>b) Use the plan to control the project;</td>
</tr>
<tr>
<td>c) Identify and manage your win-lose or lose-lose risks;</td>
</tr>
<tr>
<td>d) Keep people involved;</td>
</tr>
<tr>
<td>3. Structure a win-win software product</td>
</tr>
<tr>
<td>a) Match product to users’, maintainers’ win conditions.</td>
</tr>
</tbody>
</table>

Table 2: Theory-W Win-Win Steps
One important point, of this theory, is that it supports negation to solve disagreements about requirements, so that each stakeholder has a “win” (23). Making everyone a winner has a number of implications including the use of two subsidiary principles (26):

a) Plan the flight and fly the plan;
b) Identify and manage your risks.

The first part of the first principle states that a well made plan should be put in place, this plan should meet standards for easy development, classification and query. “Fly the plan” means that progress should follow what was predicted in the original plan. Finally, the second principle states risk assessment and management should be used. This aspect maintains the status quo of the negotiation by guarding stakeholders’ win-win conditions (23).

In this kind of negotiation each stakeholder privately ranks requirements before negotiations start. By doing so it also considers which requirements can be negotiated and which he considers “must have”. This consideration will give to the stakeholder some maneuverability so that individual winning and losing conditions are fully understood.

For large amounts of requirements this kind of negotiation can be hard to be put in place. It is easy to prioritize 10 requirements but perform the same task (with a clear notion of all requirements) with 1000 requirements can be obviously harder. For this reason the process here described can be transformed into a slow and painful process for every actor participating. Since maintenance is to be done in regular intervals the repetition of such process is not the best solution for the problem in hands.

3.3.5. Kano Analysis

The Kano Model of Customer (Consumer) Satisfaction classifies product attributes based on how they are perceived by customers and their effect in customer satisfaction (see Figure 4). These classifications are useful for guiding design decisions in that they indicate when good is good enough, and when more is better.

Mike Cohn (27) defines Kano’s Analysis three types of features as:

a) Exciters/Delighters – Features a user doesn’t know she wants, until she sees it
b) Linear – The more of it, the better
c) Mandatory/Baseline – Must be present in order for users to be satisfied
Beside the three attributes already referred, three additional attributes are mentioned in the Kano Model Analysis (28):

a) Indifferent attributes – The customer does not care about this feature
b) Questionable attributes – It is unclear whether this attribute is expected by the customer
c) Reverse attributes – The reverse of this product feature was expected by the customer

Kano Model uses user input to perform requirement classification into the three categories defined above. The user input should be obtained by asking to simple questions for each requirement (27):

a) Rate your satisfaction if the application has this feature? (functional question) and;
   b) Rate your satisfaction if the application did not have this feature? (dysfunctional question)

To answer these questions users should be provided with a close set of standard answers. This set of answers, for each question and user, will be mapped into a matrix that classifies each requirement into one of the three categories. For example Cohn (27) suggests that each question should have the five different choices of answer:

a) Like;
b) Expect;
c) Neutral;
d) Live With;
e) Dislike.
Both functional and dysfunctional answers to each question are put into a classification matrix that will classify the requirement based on the Kano Model Analysis categories (27).

<table>
<thead>
<tr>
<th>Dysfunctional</th>
<th>Like</th>
<th>Expect</th>
<th>Neutral</th>
<th>Live With</th>
<th>Dislike</th>
</tr>
</thead>
<tbody>
<tr>
<td>Like</td>
<td>Q</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>L</td>
</tr>
<tr>
<td>Expect</td>
<td>R</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>M</td>
</tr>
<tr>
<td>Neutral</td>
<td>R</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>M</td>
</tr>
<tr>
<td>Live With</td>
<td>R</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>M</td>
</tr>
<tr>
<td>Dislike</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>Q</td>
</tr>
</tbody>
</table>

Table 3: Categorizing an answer pair

In Table 3 we can see a matrix that makes the correspondence between the answers to the questions and the category that the requirement will fit into. For each pair of answers Cohn (27) states that the correspondent category can be obtained. The letters in each cell are the different categories discussed earlier in this section:

a) M – Mandatory requirement;
b) E – Exciter requirement;
c) I – Indifferent requirement;
d) R – Reverse requirement;
e) Q – Questionable requirement;
f) L – Linear.

Cohn (27) divides requirements into three different categories, size wise. User stories are descriptions of desired functionalities told by the user’s perspective. Themes are collections of user stories that are related. Finally epics are larger user stories. It is also stated that prioritization should occur at theme level because individual stories usually cannot be prioritized against each other.

After theme categorization, from all users, is done, results are aggregated into a matrix where each line represents a theme of user stories and each column the number of responses, from users, that correspond to each categorization (see Table 4).
Based on the theme classification we get the set of features to include in the next release (27):

a) All mandatory features, by definition, must be present.
b) Some amount of linear features.
c) Leave room for at least a few exciters.

This model presents a set of interesting characteristics. It involves users in prioritization and groups requirements into a hierarchy. These hierarchies can be used to group requirements and introduce a set of policies regarding requirements implementations “portfolio management”. This enables stakeholders’ decisions regarding which set of features should development be focused on and leave to users the decision of which requirements, regarding that particular feature, will be implemented. The downside is that this model requires a large amount of interviews with big groups of users and with a lot of questions, if a lot of requirements are to be considered. This is a time consuming task that cannot be easily performed every month or so.

<table>
<thead>
<tr>
<th>Theme</th>
<th>Exciter</th>
<th>Linear</th>
<th>Mandatory</th>
<th>Indifferent</th>
<th>Reverse</th>
<th>Questionable</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>10</td>
<td>31</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>22</td>
<td>20</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>27</td>
<td>10</td>
<td>12</td>
<td>6</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 4: Aggregating Results
4. First Iteration

This investigation method used for this thesis is based in the Action Research Method (29) (see Figure 5) and comprises five different stages. In this section we will describe the different actions taken during this investigation and how they affected the end result.

![The Action Research Cycle (29)](image)

Using this research method requires an evaluation that includes determining whether the expected outcomes of the proposal were realized, and whether the effects relieved the problems. A critical evaluation must be done in order to determine if changes were caused by a number of routine or non-routine actions, or if indeed were caused by the actions underlying the approach taken in the proposal implementation.

In each phase we will describe what the research objectives were, what conclusions were drawn from the action and how they affect the initial proposal.
4.1. Environment Characterization

The action was performed in a corporate environment. The object of study was an application in production for almost a year, when the study started. The application was used in regular basis by 20 users with different usage frequencies and responsibilities.

![Application Usage Frequency](image)

**Figure 6 - User distribution by usage frequency**

Figure 6 depicts the distribution of users accordingly to the type of usage that they claim to have. This result indicates that a fair amount of knowledge about existing features and usages is common to all users.

In terms of responsibilities, process participants can be categorized into four categories. Each of them represents a distinct type of utilization. In the Figure 7 we can see the distribution for each type of user.

![Types of Users](image)

**Figure 7 - Application's Types of Users**

- Team Coordinator: 15
- Project Owner: 3
- Others: 1
We can define each type of user by its type of utilization and application responsibilities:

- **Team Coordinator:** Uses the application for data manipulation and team control. Controls the activity of other users with some of the features that the application offers. This kind of user is more concerned with data interpretation and business process control;
- **Project Owner:** Uses the application for high-level information reports. He is concerned with process reengineering, user data collection and application evolution;
- **Business Project Manager:** Also a Team Coordinator. Additional responsibilities include application back office maintenance and request gathering and prioritization. Establishes the possible evolution paths, which the application may follow, with the Project Owner.
- **Other Users:** Other users are responsible for data gathering in the field and data insertion into the application. The application supports and accounts their activity.

Decision making, regarding application developments, was performed, in this particular company, by taking a centralized approach. Every request was reported to the business project manager, including the requests from the project owner. At the end of each implementation cycle, which duration varied from cycle to cycle, business project manager and project owner discussed what should be done in the next cycle, considering the present requests and business process needs.

Not having a fixed duration for each cycle made impossible for users to have a perception of how long each request implementation took. If we see maintenance as a service (30) this characteristic may harm the perception of the service quality.

![Figure 8 - User perception of how long it took to implement a request](image)

In Figure 8 we can see that very few users had any kind of idea about implementation times for request. The ones who answer yes were them
asked to give an estimate for request implementation effort. The values were comprised between 4 and 24 hours.

The notion of backlog did not existed in the strict sense of the word. Requests were registered by business project manager and compiled. The development team only knew that they existed when the next maintenance cycle was planned. There is no guarantee that all requests made were registered by the business project manager or even communicated to the project owner, other users or the development team.

As we can see from this contextualization, the population eligible to participate in the process was committed to use the application and it embraced all kinds of different usage and responsibilities. It had little perception about the effort involved in implementation tasks or what were the pending requests.

4.2. Proposal

Our proposal combines some of the models and processes described earlier. The basic assumption of this proposal is that IT demand management can be coped through an effective software perfective maintenance. Since maintenance results in a service being delivered to the costumer (30) it needs to be performed recurrently. Agile methodologies suggest a similar approach for software development. In this thesis we assume that software maintenance is a task performed in regular time intervals and, therefore, can be compared to SCRUM methodology’s Sprints (see Appendix 1).

A single sprint can be view as a complete project that starts with a small planning and requirements analysis and ends with an integration testing and, sometimes, customer inspection and acceptance (31).

We established in chapter 2 that requirement prioritization is one way to minimize risks that can be assigned to too many challenged projects (2). So the challenge is to define a process, which being agile, can make a clear and quick request management.

The objectives of this proposal are:

- Empower application users that usually do not have a say in the prioritization process;
- Publish and inform everyone of what is being done and what can be done in terms of application development;
- Make clear to all users what the cost of their requests is. If application management is a service we need to create some sort of service level management and service level agreements that can be audited by “clients”;
- Increase the business value of the application maintenance cycles by creating a prioritization method that enables a better management of the backlog;
To achieve the goals we recognize several requests sources. Gathering requests is the first thing to do, when managing demand. Request gathering is the pre-requisite to implement this proposal. Requests can have many different sources and types. We recognize three different types of requests sources:

1) **Business Process Requests:** typically, these are requests that are defined by process owners and higher management levels. They relate to process reengineering and process implementation tuning.

2) **Usage requests:** these are requests that end users produce based on the utilization of the application. The objective here is not to add function, or at least no usually, but to improve the user experience quality.

3) **Bugs:** bugs are requests that report faults or unexpected behaviors in the utilization of the application.

For an efficient request prioritization a request backlog has to be in place. This backlog should contain all requests done by stakeholders. The process that we propose comprises four different activities:

1. Request registry;
2. Voting;
3. Implementation;
4. Testing.

We will describe in more detail each phase in the following subsections.

### 4.2.1. Request Registry

Request registry is an asynchronous activity. It does not stop when the process enters into the following phases. This assures that requests made during the prioritization process, implementation, testing or acceptance, are still gathered into the backlog for future maintenance cycles.

Requests should be communicated to the requirement engineering team, without intermediaries or approval mechanisms. Request validity is achieved by guarantying that users are not misinformed and the request is already implemented. A quick inspection of the application’ features should separate new requests from the already implemented features.

Registering requests can be achieved in many ways and by many means. The basic principle is that all requests should be registered with the following information:

- A detailed request description;
- The request author (user, customer or other);
- Request category.
A request detailed description will be used for voters’ clarification during the voting activity. A good and clear definition will help voters understand clearly which goal, feature or process is underlying the request. A clear definition is a corner stone for the whole process. Without it, the voting process can be challenged or the final result inaccurate due to a subjective interpretation of the request meaning. If the requirements engineering team has any doubts about a request it should contact the author for further elicitation.

4.2.2. Voting

This activity should include the participation of a large majority of the end users and of all business staff that has some kind of participation in the project development. A voting period should be set, on which all voters can access the voting tool, and voters should be informed that the requirements list is available for voting.

The voting method is a combination between 100 point estimation and theory-W negotiation method (see sections 3.3.3 and 3.3.4). The priority assigned to each requirement represents a number of votes cast on that particular item. Keep in mind that a single request may be decomposed into several requirements.

As stated before, 100 point estimation may originate biased prioritization when the results of the first round are made public, the solution proposed for this problem is to use this method only once in each project (25). Another problem is the number of points put into a single request. If a request is of extreme importance for a single voter, he could be tempted to put all his points into this single request (25).

The approach, taken in this thesis, is to set the number of votes that can be assigned to each requirement. Each rank of priority has a defined number of votes associated. This means that voters cannot put more weight into a single requirement and the decision has, therefore, to be consensual among the group.

Theory-W states each participant should have a list of requirements already prioritized. The main objective of this theory is to make every participant a negotiation winner. Each participant selects a number of requirements that he considers “must haves” and a second list of requirements open for negotiation. The main objective of each participant is to guarantee that all its “must haves” are on the final list and as many as possible negotiable requirements (see section 3.3.4).

Theory-W is translated, in the process here suggested, as the possibility of any voter checking the current overall state of the voting process and which requirements are, so far, the most voted ones. Viewing the current state of the final list gives them the capacity of prioritize accordingly to their personal agendas, but since each priority has a fixed number of votes associated even the highest priority of a single voter won’t be enough to make sure that a requirement is elected.

Votes should be assigned to each requirement based on the place that they occupy in the personal list of each voter. This means that the requirements with higher priorities will be the ones that actually have fewer votes. For example the requirement with the highest priority will
have only one vote, the second one will have two and so on. The final list is calculated based on the number of votes that each voter cast into each requirement. The requirement with fewer votes will be therefore the most voted one.

4.2.3. Implementation

Implementation techniques are not part of the scope of this thesis. The end result of the process, here proposed, can be seen as the requirements list of any given project with any given implementation methodology or technology.

The actual list of requests that should be implemented during the cycle is defined by the most voted requirements, during the voting activity.

4.2.4. Testing

Testing is the final activity of the process here proposed. In this phase request authors will be asked to validate implementation. For each request implemented, in the previous activity, the request author will be asked to validate the implementation. Upon validation the request is marked as implemented, in the backlog. If request is not validated it should be kept available for implementation in future maintenance cycles.

4.3. Action Taking

In the first iteration the main objective was to determine how the backlog should be constructed. At the beginning of the test the backlog was composed by technical requirements that were constructed and derived from stakeholder’s requests. The main issue was understating if this particular structure was compatible with non-technical stakeholder prioritization. Since the proposal requires non-technical stakeholder participation it was necessary to determine if what we were showing to them was something that they could understand.

In order to determine if eligible voters were able to prioritize technical requirements and associate them with the underlying requests, we asked the business project manager to help us prioritize a list of requirements. Each requirement had an effort estimation associated, that represented the time to complete the implementation of the requirement.

The business project manager had the responsibility of channeling all requests from business actors to the development team and was, therefore, the most informed business actor about technical issues. If he was not able to prioritize requirements no one would be without additional training.

The list presented to be prioritized was composed by 45 technical requirements with effort estimations that were comprised between 2 and 16 hours. Some requirements had dependencies, meaning that they could not be implemented before some other requirement, and all of
them related to requests registered and communicated by the business process manager.

No kind of tool, beside a spreadsheet, was used to support this activity. The prioritization was done with the researcher. The researcher simply asked the business project manager what requirements should be implemented first.

Each requirement was presented with a brief description, a column for observations that was used to clarify some of the more complex requirements, the number of hours estimated for the implementation and a field where the implementation date should be put.

The business project manager objective was to prioritize, accordingly to what he thought were the business priorities, the whole list of requirements. No information was given about what where the time restrictions for implementation. This means that the person doing the prioritization assumed that all requirements would be eventually implemented.

During the activity the researcher observed the business project manager and responded to any doubts about the requirements. At the end of the prioritization a brief interview was performed in order to identify issues about the activity and the information presented to the person doing the prioritization.

In the following section we will describe the issues that were identified during the observation and interview. After that we will discuss the conclusions that were drawn from the activity observation and the interview.

### 4.4. Evaluating

The evaluation of the first iteration was performed based on observation of process execution and by interviewing the participant to understand possible roadblocks and unexploited potential. In the following subsections we will describe each of them in more detail.

#### 4.4.1. Observation

During the prioritization process the participant showed difficulties understating what was the underlying request of the requirements presented. Since requirements were a decomposition of a request the estimation times were small, so the participant stated that the requirements with bigger estimation times should be done first.

Another problem observed was related to existing dependencies between requirements. The voter did not understand the concept of requirements dependencies. This means, that the research had to warn him every time a requirement, that needed another one implemented first, was put with a higher priority than the one it depended on.

The participant also had difficulties prioritizing requirements that he did not understood. Despite the fact that he was responsible for the communication between business and the development team he did not show any knowledge about technical requirements. Often the researcher
had to explain what the requirement was and how it was related with a specific request.

4.4.2. Interview

After the observation phase a number of questions were asked to the participant. These questions were used to clarify some issues observed during the previous activity in order to better understand why particular choices were made and how the information items used in the activity affected them.

The issues that we addressed during the interview were:

a) Understanding technical terms and requirements description;
b) Request decomposition into technical requirements;
c) Prioritization criteria;
d) Amount of requirements;
e) Requirements dependencies.

For each issue a brief discussion was conducted indicating problems, related to each of them, that affected the prioritization task.

Regarding the language applied in requirements definition, the participant stated that the technical terms used were not easily understandable, even for the business project manager that had some knowledge due to his involvement with the development team. Some of the descriptions were far too technical for his comprehension and he stated that without the presence of the researcher he would never be able to perform the task in hands in a reasonable amount of time.

When asked if the participant identified the requirements that implemented each request, the participant responded that the only way he identified the request beneath any given requirement was through specific words that identified known features of the application. He also stated that he did not need to know all requirements necessary for request fulfillment and that it would be easier to prioritize requests or developments associated with specific application features.

The researcher, during observation, noticed that the requirements with higher effort estimations were the ones with higher priorities assigned. When inquired about this the business project manager argued that he concluded that those were the requirements with higher risks associated and therefore should be done first. Since all requirements were going to be implemented he thought that would be safer to implement them first to minimize possible delays in requirements implementation.

Another issue addressed during the interview was the amount of requirements used in the prioritization process. The list was composed by 45 requirements with different effort estimates. The participant argued that the number of requirements added additional difficulty to the process. We were told that the amount of requirements was an obstacle for a holistic understanding of the backlog, being prioritized, and that it was hard to finalize the prioritization because of number of requirements.
Finally, we discussed requirements dependencies. Since the list was composed of requirements of small granularity, dependencies between requirements had to be introduced into the list. This fact was not obvious to the person doing the prioritization. He argued that dependencies between requirements were far too technical for his comprehension and that would not be easily apprehended by other users. He reinforced that he, and other stakeholders, did not need to know what requirements composed each request but only the request waiting implementation.

When asked if his criteria would be different if only some of the requirements could be implemented the participant said that he would focus on requirements that, in his personal opinion, would create higher value and support business process core activities.

4.5. Specifying Learning

Based on the observation and interview we concluded that the process had to be redrawn to accommodate some constraints and remove some roadblocks to assure non-technical stakeholders participation.

The main conclusion was that using requirements as a prioritization artifact was not feasible due to non-technical stakeholders difficulty to understand them. Technical requirements can be very descriptive and, therefore, not grasped by eligible voters. The use of requirements also makes it harder to understand what were the requests being prioritized.

Another conclusion is that if the maintenance cycle does not have a fixed duration voters may change their prioritization criteria. The business project manager stated that having a limit on the quantity of requirements that could be implemented would have changed his prioritization criteria. Having fixed cycle’s duration enforces voters to focus their priorities in issues that are important to them. For example changes in the business process core activities support, in the business project manager case.

Dependencies were another main issue in the prioritization. Once again this is directly related to the use of requirements instead of requests as prioritization items. The use of requirements, as the objects to be prioritized, should be avoided at all cost since it is not a familiar concept to non-technical stakeholders and requires additional technical knowledge.

Finally the number of items being prioritize should be reduced. This will enable a better understanding of the backlog, understanding what are the issues being addressed and enabling an easier comparison between item’s priorities.
These conclusions led us to introduce changes into the prioritization process proposal:

- The use of requirements should be replaced by another concept that facilitates voters understanding and, therefore, voters prioritization;
- Changing the objects that are being prioritized will require a different estimation method. Since now a number of requirements will be aggregated into a higher-level artifact a holistic approach should be used;
- The concept of dependency should be eliminated. This will create constraints when building the list that has to be prioritized. Even requests can have dependencies between themselves. The use of some kind of feature that aggregates requests by common features is more suited to prevent dependencies.
- The number of artifacts being prioritized should be reduced to a manageable number. Voters will have a hard time trying to prioritize a large amount of artifacts.

Mike Cohn (27) suggests a sort of hierarchy to organize backlog requests. This approach can be useful to suppress the problem of granularity. Using stories in the voting process can prevent dependencies between requests, aggregates different request into categories of common aspects and offers to voters a high-level definition of the work to be performed. To prevent the disappearance of request registry, all stories used in voting activities, should have associated all the requests that compose them. This will enable story identification based on voter’s requests. This aggregation will also reduce the number of artifacts presented to voters and therefore reducing the amount of time needed to perceive the backlog state and the different issues being addressed.
5. Second Iteration

The second iteration was performed under a different scope, objectives and participants. The main goal was to test the entire proposal, assess user participation and compare results with the currently used method to execute the same task. The second iteration took action in the same company and for the same application. The definition of the environment context done in the previous chapter still holds, in this chapter.

5.1. Proposal

For this second iteration we made some changes into the original process based on the conclusions drawn from the previous one. In this section we will describe in detail what were those changes and what was the objective to be achieved with them.

The first change related to the duration of each maintenance cycle. Since prioritization strategies may vary accordingly to the size of the cycle or the lack of definition on it, fixed cycle duration should be set and communicated to all users. This means that only some of the prioritization items would actually fit into the final list and voters will have to create their personal prioritization strategies, based on their personal agendas.

The second change regards the items that are going to be prioritized. These items should translate into perceptive changes in the application being maintained, instead of development team requirements that may or may not be perceptible for end users. This change will enable a better understanding of what each prioritization item presented is supposed to achieve and how it will affect in the end the functioning of the application.

In order to operate this change, requests should be gathered and evaluated, as before, but their decomposition into technical requirements should be kept hidden from non-technical stakeholders. The organization of these requests should be done by putting them into stories. Stories are a number of requests that can be put into a usage scenario as a user story (32). Each story may enclose a number of requests registered into the application/product backlog. Since we already used request categorization, this change will not affect deeply the way how the backlog is managed.

The change from requirements to user stories will lead to effort estimations to be performed in a different manner and accounting for different factors. Estimating stories means that a number of requirements would have to be implemented in order to satisfy it. We suggest that a participative method, using all development team members, should be used. From our research we concluded that a suitable method would be the planning poker method (33). This method estimates each item based on the participation of every member of the development team and tries to reach consensus by arguing the estimations that each member does.
The use of user stories means that another of the problems identified in the previous iteration is also suppressed. The concept of requirement dependencies disappears with this introduction because of the scope of user stories. These stories represent a set of requests that relate to each other. Putting them into a single prioritization item means that they are all implemented or none of them is. For the final voter this is transparent and does not require a deeper understanding of how the application is built or how requirements relate to each others.

Finally, the introduction of user stories will significantly reduce the number of items to be prioritized. The aggregation of requests into a single user story will mean that the number of items, that previously were requirements needed to satisfy each requests, are now grouped requests based on the affinity that they have to each other.

These changes justify the appearance of a new activity into the process. This activity, called grouping, will be used to organize requests into user stories and estimate them. The activity should be the first activity performed when entering a scheduled maintenance cycle.

When entering this activity it is assumed that the current backlog will not change, this means that at grouping stage only requests present in the backlog will be considered. Requests will still be registered, but for prioritizations purposes they will be neglected.

User stories are composed of three parts (32):

a) A written description of the story used for planning and as a reminder;

b) Conversations about the story that serve to flush out the details of the story;

c) A list of all requests that relate to the story.

Earlier request categorizations should facilitate this activity. Each request was assigned to a single category upon registration. In this phase, and considering that backlogs can have hundreds or even thousands of requests, a simple validation of previous categorizations should be enough.

After grouping requests into stories the development team, combined with requirement engineering team, should perform effort estimation for each story. This estimation doesn’t need to be precise. The main objective is to make a rough prediction for voter’s information and to check if the story isn’t too big for the maintenance cycle. At this point special attention should be put into the size of the stories. If the story has an estimation that exceeds the duration of the maintenance cycle it should be divided into smaller stories that fit the constraints previously defined (33).

5.2. Action Taking

As in the first iteration, the second iteration was performed in two phases. The first one was the test itself. Voters used a custom made tool
that supported the process that we were trying to test. And a second phase where results were analyzed and some voters were interviewed.

**Prioritization Tool**

A tool, to support the process, was constructed. The solution used was implemented using Outsystems Agile Platform. The Outsystems Agile Platform was specifically designed to support software implementation using agile methodologies and the process suggested by it is based in the SCRUM methodology. It also has already implemented an agile project management tool and a sizing tool. The sizing tool is particularly useful for effort estimation.

The platform has also embedded a change technology that enables users to give feedback directly from the application to the developers. This fact facilitates acceptance testing and usage requirement, bug gathering and request registry.

This tool made possible to voters to participate anonymously and remotely. It offered a great deal of flexibility to users by providing a graphic representation of their requests and of the backlog.

**Participants**

For this iteration a larger number of participants were chosen, when compared with the previous one. In this iteration eleven users, with different roles, were asked to prioritize a number of requests. The group included three team coordinators, one of them the business project manager, the project owner and seven “end-users”. All of them had extensive application usage for the last year and were very much familiarized with application functionalities and limitations (for more information please refer to section 4.1).

The decision about the voters that should participate in the test was left to the business project manager. Not all eligible voters need to participate in the process. The voting group can be reduced accordingly to business project manager and project owner will, but still should be representative of the stakeholder population. In this case we have over 50% of the eligible voter population.

**Information Gathering**

To have a prioritization process we needed request gathering from users. Since, before the test, requests were registered by business project manager in his own way, the researcher felt the need to talk directly to users, in order to capture requests and business needs.

Requests were gathered from different sources. The researcher talked to at least to one member of each user type defined in section 4.1. From these talks a number of requests were gathered and stories were built.

From this gathering we were able to build ten stories. Estimations were made for each story. Each story had a description of the changes or additions that were represented, in clear and accessible language.
This change was a product of the first iteration, where we learned that requirements were not the best way to perform the prioritization. As suggested in our proposal, a collaborative estimation method should be used to perform this task. Unfortunately the development team had only one member, making the application of such a method impossible.

**Prioritization Process**

After information gathering and story building, the resulting list was published and voters were notified. A voting period of one working week was set for voting. During this time users could login into the voting application used and change their priority list as many times as they wanted.

Each voter had a graphic representation of his priority list and of the prioritization final result list. The final list was presented as the result of the prioritization performed so far. This means that each participant could dynamically see how his prioritization was affecting the process end result and what the community voting tendencies were.

Voters had also information about the number of hours available in the maintenance cycle, which could be allocated in this phase. The number of voting hours was calculated by the number of members in the development team, the number of days in the cycle and the number of working hours in a day. To this duration was then subtracted a percentage that is used to corrective maintenance. This guarantees that some time put aside to correct eventual bug occurrences during the perfective maintenance tasks.

To change the prioritization list the voter simply clicked on the respective arrow moving stories up or down according to their personal priorities. When changes were made, which resulted in changes in the final priority list, the application reflected them by changing the color of the line. Therefore the user knew which stories were to be implemented, before the voting period ended, by the color of the line of each story. If the line was green it meant that the story had collected enough votes to be one of the most voted and, therefore, one of the stories that would be implemented.

At the end of the voting period the prioritization list was withdrawn and no longer available for change, the final priority list was found. This list was then communicated to all voters.

Since this was an experimental process we did not get the necessary support from management to enforce the end result. This meant that the next phase of our proposal was not done. The final prioritization list was not implemented, in the next maintenance cycle, as it was supposed making impossible to test the constraints, rules and tasks that should be performed in the following process’ activities.
5.3. Evaluating

The prioritization results analysis and comparisons, with particular voters list, will be made in the next section. In this section we intend to talk about voter’s participation in the process and prioritization problems and challenges occurred during the experience.

Beside the interviews some statements can be made about the process result that we will describe now.

The resultant list of the test showed that ties can occur. This happens whenever two different stories obtain the same number of votes. If both stories have the same number of votes and they are the least voted stories of the ones selected for implementation, a decision has to be made. The proposal tested in this iteration did not accommodate this kind of situations and therefore had no kind of process to eliminate ties.

Considering the number of voters that were eligible for participation, which represented more than 50% of the total users, and the premise that supports this thesis, the participation fell short when compared to expectations. Only six of the eleven eligible voters actually performed the prioritization at least once. Another fact that is worth mentioning is that voters, that already had influence in the prioritization process before this test, were the ones that voted in larger number. From the six effective voters, one was the project owner, two were team coordinators, including the business project manager, and only three were users that did not have any kind of management responsibilities.

Interviews

In this section we analyze the end result of the test and compare it to key voters list. This will enable some analysis about how the process changes the implementation list, which is usually a decision made by a smaller set of people, and how they react to the change.

We conducted three informal interviews to assess user satisfaction about the outcome of the voting session, when compared to personal priorities, reasons for participation or non participation in the process and empowerment issues. The business project manager, the project owner and one “end-user” were interview. This choice was based on project owner and business project manager application responsibilities and the other “end-user” non participation in the process.

The interview with the project owner was done to infer if the process result would clash against business directives, process and application alignment and tactical decisions. During this interview, the project owner was asked to criticize the implementation list resulted from voting. He stated that the end was almost identical to his own personal prioritization and was therefore happy with the result. He pointed out though that the control that he has over application decisions can only be challenged to some extension. The perfect example was the inclusion of a story that was composed by CEO’s requests that supported a business process change. The project owner stated that no matter what the outcome of the
prioritization process would be that particular story had to be implemented as soon as possible.

The business project manager had even more stories elected. Only one of the stories that got elected was not in his personal list. Despite this fact he suggested that some control should be given to the participants that usually do the prioritization. This argument was justified by the need to establish some direction into the application development planning.

Another main issue was the low level of participation. Since we advocate that it will be of benefit for application development and evolution bringing most of the, usually left out, users to the prioritization process, we wanted to understand why this part of the voting population had such low levels of participation and little expression in the end result.

When enquired this user argued that the experience had no kind of repercussion in the actual implementations that were scheduled and, because of that, had little interest in participating. The fact that the company was going through substantial changes, in department organization, did not help either. When asked if he would be interested in participating in this process, if its result was used as an effective maintenance cycle implementation plan, the user replied that he would be very interested. He pointed out that he had some ideas that he would like to see implemented and because he felt that he had some knowledge about application usage that might contribute to improve the project's usability level.

5.4. Specifying Learning

This iteration brought to light so issues about process control and change management that has to be considered and analyzed.

First of all a minimum quorum of voters should be required to validate the prioritization process. This means that, if a percentage of designated voters do not exercise that privilege, the process resulting priorities list should not be considered. An incentive/punitive policy might be used to motivate designated voters to actively participate in the process. The quorum may vary from project to project but, in the end, at least the stakeholders that used participate in the previous prioritization process should have exercised their voting privileges.

A second recurring issue was the lack of control reported by both project owner and business project manager. Each of them had different issues that related directly with the necessity of exercising some kind of control into what gets to be voted. From the interviews performed we concluded that two additions should be made to the process.

The first change refers to the necessity of having some mandatory implementations, which are decided by the project owner and, possibly, the business project manager. A new process activity should be created where those two actors decide, up to a portion of the total duration of the cycle, what has to be done. This decision may use another prioritization method, such as Kano Model Analysis (see section 3.3.5). This new activity will grant some decision power to the traditional participants in
the prioritization process by granting them the power of planning a percentage of the maintenance cycle without other voters' interference.

The second change should be operated into the newly created activity. To direct the application into some specific path of evolution the participants of this activity should also decide which set of stories can actually be voted. To enable this we recur again to Mike Cohn's requirements categorization (27). Until now requests would be gathered in stories and stories would be published for voting. With this change we will add another rank into the request hierarchy that aggregates stories. This new rank will symbolize different subjects to which the application can evolve. Gathering stories under this categorization will offer business project manager and project owner the ability of restricting which subjects should be the focus of the next maintenance cycle.

This new process feature should be approached carefully, including many control tools will transform a process that is supposed to be open into a process that differs too little from the approaches available until now.

Regarding voting ties, a new tie-breaking system should be created to resolve recurring ties. Ties can occur when two stories have the same number of votes. In these cases it may be required a decision regarding which of them comes first in the priority lists. Some kind of secondary priority system may have to be used.

Finally we concluded that, to implement this process, change management actions should be put into practice. This subject is not in the scope of this thesis but we can still identify two different points of action. Ensure that non-prioritizing stakeholders are actively interested in participating in this prioritization process and in generating requests. Ensure their participation and that they are committed to the process. Work with business project manager, project owner and other stakeholders, which usually prioritized requests, should be done to offer a guarantee that the process does not strips from the control from tactical decisions and business process changes, supported by the application, but instead offers a fresh perspective that can add additional value to the application.

Considering the problems identified in the previous iteration regarding the number of items being prioritized and the language used to describe them, we observed that in this second iteration, no additional explanations were needed for voter's elicitation. This leads us to conclude that the changes in items description and scope helped clearing the objectives of each item being prioritized and reduced problems with voter's confusion.

Adopting stories, as the prioritization item, enabled a reduction of the items being prioritized, when compared with the previous iteration. In this iteration we had thirty three technical requirements that were distributed over ten stories. This was helpful on reducing the prioritization voting activity completion times. From observations we concluded that each voter took, no longer, than five minutes ordering the items accordingly to their own personal priorities.
5.4.1. Process Impact

In this section we will describe the overall evaluation about the process. This evaluation was done to understand the organizational changes that occurred, due to process implementation, and not changes that should be done into the proposal.

This evaluation was done based in two different information sources. Two questionnaires were given to the participants of the test described in the previous chapters. The first was given before the test started and was used to characterize the tested population and to assess the overall satisfaction with application developments. The second one was used to assess the impact of the test and determine changes in the application maintenance process perception and overall satisfaction with the proposal.

The second source of information was the direct observation performed by the researcher during his field work and interviews conducted during both iterations. This observation and conversations with different stakeholders are a valuable source of information that will help to better understand weaknesses and strengths of this proposal in a corporate context.

The tests made were performed on a controlled environment. Voters knew, before the tests, that the end result would not affect application maintenance plans. This made impossible to test acceptance testing and implementation activities. The evaluation that we have made concerns only with activities performed prior to request implementation. A final remark should be done to the validity of the results. Since the process was not fully implemented and participants knew that the changes operated would not affect the outcome of development tasks, some of the results may be somewhat different of what would happen if that was not the case.

Questionnaires

As said before, two questionnaires were made. In the first one we assessed the perception that the test participants had about the perfective maintenance worked performed until then.
In Figure 9 we can see that users recognized improvements, in the application, in the post-production phase. Despite this fact, when asked if they had any request that they would like to see implemented the answers were mainly yes (see Figure 10). Some of these answers were unexpected because the development team had no idea that so many people had pending requests. This was caused by the lack of communication between users and development team since all requests were given by the business project manager. This situation does not stimulate user participation in the application evolution because it grants them little recognition.

The other issue identified was the lack of knowledge, more than 50% of responses, of requests implementation costs, in man-hours. The absence of perception is caused by the lack of participation in the maintenance process.

Viewing maintenance from a service perspective view makes it imperative that “clients” have knowledge about service levels. The
questions made after the test were constructed to assess if the issues detected, in the first one, were somewhat suppressed.

Unfortunately the number of voters that actually participated in the process was less than the one expected. In the previous section we analyzed the results and took conclusions about it. The main objective of this thesis was to create a process that empowers users. In the tests conducted we found out that users do not feel that need, or else they would have participated, or did not take the test seriously. The second reason pointed out may be related to the prioritization process itself. Since implementation of the final voting list did not follow the voting stage, users probably did not felt that it would worth the time that they would spend prioritizing the stories presented.

Disregarding the low participation and focusing on the voters that actually participated in the test we conducted a final questionnaire to check for changes in the process view and empowerment issues.

We focused the last questionnaire on empowerment changes and voter perception about implementation costs.

In Figure 11 we can see participant’s responses when asked how the use of this prioritization method would affect their participation in the process. As we can see only one person responded that the use of this process would decrease his decision-making power. We do not know who has given this answer, because the questionnaire was anonymous, but the interview, after the second iteration, that the researcher had with the project owner made clear that it was one of his major concerns.

Disregarding the one negative answer to this particular question, we can conclude that the application of the proposed prioritization method, on this particular case, would make users, which do not usually
participate in the prioritization process, feel more empowered or not affected by the change.

Another issue accounts to the participant’s perception of development times, the goal was to make more transparent to all users the effort associated which item being prioritized and the number of hours that each maintenance cycle has. This is particularly important due to the approach taken in this thesis. Maintenance is a service and therefore has to have service level management and auditing capabilities. Items’ estimation offers information about service expectations and prioritization list offers information about service deliverables.

When we asked users if the process helped them to better understand the duration of implementation all of them answered yes. This is a positive impact of the test realized. Knowing development times, implementation deliverables, offering participation in the prioritization process and giving information about the backlog status makes the process/service completely transparent to everyone that takes some part on it.

The fact of the entire voting activity is done in an anonymous way helped voters to prioritize without any kind of pressure from upper manager. This does not mean that the pressure was explicitly there before but using this technique voters were able to express their own opinions without feeling that they were going against anyone else’s opinion.

Voters that usually did not participate in the prioritization process felt empowered, by this process, and that they were actually taking part on the decision of what was being implemented. Voters that usually had participation on the prioritization process felt the opposite. They stated that the process implementation would reduce their control over developments and jeopardize tactical decisions and process definition or reengineering. As said before, accommodations should be made to mitigate this feeling.
6. Result

In this chapter we will construct a final proposal based on the research performed, the initial proposal and the actions taken. This proposal includes changes made to the initial proposal based on the conclusions that were drawn during the two iterations performed.

Some of the activities did not suffer a lot of changes but, nevertheless, a formal and complete definition of the process should be done. We will state again the research objectives and describe in more detail the activities that define the process itself.

Remembering which the objectives of this proposal are:

- Empower application users that usually do not have a say in the prioritization process;
- Publish and inform everyone of what is being done what can be done in terms of application development;
- Make clear to all users what the cost of their requests is. If application management is a service we need to create some sort of service level management and service level agreements that can be audited by “clients”;
- Increase the business value of the application maintenance cycles by creating a prioritization method that enables a better management of the backlog;
- Create a method of prioritization that increases the value of each maintenance cycle through more efficient decisions about which requests should be implemented.

As said before we recognize three different types of requests sources:

1) **Business Process Requests**: typically, these are requests that are defined by process owners and higher management levels. They relate to process reengineering and process implementation tuning.

2) **Usage requests**: these are requests that end users produce based on the utilization of the application. The objective here is not to add function, or at least no usually, but to improve the user experience quality.

3) **Bugs**: bugs are requests that report faults or unexpected behaviors in the utilization of the application.

Because we have different request sources we will have to define different maintenance cycle parts. Each cycle implementation can be divided into three different parts:

- Tactical decisions implementation;
- Voting winner stories implementation;
- Corrective maintenance tasks.
For an efficient request prioritization a request backlog has to be in place. This backlog should contain all requests done by stakeholders.

![Prioritization Process View](image)

**Figure 12 - Prioritization Process View**

This definition does not mean that business process requests have to be implemented in the tactical decisions implementation part of the maintenance cycle implementation or that a usage requirement has to be implemented in the voting winner's implementation part. The value of each request will be taken into account when decisions are being made for each part of the cycle. This means that a usage request can be valued as very important and an interesting addition to the business process definition and therefore implemented as a tactical decision.

Corrective maintenance tasks are different. This maintenance cycle implementation part is used to accommodate unpredictable events on the application. If some bugs are reported during the maintenance cycle iteration guarantees have to be made to ensure that the plan does not deviate and the bugs are still fixed.

The result process of our investigation comprises six activities with a set of artifacts and rules. The complete process view can be seen in Figure 12.
For each part a percentage of the total time available in the cycle implementation activity should be allocated. The actual amount of man-hours given to each part of the activity is calculated based on the team size and the cycle duration. For example, considering a team of 10 developers with a working day of 7 hours for a cycle of 30 working-days we will have the following cycle duration:

\[ \text{Implementation Duration} = 10 \times 7 \times 30 = 2100 \]

The result will be the number of hours in a maintenance cycle implementation activity. So for any given maintenance cycle, the number of hours available for implementation can be calculated as follows:

\[ \text{Number of Developers} \times \text{Working Day Hours} \times \text{Days in the Cycle} \]

Each implementation activity, as said before, has three different parts that compose it. The number of hours that each of them can consume is calculated by assigning a percentage of the cycle to them:

\[ \text{Hours for Tactical Decisions} = \text{Duration} \times \text{Tactical Decisions Percentage} \]

The same equation should be applied to calculate the number of hours assigned for voting winner stories implementation and corrective maintenance implementation. This way the development team, and everyone, that as any kind of participation in the process, knows what is the time available for each different part.

Beside the three parts already described, some extra time needs to be allocated to other tasks that must be performed. It is not arguable that implementation would account for the biggest portion of the maintenance cycle, but voting, grouping and acceptance testing also consumes resources and time. Deciding how much time is given to the development team to group request should be done as soon as possible. This value should be the same in all cycles.

Voting and acceptance testing activities should also have fixed and limited amounts of time on which they have to be performed. It is of the utmost importance that these limits are public and passed to participants. If not user participation in the voting activity or author’s acceptance testing may be compromised.

We count, to the overall duration of the maintenance cycle, the time spent on tactical decisions. This is an activity that should not take more than half a day of work and does not consume significant human resources. Nevertheless the process cannot continue without it so it should be also accounted in the overall duration.
Adding these limits to the maintenance cycle duration will set a formula to calculate how long an entire maintenance cycle lasts. The resulting duration of a maintenance cycle can be calculated as follows:

\[
\text{Maintenance Cycle} = \text{Grouping} + \text{Voting} + \text{Acceptance} + \text{Implementation}
\]

In the following subsections we will discuss in more detail each part of the maintenance cycle and what kind of activities have to be performed on each of them.

6.1. Request registry

Request registry is an asynchronous activity. It does not stop when the process enters the other phases. This assures that requests made during the prioritization process, implementation, testing or acceptance, are still gathered into the backlog for future maintenance cycles.

Registering requests can be achieved in many ways and by many means. The basic principle is that all requests should be registered with the following information:

a) A detailed request description;

b) The request author (user, customer or other);

c) Request category.

A request detailed description will be used for voters’ clarification during the voting activity. A good and clear definition will help voters understand clearly which goal, feature or process is underlying the request. A clear definition is a cornerstone for the whole process. Without it, the voting process can be challenged or the final result inaccurate due to a subjective interpretation of the request meaning. If the requirements engineering team has any doubts about a request it should contact the author for further elicitation.
Keeping information about the request author is important for request implementation validation. Each request, at the end of the implementation, requires acceptance tests that should be performed with the author of the request.

Finally, each request should be put into a category. Request category identifies common characteristics between requests. The categorization should be done using technical parameters, such as entities usage, design patterns or application features and will be useful for time saving in the grouping activity.

The request is registered into the request backlog (see Figure 13). This is a repository of all requests registered. This repository will be used if future activities to determine which requests are implemented in any given maintenance cycle. Requests that haven’t been registered in the backlog do not exist for maintenance cycle decisions purposes.

In the following subsection we describe the artifacts used in this activity.

### 6.1.1. Artifacts

Two artifacts are consumed and produced during this activity:

a) **Requests**: The request is the source of information that builds up a maintenance cycle. It comes from users, project owner, business project manager, clients or any other source that relates to business;

b) **Request Backlog**: The request backlog keeps a registry for all requests and their associated information. Beside the information described in the previous section, each request should have an associated state. This state is used to determine if the request has been implemented or not.
Periodically should be taken a “snapshot” of the request backlog. This snapshot will be consumed in the next activities of the process.

6.2. **Grouping**

When entering this activity it is assumed that the current backlog will not change, this means that at grouping stage only requests currently present in the backlog will be considered. As we said in the previous section, requests will keep being registered, but for current prioritization purposes they will be neglected.

The main objective of this activity is to build stories. Stories describe functionality that will be valuable to either a user or purchaser of a system or software. User stories are composed of three parts (32):

a) A written description of the story used for planning and as a reminder;

b) Conversations about the story that serve to flesh out the details of the story;

c) A list of all requests that relate to the story.

Earlier request categorizations should facilitate this activity. Each request was assigned to a single category upon registration. In this phase, and considering that backlogs can have hundreds or even thousands of requests, a simple validation of previous categorizations should be enough. To better accelerate this activity stories should be built based on request categories and therefore a story should contain only requests of one category.

This activity is represented in Figure 14. It defines a number of tasks that have to be performed in order successfully accomplish the activity purpose.
When entering this activity decisions should be made about the percentage for tactical, corrective maintenance and voting implementation that compose the implementation activity duration. This decision should only be done when the backlog is analyzed and the business project manager and project owner fully comprehend the work to be done. Each of these should have a part of the cycle allocated. The percentages can be adjusted from cycle to cycle reflecting process learning and necessities specific to the state of the asset being maintained. Remember that a fixed amount of time has been set for this activity. Efforts should be made to comply with time constraints. In case of recurring delays the time limits should be revised.

As said before, each maintenance cycle should not have a duration that exceeds 30 days. This duration is based on SCRUM methodology recommendations (34) (35) (36) (37) (38). Any cycle that exceeds that duration should be seen as an exception.

Requests should be grouped into stories. Stories are groups of requests that relate to the same feature. It is a broader definition of request that identifies requests familiarities and defines usage stories for them. Based on request categorization the stories are built by describing a particular feature usage. This description should depict every request that the story fulfils.

After grouping requests into stories the development team should perform effort estimation for each story. This estimation doesn’t need to be precise. The main objective is to make a rough prediction for voter’s
information and to check if the story isn’t too big for the maintenance cycle. At this point special attention should be put into the size of the stories. If the story has an estimation that exceeds the duration of the maintenance cycle it should be divided into smaller stories that fit the constraints previously defined (33).

In our proposal we suggest that a story should not exceed 40% of the maintenance cycle time reserved for implementation tasks. This excludes corrective maintenance time and time reserved for tactical decisions (for more information please see proposal introduction). This restriction can be put into the following condition:

\[ \text{Story Estimation} < \text{Voting Winner Stories} \times 0.4 \]

Each story should hold this condition as true. In case the story fails the condition it should be divided in smaller ones with smaller estimations. For more information about how to calculate the number of hours allocated to voting winner stories please see the proposal introduction.

The estimation method itself is a research topic that falls out of this thesis research topics, still one method suggested is planning poker (33). This method is based on an open discussion between development team members. For each story the team hears a description of the story and selects and estimation card from its personal deck. Afterwards all team members show their card to the rest of the group at the same time. If everyone selects the same card the story estimation is written down, if not a brief discussion follows, with all members required to participate, until a consensus is reached.

After story estimation and validation, another kind of grouping should be done. To identify different paths on application development, stories should be grouped into themes. These themes represent collections of related stories (27). Themes will be used in the tactical decisions activity to offer additional control, to business project manager and project owner, over the application’s maintenance cycle.

The end result of this activity is a list of stories that aggregates all valid and categorized requests. Each story has a list of requests that compose the story. This list should be available for voters’ consultation.

6.2.1. Artifacts

The artifacts used in this activity are:

a) **Request Backlog**: The “snapshot” of the request backlog used to maintain all requests.

b) **Stories List**: List of all stories produced during this activity and their respective estimation. The stories should be grouped by into collections that form themes.

The list of stories is produced during the course of this activity. It will be used in the following activities to plan maintenance and decide which requests are going to be implemented.
The request backlog is used to build stories. As said before, each request belongs to a single story.

6.3. **Tactical decisions**

This activity relates to the need of alignment between business processes and software systems and applications. Since the main goal, with the process here suggested, is to include all end users and/or custumers in the prioritization process it is important that business process decisions are kept under traditional decision-makers domain.

In this activity the project owner and business project manager are briefed on all stories produced in the previous activity. A detailed description of each story is given and all doubts are cleared. After that we use a Kano Model Analysis to identify the most important stories for these particular stakeholders (see section 3.3.5). This special prioritization is done to guarantee that all tactical decisions related to process reengineering or process definition are performed despite user/customer decisions.

The participation of project owner and business project manager are critical at this phase to guarantee that some control is still on hands of the traditional decisions makers. It prevents, for example, in corporate applications, employee's decision about changes on critical business processes, which usually are a responsibility of higher management positions.

As stated before a percentage of the overall cycle time should be set to the decisions made in this phase. The limit should be mandatory to guarantee that decisions made here do not occupy the totality of time allocated to maintenance cycle.
The grouping activity defined the percentage of time, in the maintenance cycle, that can be allocated to tactical decisions. No matter what the number of “must have” stories arises from the tactical decisions activity this percentage should not be exceeded. It can happen that the percentage of the cycle allocated to tactical decisions isn’t enough to accommodate all “must have” stories identified, in these cases the decision, of which of the “must have” stories are implemented, should be left to business project managers and project owners. Make notice that deciding which “must have” stories are implemented does not mean that the overall percentage allocated for tactical decisions can be changed. The selection of the stories may only occupy the totality of this time or less. If it occupies less than the overall time available, the remaining time should be allocated for the implementation of voting winner stories.

For a complete activity view see Figure 15.

If the number of “must have” stories does not fulfill the entire time reserved for tactical decisions, the remainder should be added to the time available for implementation of stories that would be voted in the next phase.

Afterwards another decision has to be made, by the activity participants. To control evolutionary application’s paths, business project manager and project owner should decide which of the themes should be used in the voting activity. This is done to guarantee that tactical decisions about how the application should evolve and subjects the application maintenance should be focused are still made by people that own and control the business process.

This particular activity should be performed in a small amount of time, when compared to other process activities, due to the participants involved and physical participation requirements that have to be assured.
In the following subsection we define the artifacts used and produced during this phase.

6.3.1. Artifacts

In this activity we define four different artifacts:

- **List of percentages**: The list of percentages allocated to each part of the maintenance cycle implementation activity;
- **Story list**: list of all stories available for implementation. Each story should have the requests, which compose it, associated and the respective effort estimation;
- **“Must have” list**: list of stories that have been selected as tactical decisions by the project owner and business project manager. This list may contain more stories than the ones that are going to be implemented. The stories selected for implementation should be marked;
- **Theme List**: this is the list of all themes selected by business owner and business project manager that can be voted. Please remember that each story belongs to theme.

The first two artifacts are consumed during this activity and the last two are the produced during this particular phase.

6.4. Voting

All activities done until this point were preparations. At this point any story, that has not been select to implementation in the tactical decisions activity and belongs to a select theme for implementation, is going to be present in the list presented to voters.

This activity should include the participation of most of the end users and of all business staff that has some kind of participation in the project development or process support and definition. A voting period should be set, on which all voters can assess the voting tool, and voters should be informed that the story list is available for voting.

There are a number of different artifacts used in the voting activity that should be understood before we explain the method itself. In the following subsection we will define the concepts and, afterwards, define the prioritization method used in the voting activity.
6.4.1. Artifacts

In this particular activity we define four different artifacts to be used. Each of them performs a different and important role in the process that it is important to clarify:

- **Voting list:** This is the list of stories that will be presented to voters. In this list should be all validated stories, with their requests associated, that haven’t been selected in the tactical decisions activity for implementation and belong to the themes selected for voting;
- **List of percentages:** The list of percentages allocated to each part of the maintenance cycle implementation activity;
- **Personal List:** Each voter keeps a personal priority list, during the voting period, with all stories prioritized accordingly to each personal priorities;
- **Final List:** This list represents, at any given point during the voting period, a prioritized list of all stories according to all participants’ votes with all stories that are current voted for implementation marked.

The voting list is used to set the initial personal list of all voters. Each voter is presented with a list of stories, in which themes not select during tactical decisions activity and “must have” stories are left out. The personal list is created for each user as soon as the first prioritization is performed by the voter. This list is updated every time the voter changes at least the priority of one story.

The list of percentages is used to determine which of the voted stories fit into the maintenance cycle available time for voting winner stories implementation. These values are used to calculate how many of the available hours are occupied.

The final list will be the end product of this activity. This list includes all stories, ranked from most voted to the one with the least votes, and with the most voted stories of the list, that fit into the maintenance cycle implementation timeframe, clearly identified.

6.4.2. Method

The voting method is a combination between 100 point estimation and theory-W negotiation method (see sections 3.3.3 and 3.3.4). The priority assigned to each story represents a number of votes casted on that particular story. Keep in mind that each story has a number of requests associated.
The voting process, depicted in Figure 16, should be transparent to each participant. This means that at any given time the voter has to have the ability of checking the current overall state of the voting process and which stories are, so far, the most voted ones. Viewing the current state of the final list gives them the ability of prioritize accordingly to their personal agendas, but since each priority has a fixed number of votes associated even the highest priority of a single voter won’t be enough to make sure that a story is elected. Votes should be assigned to each story based on the place that they occupy in the personal list of each voter. This means that the stories with higher priorities will be the ones that actually have the fewer votes.

There should be a clearly set and communicated time interval where the voting may occur. Only during this period should voters be able to define their personal priority list.

When voting, two or more different stories can finish with the same amount of votes. This situation can create problems deciding which of the two, or more, stories are going to be implemented. If there is only place for one more story in the maintenance cycle implementation, a decision has to be made. In those cases the stories should be ordered according to the number of voters that put each story as its number one priority, if the tie still persist the same inspection should be done for the second place and so on. If all tied stories have the same number of voters placing them in each rank a quality voter should be used as a tie-breaker. In this case a pre-decided tie breaker voter should decide the priority of this story. This is easily achieved by checking the rank of each story on the voter personal list.

![Figure 16 - Voting Activity](image)
Combining all these constraints will result in a process were voters make an informed decision, because they can see the final result and how their prioritization affects it. It also obliges compromises to push votes' personal agendas and uses a voting system based on quantifiable metrics were they can readjust their priorities during the process.

The constraints, here described can be summed into the following list:

a) Each priority rank, or the position in the priority list, has a fixed number of votes associated;
b) The state of the overall prioritization result should be visible at all time. This means that the most voted stories are clearly identified at any given moment and can be recognized by the voter.
c) None of the voters can have access to the personal lists from other voters;
d) Each voter must see how his prioritization impacts the final list state;
e) A voter can change his personal list as many times as he wants during the voting period.
f) The final list cannot have stories with the same number of votes. In this case tie-breaker criteria should be used.

These constraints are the set of rules that define our proposal. The previous activities, described in the previous subsections, are a preparation of artifacts and people, to the voting phase.

It can be considered that the prioritization is only performed in this phase, but it is our belief that previous activities are a big influence in the success of the voting activity. Remember that early decision about “must have” stories and themes eligible for voting have already been done, limiting the number and type of stories presented for voting.

This proposal is based in voter's participation. For each application the types of voters that can be selected may vary, however, this process was created assuming that voting should be done only by people that own, participate or have a deeply understanding of the business process. Because only the stakeholders (see chapter 2 for a clear definition of stakeholders) can properly prioritize the requirements, while only the developers can estimate the cost and schedule consequences of stakeholders' priorities, this choice brings the advantage of having stakeholders deciding what gets done first and developers deciding how long will it take (39).

All stakeholders can be eligible voters but the process supports a smaller set of voters. It should be kept in mind that this process advocates that the biggest the voting group the better will be the prioritization results.

Eligible voters are not obliged to participate in the voting process, but a minimum quorum should be set. If a minimum number of voters do not participate the resulting priority list should not be considered for implementation and alternative methods should be used. The minimum quorum may vary from case to case but the number of people that
actually vote should never be smaller than the number of project owners, business project managers and key-users identified.

To increase voter’s participation a number of policies may be used. A simple solution is impeding non participative voters of voting requests for the next maintenance cycle. If those particular users have any request being voted for the next iteration this will remove their ability of lobbying, as far as the process lets, for their own requests.

Voting should be performed in a centralized base and enable remote participation. The use of tool that reads product backlogs and constructs sprint backlogs becomes essential. The story voting activity should be done in an asynchronous way enabling each user to access to the voting tool remotely, at any time and as many times he wishes.

6.5. Implementation

Implementation techniques are not part of the scope of this thesis. The end result of the process, here proposed, can be seen as the requirements list of any given project with any given implementation methodology or technology.

The actual list of requests that should be implemented during the cycle is defined by the “must have” stories marked for implementation, during tactical decisions activity, and the most voted stories, during voting activity, that still fit into the time available in the cycle.

In following subsections we will describe some events that might influence the correct implementation of the prioritization made earlier.

6.5.1. Corrective maintenance

This thesis has its main focus on perfective maintenance however corrective maintenance may be needed while performing perfective maintenance implementation. When calculating the number of man-hours available in maintenance cycles (see section 6.2), a percentage should be allocated to corrective maintenance tasks. If this time is not consumed with these tasks it can be considered as if under-estimated stories were present. This event is described in the following sections.
6.5.2. Stories were under-estimated

If all planned implementations end before schedule and the cycle has not yet ended extra story implementation can be done. The decision of which story gets implemented can be done in four ways:

a) The story with more votes of the non-elected stories is selected for implementation;
b) The story with more votes of the non-elected stories that has an estimation that fits the time available is selected for implementation;
c) Decision is made by project owner and business project manager regarding which stories are selected for implementation;
d) The implementation activity ends before the deadline.

The difference between the first two choices is that in the first we have no guarantee that the story selected will be implemented in the available time. The most voted story of the non-elected ones may have an estimation of 10 days when the maintenance cycle has only 5 days left, for example. If this was the choice, accommodations should be done to prevent that the cycle duration does not exceed the initial planned one. Over-estimated stories are described in the following subsection.

The second option gives more assurance about completion within maintenance cycle intervals. Since the story has an estimation that fits into the available implementation time left the risk of having this story take more than the time left in the cycle is severally reduced. Nevertheless over-estimated stories can occur and in those cases strategies should be defined. Also, the story that fits the available time may be the less voted one and, therefore, the one that brings less additional value to the application.

The third choice is to ask the business project manager and project owner which story they want to see implemented in the current cycle. If “must have” stories, from the tactical decisions activity, were left out of and were not one of the most voted ones in the voting activity, business project manager and project owner should decide which of the “must haves” they want to see implemented. It should be kept in mind that these stories may have estimations larger than the available implementation time. If such a case happens, it can be considered as an over-estimated story. Over-estimated stories are described in the following subsection. The choice should, nevertheless, be left to the stakeholders with those specific roles.

Ending the implementation activity before time can also be an option. In this case some disturbance may be caused in other activities, running simultaneously, or to other business processes that depend on this one. For example, users may not yet be available for acceptance testing.
6.5.3. **Stories were over-estimated**

Stories that are over-estimated may create a situation where the initial man-hours allocated to the maintenance cycle aren't enough to fulfill the current implementation requirements. This event can be dealt in several ways. Each implementation methodology has its own way to deal with the problem. In the end two decisions, with different implications, can be made:

- Extend the deadline for implementation delivery;
- Remove the less voted story that hasn't been yet implemented from the list.

The first option goes against one of the principles of this method. Implementation duration's should not be changed after the beginning of this activity.

The second option may be better. Since some stories have not yet been implemented, over-estimations may have occurred and therefore the implementations times may, in the end, fit into the cycle duration. Removing one story will mean that some maneuverability is achieved. In the end if some time is left in the cycle the removed story can be put back into the implementation planning.

6.5.4. **Artifacts**

For this activity we can consider to be present three different artifacts:

- **Priority List**: This list is the result of the previous activity. It includes all stories prioritized accordingly to the votes of all users. The list should be ordered from the most voted to the story with the one with the least votes. Stories selected to implementation are the most voted ones, as long as they fit into the maintenance cycle available time. This list should, nevertheless, have all stories that are supposed to be implemented marked.
- **“Must have” list**: This list is the result of the application of the Kano Model Analysis performed in the tactical decisions activity (see section 6.3).
- **Implementation list**: This list is the report of the maintenance cycle. At the end of the cycle a list, with all implemented requests, should produced for backlog update purposes. Remember that each story has a set of request associated.

The priority list and the “must have” list are used to determine which stories are going to be implemented. All stories in the “must have” list that are marked for implementation and the most voted stories from the priority list that fit the remaining time of the maintenance cycle define the
list of stories, and subsequent requests, that are going to be implemented.

The implementation list is the list of request that actually got implemented. Because of delays or anticipations in the initial planning, not all stories initially set for implementation may have been actually implemented or more stories than the initial prediction may have been implemented.

6.6. **Testing**

Testing is the final activity of the process. In this phase request authors will be asked to validate implementation. For each request implemented in the previous activity the request author will be asked to validate the implementation. When this happens three cases may occur:

a) The user validates asserts that the request was properly implemented;

b) The user stated that the request was not properly implemented;

c) The user isn’t available for acceptance testing.

If the author of the request accepts the implementation the request should be marked as implemented and the backlog list should be updated. If the author isn’t available for acceptance testing, the same action should be done. This prevents delays on the delivery of the maintenance cycle developments that may occur due to requests authors’ absence. If the author does not accept the implementation two situations might occur:

a) The implementation does not comply with author wishes at all;

b) Small corrections are needed.

If the implementation does not comply with author wishes the request should be marked as not implemented. In this the request back in the backlog and eligible for voting. If small corrections are required it should be created a corrective maintenance request with the alterations need to achieve compliance and the request should be marked as accepted. This activity is defined in Figure 17.
Acceptance tests should be performed, in a preset amount of time, for each of the requests, not stories, since a story may have more than one request and more than one author. This phase does not require a formal testing suite. At this phase request authors should be informed that the request was implemented. After being informed the author has a limit amount of time to test the request implementation and report back his decision.

The artifacts used and produced in this activity are described in the following subsection.
6.6.1. Artifacts

In this activity we consider two artifacts:

- **Implementation list**: this is the list of all implemented request in the previous activity;
- **Backlog**: this is the list of request waiting implementation after the acceptance tests. The backlog needs to be updated in case the request implementation is accepted by the author or if the acceptance tests resulted in a corrective maintenance issue.

The implementation list is consumed during this activity and the backlog will be updated as a result of this phase.
7. Conclusion

This thesis was concerned with managing demand in information systems. Through the investigation performed, we discovered that up to 80% of the overall cost of information systems is related with maintenance tasks. Adding to this, applications spend most of their lifecycle in this phase and it’s on this phase that applications suffer changes that could not be predicted at the beginning of their implementation.

Applications are dynamic entities that have to accommodate changes through their lifecycle in order to cope with business challenges and usability requests. Managing demand is, because of that, a problem that occurs mainly in maintenance. In this phase development teams have to manage different requests, coming from wide range of sources and bugs. Requests and bugs have to be compared and prioritized against each other no matter the source or purpose.

Through the life of an application, a number of changes may occur either in scheduled intervals or due to unpredictable factors. No matter what the reason was, application maintenance can be viewed from a service perspective. This service, as any other kind of service, has to be audited, have associated metrics and be managed.

Managing demand in maintenance is the process that decides what should be implemented first. The problem in information systems is not finding something to do but instead decide what to do first. Demand exists, in information systems maintenance, in the form of change or implementation requests, and supply is the available resources to satisfy these requests.

When demand overcomes supply, accommodations have to be made in order to register and manage all requests. This can be achieved through an application or product backlog. Prioritizing this backlog is the way to manage demand.

Several prioritization methods exist already but the problem still persists. Prioritization is made by traditional stakeholders, such as project owner, business project manager, key users and technical staff. We argued that all business actors should be involved in the prioritization process, since there are the ones that understand the business and use the application.

We created a process that enables the participation of as much actors as we want and that makes everyone accountable for their participation and responsible for improvements in the application. Empowering users that usually do not have any kind of participation in the prioritization process has shown to be an effective way to obtain higher user satisfaction and improve user perception of the backlog state and implementation times.

Quality of service of maintenance can only be measured if “clients” have available metrics. The process defined in this document offers information about which request are implemented, the number of requests in the backlog and associated implementation times.
The tests performed have showed that the process can perform an effective prioritization and, at the same time, transform the decision process transparent to all participators. Results show that user satisfaction increases with the process implementation through user empowerment and accountability. Users feel involved and are invited to suggest changes, bringing a fresh perspective to application development.

The process also showed a decreased prioritization time by making discussions impossible to happen and turning the whole process increasingly dynamic. Voters can change their priorities and negotiate trade-offs without never talk with anyone else. A clear view of the prioritization state enables a silent negotiation with a visualization of the impact made by each personal prioritization.

Research has also showed that user commitment to the process can be a challenging factor for a successful implementation. A clear understanding of the process advantages and capabilities should exist before it is implemented.

Another challenge was the preoccupation of business process owners and business process managers about the control that they may lose when using the proposed prioritization method. In the last iteration we performed some changes to the process with to mitigate these concerns. Unfortunately we did not have the time to test them.

Because of user commitment issues and business process control a change management process should be performed before the prioritization process implementation. A collaborative environment focused on application and business process should be created, where everyone is conscious of the challenges and tasks in hands.

Finally, the proposed process is iterative but because we could not complete, in none of the iterations, a full maintenance cycle, we cannot determine if activities from different cycles should be overlapped or not. This may be justified specially in the voting activities where the development team has to wait for the results. If this particular activity takes a significant amount of time, or else voters will not have enough time to negotiate, the development team may fall in a hiatus while waiting for the end of the activity.

7.1. Future Work

The work is not close to be finished. This kind of process has always room for improvement and needs to evolve with experimentation. It is not yet a mature process with enough concrete experimentation.

All test conducted were performed in corporate environment using real data, from a real application, and using real business actors. Still the tests were performed in a control environment where we could not test the whole process and where participants knew that process results would not be of consequence. Testing the process only with one application is not enough to extrapolate results for all applications or organizations. A more heterogeneous set of tests should be done to verify proposal applicability.
From our research we suggest the following ideas for improvement and testing:

- Improvements made at the end of the second iteration should be verified through testing;
- Scalability tests should be done by increasing the number of participants and stories being prioritized;
- Perform tests with consequences. Tests should be made by using the end result of the prioritization process as the maintenance cycle implementation plan and execute him;
- Change management process should be created and tested.
8. Bibliography

9. Appendix 1

9.1. **SCRUM**

SCRUM is one of the new agile approaches to building software. SCRUM is a simple approach to the management of complex problems, providing a framework to support innovation and allow development teams to deliver high quality product in short time-frames (35).

SCRUM offers some advantages (34):

a) Client-driven iterative development: implies that the choice of features for the next iterations of a software project are made by the client, whatever they perceive as the highest business value to them;

b) Time boxed Iterative Development: is the practice of fixing the iteration time end date and not allowing it to change. An overall project can be time boxed as well;

c) During the iteration, no changes from external stakeholders: Once the requests for an iteration have been chosen ant it is underway, no external stakeholders may change the work;

d) Adaptive development: It implies that elements adapt in response to feedback from prior work – feedback from users, tests, developers, and so on.

e) Evolutionary delivery: There is a vigorous attempt to capture feedback regarding the installed product, and use this to guide the next delivery.

Some key practices of SCRUM include (34):

a) Self-directed and self-organizing team

b) No external addition of work to an iteration, once chosen

c) Daily stand-up meeting with special questions

d) Usually 30-calendar day iterations

e) Demo to external stakeholders at end of each iteration

Scrum is known to deliver faster and better software for customers or end users, and can be viewed as a collection of good ideas and best practices (36). The term SCRUM was first used by Takeuchi and Nonaka in a study called “The New New Product Development Game”, which was published in the Harvard Business Review (40).

SCRUM is best viewed as collection of good ideas and best practices to manage product development. It builds cross-functional development teams that aggregate different areas of knowledge to deliver faster and better software products. (36).

SCRUM is a set of interrelated practices and rules that optimize the development environment, reduce organizational overhead, and closely
synchronize market requirements with iterative prototypes. Based in modern process control theory, SCRUM causes the best possible software to be constructed given the available resources, acceptable quality, and required release dates. Useful product functionality is delivered every thirty days as requirements, architecture, and design emerge, even when using unstable technologies (37).

9.1.1. SCRUM Lifecycle

The SCRUM lifecycle is composed of four phases (34):

a) Planning
b) Staging
c) Development
d) Release

Each phase has a set of purposes and activities. Table 5 illustrates the activities and purposes of each phase.

<table>
<thead>
<tr>
<th>Purposes</th>
<th>Activities</th>
<th>Pre-Game</th>
<th>Development</th>
<th>Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planning</td>
<td>Pre-Game Planning</td>
<td>Planning</td>
<td>Development</td>
<td>Release</td>
</tr>
<tr>
<td>Staging</td>
<td>Establish the vision, set expectations and secure funding</td>
<td>Identify more requirements and prioritize enough for first iteration</td>
<td>Implement a system ready for release in a series of 30-day iterations (Sprints)</td>
<td>Operational deployment</td>
</tr>
<tr>
<td>Purpose</td>
<td>Write vision, budget, initial Product Backlog and estimate items</td>
<td>plan exploratory designs and prototypes</td>
<td>Sprint planning meeting each iteration, defining the Sprint Backlog and estimates daily SCRUM meeting Sprint review</td>
<td>documentation training ...</td>
</tr>
<tr>
<td>Activities</td>
<td>exploratory design and prototypes</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5: SCRUM Lifecycle
9.1.2. The Process

The SCRUM process starts with the sources for the product backlog. These sources can be Product marketing, sales, engineering and customer support. The items that might be good ideas for the product are added to the Product Backlog, which is a prioritized list of all product requirements, such as features, defects, and desires. The prioritization is only done by the Product Owner.

In the Sprint Planning meeting, SCRUM Teams meet with the SCRUM Master to plan the next sprint. The product of this meeting will be the Sprint Backlog, which is a list of requirements to be implemented during the next iteration. During the sprint, daily meetings are performed to check the status of the project and assign daily tasks to team members.

At the end of the Sprint a deliverable is produced. This deliverable is an executable product increment that must be presented and any incomplete work returns to the product backlog. The team gets together with management at the Sprint Review meeting to inspect the product increment. Finally, the product backlog is rearranged and the process starts again with the Sprint Planning meeting (38). This process is defined in Figure 18.

![Figure 18: The SCRUM Process](image)

This process definition encloses different roles, ceremonies and workproducts. We will now discuss each of them in more detail.
9.1.3. Workproducts

Product Backlog

Product backlog is a set of all conceivable items that can be turned into requirements and placed into a Sprint Backlog. These items are prioritized by the Product Owner and estimates are done. These estimates are rough guidelines, refined once the team commits to an item (34).

Sprint Backlog

Sprint Backlog includes some items from product backlog that were selected to be implemented during the next iteration. Daily estimate of work remaining for each task and total hours remaining in each day are tracked. This control is done daily by the responsible members or by daily tracker who visits each member. New estimate are allowed to increase above the original estimate (34). The Sprint Backlog is constructed based on product backlog prioritizations. This prioritization is done through a method called dot voting. Dot voting is very similar to the 100-Point Estimation technique with the difference of using only developers and project managers as process participants.

9.1.4. Ceremonies

Sprint Planning Meeting

The Sprint Planning Meeting is actually two consecutive meetings. During the first part of the meeting, users, management and Product Owner, with the SCRUM Team review the Product Backlog, and determine the next Sprint goal and functionality. The second part of the meeting is focused work and responsibilities distribution through the SCRUM Team. Individual tasks are established in order to perform the work required to build the product increment. This part of the meeting is closed to participation by users and management (41) (42).

Sprint Review Meeting

In the Sprint Review meeting the team presents to management, customers, users, Product Owner, and Scrum Master a demo of the product increment that they built during the Sprint. The Scrum Master is responsible for coordinating and conducting this meeting.

Only completed, tested and functioning features should be presented at this meeting. There are allowed questions and user feedback is encouraged (36) (38) (41) (42).
Daily SCRUM Meeting

Daily SCRUM meetings are used for the SCRUM Team keep track of project evolution on a daily basis. In this 15 minutes meetings work is synchronized between all team members and any impediment is reported to the SCRUM Master. This is the opportunity to keep communication channels between team members open.

Each member should answer three main questions:
- What have you done since the last SCRUM?
- What will you between now and the next SCRUM?
- What got in your way of doing work?

These questions provide a set of benefits. It improves communication by itself, eliminates any other kind of meeting, indentifies and removes work impediments earlier than usual, highlights and promotes quick decision making and improves everyone’s level of project knowledge (42) (36).

9.1.5. Roles

Product Owner

The product owner is responsible for managing the product backlog and also ensuring that it is visible to everyone (41). He is responsible for converting all different inputs into a common product vision (42). He is also responsible for the prioritization of the product backlog, which is characterized by ensuring that the most valuable functionality is produced and built first (41) (38).

The Product Owner is the person responsible for the product and everybody in the organization must respect his decisions. He is only one person, not a committee, because without a single Product Owner floundering, spin, contention, and frustration will arise (41).

SCRUM Master

The SCRUM Master assumes the role of project coordinator. He is responsible for the daily SCRUM meetings organization and promotes the process by removing impediments, enabling the team to organize and manage itself. The SCRUM master is responsible for the success of the SCRUM and for ensuring that SCRUM values, practices and rules are enacted and enforced. (41) (42).
**SCRUM Teams**

Scrum Teams are committed to achieve the sprint goal, i.e. turning a Product Backlog into a product increment within iteration. Teams are self-managing, self-organizing, and cross-functional, and have full authority to do whatever they decide is necessary to achieve the goal (38) (41).

The recommend size of each SCRUM team is seven members, minus or plus two. This size is suggested based on the fact that big sized teams don’t work well, productivity decreases and the SCRUM’s control mechanisms don’t work become heavy. On the other side, small teams limit the amount of interactions and reduce productivity. At the end of each sprint the composition of the team can change (41) (34).

The SCRUM methodology offers a set of practices that can be easily applied to application maintenance. The use of iterations and sprint and product backlogs are good practices that can be used to develop and improve application maintenance processes. The use of dot voting as the preferential method to prioritize requirements can be short sighted because users and most of the stakeholders are left outside of the decision making process. Even with user feedback between iterations it is hard to all users, as a group entity, to understand and realize all the available request, they only know theirs, and therefore the decision of which user requests are put into the sprint backlog may not be the best one.