Software Project Management Anti-patterns in Students’ Projects

Raptopoulou Charikleia¹, Poranen Timo², Berki Eleni², Stamelos Ioannis¹

¹Department of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece
{raptopoc,stamelos}@csd.auth.gr

²School of Information Sciences, University of Tampere 33014, University of Tampere, Finland
{timo.t.poranen,eleni.berki}@uta.fi

Abstract

Management of software projects has proven to be one of the most challenging tasks in the field of software engineering. During the development of a project several problematic situations and anti-patterns may arise that hinder the successful application of management practices. In this paper we analyze anti-patterns met by students attending a Software Project Management course. During their projects students met over twenty different anti-patterns. Train the Trainer and Hero Culture were most common anti-patterns. We also report a new anti-pattern, Unbearable Lightness of Being.

1.0 Introduction

Software project management (SPM) [1] is the art and science of planning and leading software projects and considered to be one of the most complex and difficult tasks during the construction of a software product. The successful development of a software project is highly demanding and involves a number of different factors that must operate and cooperate properly. So it is greatly relying upon proper handling of the three basic variables, existing in every project: people, technology and processes [2].

In Software Project Management, commonly occurring, repeated bad practices are stated as anti-patterns [3]. These practices are used frequently by software companies because they disguise themselves as an effective and efficient way to resolve common problematic situations, hence rendering it difficult for their negative consequences to be identified. As a result, many project failures in software industry can be attributed to the appearance of anti-patterns [4].
From what was mentioned above, it becomes apparent that SPM anti-patterns constitute a very important factor directly related to the success rate of software projects. Therefore, the study of SPM anti-patterns (causes, symptoms and consequences) can definitely lead to better understanding of problematic situations and improve management practices, increasing the success rate of software projects.

In this paper, we present anti-patterns that students have met in their software projects during academic year 2010-2011 at the University of Tampere. Data was collected via a questionnaire that was answered by students who managed software development projects. In Section 2, we give the terms of patterns and anti-patterns. Data gathering method is explained in Section 3. In section 4 anti-patterns are explained more specifically and we give some examples on their usage, we also present the results and analyze the encountered anti-patterns. Finally, Section 5 contains the summary and Section 6 the conclusions. Proposals for future work are given as well.

2.0 Patterns and Anti-patterns

In software engineering, a design pattern is a general reusable solution to a commonly occurring problem which tackles the problem and generates positive outcome to the project.[3] On the contrary, an anti-pattern is a mechanism that describes a commonly occurring solution to a problem that generates overwhelming negative consequences. So far about 200 anti-patterns are identified [2,3,4,5,6]. In software project management, an anti-pattern involves human communication and resolve people issues, which in many cases might be destructive to software processes. [2]

To define an anti-pattern, usually name, central concept, generic and unique causes, unbalanced forces (the kind of wrong management behind the anti-pattern), symptoms, consequences, identification, refactored solution and references are given [4].

3.0 Data gathering

At the School of Information Sciences of the University of Tampere, Software Project Management course (SPM) [8] is recommended to be taken in fourth or fifth year of studies. Students taking SPM are required to have earlier project experience and the theoretical adequacy needed to successfully apply management techniques in practice. The main objective of this course is to give students practical managerial experience on the complete development process of a software product.

During the academic year 2010-2011, SPM students were managing 13 software development projects. Because one of the teams suffered inadequate reporting, our study considers only 12 projects. Every project team consisted of two project
managers from SPM course and 4-6 third-year students as developers. The managers had a freedom to decide how they divided management tasks in their project. Each project team had a real client coming from local companies, non-commercial associations or from university units.

In 4 out of 12 projects the daily working language was English because the team members had different mother tongues. The average amount of projects’ work was about 1200 hours. In the majority of these projects, scrum development model was applied and the application type was a www-based application with a database. Detailed Information on projects can be found from a course report [9].

To familiarize project managers to the anti-pattern concept they were required to select an anti-pattern from literature and then to write a description of that anti-pattern to the course’s wiki [10]. As an addition to the standard anti-pattern definition the managers were asked to describe their own remarks on the anti-patterns in the context of student projects. Students were also encouraged to familiarize themselves with the concept of the anti-patterns by studying a relatively large number of them.

During the course the managers were asked about the anti-patterns they encountered during the initiation, development and project finishing phase. In total we received answers on initiation phase from 12 projects (answered by 24 managers), on development phase from 12 projects (answered by 18 managers) and on the project finishing from 11 projects (answered by 15 managers). Not all projects met the anti-patterns at all, or reported anti-patterns were not real anti-patterns (they were just difficult situations that hindered their projects’ progress). It should also be mentioned that students were in position to recognize only the anti-patterns that they had read and not others.

4.0 Anti-patterns in Students’ Projects

In Table 1 Unbearable Lightness of Being anti-pattern is defined. This anti-pattern was identified by a student when they were asked to write anti-pattern definitions (see Sections 3 and 4) as a course task.

<table>
<thead>
<tr>
<th>Name</th>
<th>Unbearable Lightness Of Being</th>
</tr>
</thead>
<tbody>
<tr>
<td>Central concept</td>
<td>A project group is formed. It is assumed that every member of the project group has (more than) enough to do. In reality, some group member (or possibly several) has not much to do. Their tasks are either sort of which cannot be proceeded until some other part of the project is done first, or there just actually isn’t so much to do with the task(s) as it was assumed beforehand. In the organisation there are many projects, and the project members are from different organizational groups. The project manager assumes that when a member of his project group does not have enough to do, s/he is assigned to another project by their</td>
</tr>
</tbody>
</table>
The supervisor assumes that because this person is working in this certain project, s/he has (more than) enough to keep them busy. In the meanwhile, the member in question is confused and uncertain whether or not to do and what. And even if s/he makes the situation clear with the project manager and their supervisor, asking for more or something else to do, neither of them isn't capable to reassign them to another job or to find more to do within the current project.

<table>
<thead>
<tr>
<th>Generic and unique causes</th>
<th>Lack of communication between general management and project management. Lack of management's ability to appoint personnel for real tasks. Vague project planning, task delegation and goals.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unbalanced forces</td>
<td>Management's unawareness and negligence. Too busy to see how things really are.</td>
</tr>
<tr>
<td>Symptoms</td>
<td>Confusion among the project personnel. A staff member has not enough work, or has a job no one can figure out. Roles and tasks might be changing every few weeks. In meetings anything is not really decided or scheduled. Instead people go round same things everytime. “Sure, I'll do that for the next meeting.” And in the next meeting everyone is forgotten what was to be done - and project manager isn't even concerned about that.</td>
</tr>
<tr>
<td>Consequences</td>
<td>Effort is lost. Personal motivation is severely lost. Project loses and cannot facilitate valuable workforce.</td>
</tr>
<tr>
<td>Identification</td>
<td>In meetings nothing or little new is introduced by certain person(s). Things are standing still.</td>
</tr>
<tr>
<td>Refactored solution</td>
<td>More transparency within the organisation. Manager-level should be more aware what workers are doing and how things are actually going. Projects should be planned more precisely so that task distribution is based on facts, not on mere assumptions.</td>
</tr>
<tr>
<td>Student projects</td>
<td>Since student project may not be as serious as “real” job, it is possible that one intentionally doesn’t participate in the project with full effort and lets other project members to assume that s/he has enough to do anyway.</td>
</tr>
<tr>
<td>References</td>
<td>Warm Bodies [5], Proletariat Hero [3].</td>
</tr>
</tbody>
</table>

Next we introduce shortly some common anti-patterns.

**Emperor’s New Clothes.** Fear of being seen as less competent and capable developer prevents people from asking questions, understanding and criticizing the project. Negative feedback is perceived as intimidating and regressive force, rather than as a constructive force [5]. Emperor’s New Clothes is usually met in teams where members are not familiar with each other. Because project teams were formed for the course’s purposes, team members neither knew each other, nor had experience in working together. That resulted in Emperor’s New Clothes appearance in the initiation phase of one of the students’ projects. This anti-pattern had a negative effect on the project 9, as feedback was not so accurate and team members were not in position to fully understand the design or code.
implementation, resulting in unequal division of work, poor team spirit and, finally, delayed development of the software product.

A **corncob** is a difficult person who causes problems through destructive behaviour for the project team and project processes. Usually, he or she unfavourably impacts the team throughout technical, political and personal means [3]. Corncobs, for example, constantly disagree with key objectives or essential processes and continually try to halt them. Corncobs might be avaricious, pride and narrow-minded individuals or just have low level of motivation about the project. There was one project where corncobs were experienced. Managers reported that valuable time of the project group was spent on confrontations instead of product development. Also, the scope of requirements was frequently changing due to disagreements, resulting in additional time loss.

**A Team Is Appointed** for the project with no particular criteria and no screening techniques to judge their skills and qualification for the job [5]. In most of the cases, this anti-pattern is met when the manager lacks knowledge of what are the abilities, skills and area of expertise of his/her team members. Due to the fact that a relatively large number of teams were formed by the instructor of the course, who could not have a representative sample of the skills of each student, two teams reported to have experienced Appointed Team, one in the initiation and the other in the development phase. Managers assigned to their team members tasks with almost no clue that these members can get on with and complete them. This anti-pattern had as a result big delays in delivery and poor quality of the final product.

**Train the Trainer** Training an entire team in some technology can be more expensive, time-consuming and less effective than training just a few individuals who can afterwards train the rest of the team [5]. For instance, one of the most discernible negative effects of training a whole team is that people have taken the class are not capable and experienced enough to teach the object they have taught to other team members. Train the Trainer anti-pattern was met four times. Managers reported that assigning a topic per person to learn and teach others would have made the development process more efficient. One of these three teams met this anti-pattern in the initiation phase and in the development phase as well.

**Fire Drill** appears when managers wait until the last possible moment to allow developers to proceed with design and implementation; then they want results almost immediately [3]. A typical example of Fire Drill is when the management team of a project spends too much time on performing requirements analysis and planning, so that in the end very little time remains for the design, implementation and demonstration of the product. In most cases, time consumed in the analysis phase is two to three times the time spent on actual development of the product. Fire Drill was met in two projects. The first team encountered it during the initiation phase and the second team during the development phase. Managers reported that the appearance of this particular anti-pattern in their projects had as
an effect time and quality budget not to be met, since the start of development had been delayed too long.

**If It Is Working Do Not Change It** Editing legacy code is time consuming because project members do not understand the old code and feel reluctant to make big changes in fear of breaking the software [5]. To illustrate this anti-pattern, let us consider the case of a software component that is seemingly functional but works in an unknown way. Nevertheless, developers make the decision to keep it “as is”. Due to the fact that the functionality of the component is not understood, every change in the system might break some assumption about the code. If It Is Working Do Not Change It is usually met in projects where legacy software has to be maintained or added extended functionality to. The team that encountered this anti-pattern had to implement such a project. The managers of the team reported schedule delays due to this particular anti-pattern and in order to address it they assigned to the team members smaller and easily implementable tasks to help them to get into the legacy code.

**Hidden Requirements.** This anti-pattern generally arises when system requirements are not adequately documented or requirements documentation is too freely followed [5]. As a result, different assumptions about design requirements are made between the client and the development team. For example, certain aspect of project is not discussed during the requirements specification phase, like whether project should be implemented using formal software design or not. Customer would be subsequently surprised by project team not using MVC-model. Hidden Requirements was reported to be met by four teams in the initiation phase of their project. Managers explained that team members systematically neglected to update the project management tool with the incoming requirements right after they had received them, and that was a key reason that they failed to keep track of things to be done, and complete the development without some features or even the actual desired functionality.

**Feud** is characterized by personality conflicts among project managers that can dramatically affect the working environment. This kind of animosity is generally reflected in the actions of employees, resulting in their low motivation for work, or even worse, their negative attitude towards the project [3]. As an example, project managers start blaming each other, claiming that the other manager’s team is responsible for the problems in the project. In most of the cases, this situation brings a lot of frustration and hostility to the members of the managers’ teams. Power, managers have over their team, can often lead to arrogant behaviour from their side to the extent that they think their opinion and point of view of the issue discussed are of more importance than that of the rest of the team. The above behaviour usually characterizes novice managers with little working experience, because it is difficult for them to handle and control the power their position holds. For that reason, The Feud is very probable to appear in students’ projects. One team reported to have met this anti-pattern in the initiation phase. According to the managers, it had as a consequence low productivity and poor quality of work.
Reinvent the Wheel arises when a software solution is being built from scratch, without prior examination of the available approaches and solutions to this problem [3]. In this case, the developers of the team think that the problem they have to solve is unique and hasn’t been solved or faced before by any other developer. In reality, there is very high probability that the problem-to-solve is not unique, and an equally high probability that there exist many different approaches (and, possibly, solutions) to the specific problem. Reinvent the Wheel appeared in one project, during the initiation phase, when team members tried to implement basic TCP functionality from the ground up. This resulted in a significant amount of time to be spent in trying to re-solve already solved problems, instead of focusing time and energy on the really unique part of their project.

Sometimes software implementation does not reflect the design made by architects. This situation can happen when design is fundamentally flawed and the developers cannot explain why it is flawed, because system architects never code and are “uninterested in implementation details” [5]. Architects Do Not Code is commonly met in teams where architects are not involved at all in the implementation process. One team reported to have met Architects Do Not Code during the initiation phase of its project.

Hero Culture anti-pattern arises when the development process relies upon the heroic efforts of a small number of capable individuals striving to achieve a successful outcome [5]. Hero Culture can be aptly described with a typical situation where team members work every day for 16 hours in order to reach impossible deadlines or be in line with the high expectations of their managers. Hero culture was reported to have been met in three projects, while one manager described that during the last phases of the project a part of the team did an enormous job so as the project to be finished.

Sometimes, the software team does not give the expected results. For that reason, project manager considers team members incapable (“idiots”) and tries to develop an idiot-proof process based on some methodology, in order for the team to make some progress [5]. So, he/she writes a style guide, to show good practices and warn against bad ones. This often leads to continuous meetings and discussions about the restrictions and recommended practices and, inevitably, style guide grows too big. In the end, manager loses his/her confidence and the project’s status does not improve. Idiot Proof Process was met in the initiation phase of two projects. One possible reason might be that older students have to work with younger ones, and the first know much more about programming than the other.

They Understood Me anti-pattern occurs when a project manager or senior developer explains shortly what he/she wants to be done without any further specifications, assuming that project members and junior developers have understood it correctly [5]. In reality, rarely this is the case, so it is almost sure that project members or junior developers will implement either what they understood
or what they are able to do. They Understood Me was reported to be met in one project during the initiation phase. Managers described that the appearance of this anti-pattern mainly resulted in loss of valuable time because members of the team worked on something that was drooped afterwards.

**Gilding the Lily.** Sometimes, during the design and development phases, additional requirements and design attributes, that are not included in the initial requirements, become of the main requirements in the project. This anti-pattern arises when these excessive and exorbitant requirements are forced onto the system, hence rendering it difficult, if not impossible, to be completed in the specified time [2]. Additionally, some of these requirements are far too complex to be documented, developed and tested, related to the value they add in the final product. Even though this kind of requirements can be introduced by the client, the architect, or the developer, it is within the scope of project manager’s responsibility to discern whether they are of critical importance or not. Two teams encountered Gilding the Lily, one during the development phase and the other during the project finishing. In the first case managers reported that the anti-pattern resulted in product’s quality deterioration while in the second case the anti-pattern affected testing process which was not implemented adequately.

**DLL Hell** [7] is used to describe problems that happen due to the usage of external dependencies in programs, like Dynamic Link Libraries (DLL). Such problems can arise when shared DLLs between many applications are overwritten with DLLs of older versions during the installation or are deleted during the uninstalling of another program. One of the teams encountered Cut-and-Paste Programming and one DLL Hell during the development phase.

When team meetings are arranged in order to solve problems concerning project scheduling, **Yet Another Meeting Will Solve It** arises. People that should be involved with delivering the project start to be involved in these meetings [5]. As a result, valuable working hours are wasted while trying to schedule what has to be done, instead of actually doing it. One team encountered Yet Another Meeting Will Solve It anti-pattern in the finishing phase. Students were trying to discover why their project was late via meetings’ arrangements which brought even more delay to the project. The scheduling problems caused more meetings, in order to solve them, and the lack of experience caused even more meetings, in order for the solution to be reached. So, most of the time was wasted on meetings with very little overall efficiency.

**Mushroom Management** [3] is an explicit policy to isolate system developers from the system’s end users. Requirements are passed second-hand through intermediaries, including architects, managers, or requirements analysts. Mushroom Management is based upon the following assumptions: a) End-users have stated all their requirements from the project’s inception, b) Managers have well-understood the requirements stated by end-users and c) Developers have completely understood the overall requirements of the product given by managers. In reality, these assumptions rarely are true and it is almost impossible for all of
Mushroom Management was documented to be met in two projects. One team encountered it during the development phase and the other during the project finishing.

Project Mismanagement anti-pattern happens as a result of overlooking or minimizing key activities of a project. These include technical planning (architecture) and quality-control activities (inspection and test). In particular, basic mistakes include: inadequate architecture definition, insufficient code review (software inspection), and inadequate test coverage [3]. For instance, very often unit, integration, and system testing are minimized. One team met Project Mismanagement during the initiation phase, while another team met it during the development phase.

Ambiguous Viewpoint happens when object-oriented analysis and design (OOA&D) models are presented without clarifying the viewpoint represented by the model. Depending on the perspective of the OOA&D model, information is filtered accordingly, so only specific parts of the system are represented in the diagrams. Thus, not clarifying the viewpoint leads to great confusion about the implementation of the system, because it does not allow the fundamental separation of interfaces from implementation details, which are one of the primary benefits of the object-oriented paradigm [3]. Due to the lack of experience in project management, managers in students’ projects are prone to vague project planning and inefficient task delegation. Two teams encountered Unbearable Lightness of Being during their projects’ initiation phase and managers described that its appearance had a significant impact on team’s effort which could not be facilitated. Ambiguous viewpoint was met by one project team during the initiation phase. This anti-pattern can be very common among students’ projects because they lack experience in constructing OOA&D models.

Untested but Finished [5] occurs when the project is lagging in implementation, and the apparent cause is usually a programming bottleneck - too few or too slow programmers. Managers need to demonstrate progress or enlarge their span of influence; at worse, they want to be able to pretend to make progress. Untested but Finished was met by three teams during their project’s finishing.

E-mail Is Dangerous. E-mail is an important communication medium among project team members. Unfortunately, it is an inappropriate medium for many topics and types of messages like confrontations, criticisms, sensitive information and politically incorrect topics [3]. For example, lack of team rules regarding e-mail can lead to long-lasting public conversations which very often include name-calling, blaming, etc. E-mail Is Dangerous was reported to be met by one team during the initiation phase. Managers described that this particular anti-pattern had a detrimental effect on team spirit, resulting in lack of commitment and motivation, incorrect tasks, time delays and, finally, bad quality and reliability of the whole project.
Overall, students faced a wide range of SPM anti-patterns. Every team encountered approximately three anti-patterns. Limited amount of anti-patterns can be attributed to the small size of teams (2 managers and 5 members). Another reason might be the fact that teams’ managers were newly introduced to the concept of anti-patterns and, thus, had not adequate experience in identifying anti-patterns inside a project team. Finally, the nature of the projects might have had a significant impact on detected anti-patterns, because managers had a very limited scope and responsibilities.

Train the Trainer and Hero Culture were the most frequent anti-patterns. In many cases, students used technologies and programming languages that they were not familiar with, resulting in the frequent appearance of Train the Trainer. Hero Culture was also met multiple times, due to the fact that several students either did not dedicate an adequate amount of time to the project or quit the course.

In the Table 2 anti-patterns detected by managers are presented. The number of the project is depicted in the first column; while the next three columns depict the anti-patterns that were met by managers in the initiation, development and project finishing phase.

Table 2: SPM anti-patterns in students’ projects.

<table>
<thead>
<tr>
<th>Project</th>
<th>Initiation phase</th>
<th>Development phase</th>
<th>Project finishing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project 1:</td>
<td>Unbearable Lightness of Being</td>
<td>Cut and Paste programming</td>
<td>Fear of Success</td>
</tr>
<tr>
<td>Project 2:</td>
<td>Hidden Requirements, Reinvent the Wheel</td>
<td>Train the Trainer</td>
<td>Hero Culture, Gilding the Lily</td>
</tr>
<tr>
<td>Project 3:</td>
<td>Hidden Requirements</td>
<td>Corncob, firedrill., DLL Hell</td>
<td></td>
</tr>
<tr>
<td>Project 4:</td>
<td>E-mail Is Dangerous, Project Mismanagement, Architects don’t Code, Idiot Proof Process, Train the Trainer, Hero Culture</td>
<td>Train the Trainer</td>
<td></td>
</tr>
<tr>
<td>Project 5:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project 6:</td>
<td>Appointed Team</td>
<td>Untested but finished, Yet Another Meeting Will Solve it</td>
<td></td>
</tr>
<tr>
<td>Project 7:</td>
<td>Fire Drill</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project 8:</td>
<td>Unbearable Lightness of Being, The Feud, Hidden Requirements</td>
<td>Train the Trainer, Mushroom Management</td>
<td></td>
</tr>
<tr>
<td>Project 9:</td>
<td>Emperor’s New Clothes,</td>
<td>Gilding the Lily, Project</td>
<td>Death by Planning</td>
</tr>
</tbody>
</table>
They understood me

<table>
<thead>
<tr>
<th>Project 10:</th>
<th>Ambiguous Viewpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hero Culture</td>
</tr>
<tr>
<td>Project 11:</td>
<td>Hidden Requirements</td>
</tr>
<tr>
<td>Project 12:</td>
<td>Untested but Finished</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mismanagement</th>
</tr>
</thead>
<tbody>
<tr>
<td>If It Is Working Don’t change It, The Customers Are Idiots, Myopic Delivery, Untested but finished, Mushroom Management</td>
</tr>
</tbody>
</table>

**5.0 Summary**

Our previous analysis has shown that: The underlying causes of the anti-patterns met, as viewed from the students’ perspective, do not seem to have significant differences. In fact, several anti-patterns share the same generic causes due to the nature of students’ projects. Such causes are poor management, inefficient communication practices, difficulties in using tools and low motivation of team members, and are directly related to the overall lack of experience of students. Therefore, it is reasonable to assume that these causes and anti-patterns will appear in future students’ projects, as well.

Even though all the student teams managed to complete their projects, the appearance of one or more anti-patterns during software construction generated several problematic situations. Anti-patterns had unfavorable effects on projects and, by extension, on their deliveries. Almost all the projects suffered serious delays and were not able to follow their initial schedule. Several teams changed multiple times the scope of their system due to disagreements. In many projects was reported that members spent time working on something that was dropped in the end. Finally, a relatively high percentage of student managers reported the quality of the end product to be poor and the system not to have the expected functionality.

The number of anti-patterns reported in the initiation phase was higher than the number of anti-patterns reported during the development and finishing phase. The fact that student managers studied the anti-patterns’ descriptions at the start of their projects was the main reason for their gradually reducing capability to recall and identify these anti-patterns over time. Also, students might get more reluctant to answer similar research questions many time during a course.

**6.0 Conclusions**

Our analysis has also shown that anti-patterns are likely to appear in students’ software projects and cause many troubles that affect negatively the final software product. It is therefore important to train future project managers appropriately so as to be able to identify anti-patterns, to avoid them and to resolve them when they appear.
Student managers in order to familiarize themselves with the anti-patterns’ concept read a list of SPM anti-patterns provided on the web-site of the course [10]. The knowledge they acquired assisted them in identifying many of these problematic situations and trying to resolve them. Although the list included a large number of anti-patterns, the students rarely were able to identify more than two in every project. This greatly depends on the degree of anti-patterns’ comprehension by the students. The more students understand about how anti-patterns appear and affect software projects, the bigger number of anti-patterns are able to identify and resolve.

Improvements to the course’s wiki page, such as more concrete descriptions of anti-patterns, better examples of problematic situations and solutions adapted to students’ perspective, can lead to increased learning efficiency. In addition, the use of a specialized knowledge-based expert system for the anti-pattern detection process, such as SPARSE [14], is recommended. SPARSE assists managers by proposing directly related, but also semantically retrieved anti-patterns, according to a list of visible symptoms that may exist in a software project. Because detection of anti-patterns is one of the most challenging tasks in software project management, SPARSE has a very significant role in the efficacy and effectiveness of the process, giving managers the ability to increase the success rate of their projects. Thus, if used in future students’ projects it has the potential to a) assist them in familiarizing with anti-patterns, b) help them acquire greater experience in recognizing their symptoms and c) improve their management practices.

7.0 Acknowledgements

We thank Johanna Huhtala on describing the Unbearable Lightness of Being anti-pattern and Dimitrios Settas on helping us with the anti-pattern definitions.

8.0 References


