Master's Thesis

The Software Configuration Management Checkup and Improvement Framework

Jonathan Thiry INT09

Department of Computer Science
Faculty of Engineering LTH
Lund University, 2010

ISSN 1650-2884
LU-CS-EX: 2010-26

The Software Configuration Management Checkup and Improvement Framework

by Jonathan Thiry

Lund Institute of Technology, 2010

Master’s thesis supervised by Lars Bendix.

Table of content

Introduction

Yesterday, many companies were simply building hardware or providing services. Today, the evolution of their businesses lead them toward the inclusion of pieces of software in their products or services to continue their improvement. In the beginning, these companies often thought that their needs in software development would be limited. Now, they start to realize the importance of the shift. However, their restricted knowledge in software development prevents them to work in the best conditions. Good practices have to be spread within the organization and the project to reach this purpose.

Small software companies might encounter equivalent issues. Thinking their small sizes can allow them to neglect a good workflow implementation is probably a decisive mistake. Large companies introduced policies to reduce their problems. One of their challenges will be to adapt them to meet the needs of their different projects.

As early as 1975, F. Brooks introduced to the world many of these problems in his book the Mythical Man-Month [Brooks75]. Most companies still do not understand them. Why do they not put more effort in this process? Using Software Configuration Management (SCM), these formal hardware and service companies, or every software organization having problems in their workflows, could resolve their problems.

"Engineers have been developing complex systems for millennia. Many of the principles of CM were developed to enable hardware engineers to design and assemble components of ever more sophisticated configurations."

-- D. Whitgift

According to the Institute of Electrical and Electronic Engineers (IEEE), SCM is the discipline of organizing, controlling and managing the development and evolution of software systems. It should support all or most aspects of software development.

Why hardware companies struggle to implement SCM while the concept of Configuration Management comes from their field?

The voice of SCM gets stronger and stronger everyday. Most companies involved in software development are now aware of parts of basic Configuration Management, mainly related to elementary change management. By only providing a configuration management tool and little directives, companies do not guide their teams as they should.

How could these companies understand their SCM problems? They only see the visible part of their problems, the symptoms. These are resulting from deeper causes. The purpose of this work is to provide a framework explaining the link between the symptoms companies see and their root causes.

Several questions have to be answered to reach this target.
What are the software development symptoms todays companies encounter? What are all the problems that SCM can solve? How these lists can be created? How can symptoms be grouped together? How symptoms and problems can be mapped?

With this research, organizations developing software and with little knowledge in configuration management will have a framework to understand the SCM problems behind the symptoms they see in their development processes. Like in medicine where symptoms are related to diseases, here the solution consists in explaining the root causes (diseases) of the symptoms companies are able to identify within their projects. Once this root cause is known and recognized as an SCM problem, various solutions are often possible. However the possible cures will not be included in this study because of the quantity of material already available about this aspect.

This study will be built around a set of interviews conducted in various companies between France and Sweden. Different types of symptoms and problems will be discussed during these interviews due to the contrasts between companies and interviewees. To complete and support the data collected, SCM literature will be studied as well. This literature will mainly bring ideas for the list of SCM problems.

This paper is addressed to companies or people involved in software development with little or no knowledge in Configuration Management, though basic understanding about software development is preferable. If employed, Configuration Management jargon will be explained. People with responsibilities such as project managers will be able to apply the framework directly to their project and get quick results. People with more configuration management knowledge such as configuration managers could also get interesting ideas by using the product.

After analyzing the problems previously addressed, the research method used during the study and the requirements for the potential solution will be explained. This approach will enable the possibility to discuss, design and motivate the choice made for the solution. Before concluding the paper, a discussion about the results will present the tests and feedback used, and compare it to other work.

I. Analysis

After being introduced and described the problem needs to be carefully analyzed. The choice of the adequate research method will become easier in this condition and requirements for the desired solution will be created.

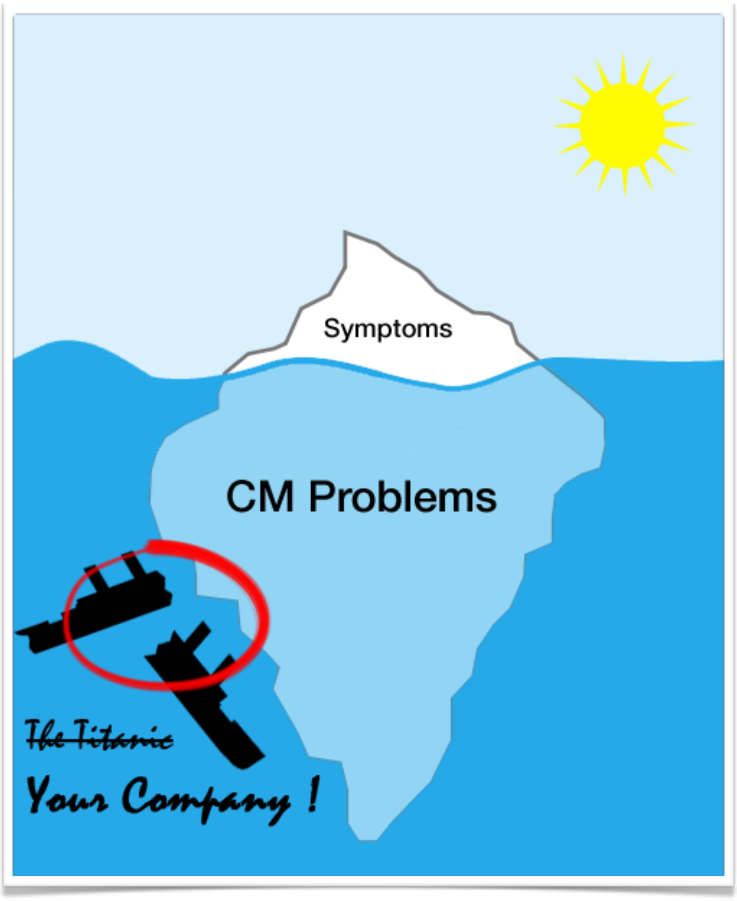
1. Problems

What are the problems undergone by companies involved in software development? Do companies with more experience have the same problems as the ones new to the discipline? Which parameters are influencing the type of problems encountered? As it has been introduced previously, these questions represent the inception of this study.

To be able to design the right solution, the problems need to be well understood. How can companies understand them? Why does the huge pile of available literature not help?

Companies often perceive surface problems only. Expound the reasons behind these problems is the purpose of this study. These outward problems will be called symptoms and their root causes, problems.

"It isn't that they can't see the solution. It's that they can't see the problem."
-- G. K. Chesterton



Like the Titanic, companies do not see under water... Trying to avoid the tip of the iceberg or the symptom is not the right solution.

The hidden part has to be avoided has well. By staying away from these underlying problems, symptoms will not appear.

The situation has been explained in the introduction using the doctor metaphor. An analogy with a car mechanic can also be used. A man brings his car to the garage and says: "I have a problem, my car does not start every time I turn on the contact". The mechanic knows it is a surface problem or symptom and will do a root cause analysis to find the real problem. When the root cause gets clearer, it is easy to find a solution, in this case, repair the starter or put a new one.

"The first step toward a cure is to know what the disease is"
-- Latin proverb

The main purpose of this work is to find a way to explain to these companies the link between their symptoms and SCM problems, or in other words, to do a root cause analysis on their software development symptoms.

While reflecting on the problematic, the need of defining precise objectives for the potential solutions appeared.
Every company involved in software development has problems related to Configuration Management. To have the largest impact possible, the product should be usable by every kind of organization creating software. To reach this target it should not go into specific details applicable in few situations only.
This study is focussed around SCM problems. Root causes should be related and solvable by SCM under all circumstances.
After designing parts of the product, interested companies should be asked for feedback. This will ensure the product is not the result of an imaginative mind but based on real cases and data.
Most SCM problems were documented many years ago. Using interviews proves that the problems presented are current facts.
Many companies come across these problems because of their lack of knowledge in Configuration Management.

2. Requirements and hypothesis

Considering the analysis performed on the problem, requirements have been defined to delimit the possible solutions. Since the discussion about them has been carried out in the last section, the list of requirements will not include any justifications. To fathom these issues, the product should be:

- **Generic**: The product's design should be based and applicable on every kind of software development. It should not reflect the kind of products or services provided by specific companies.
- **SCM focussed**: The product should only explore problems solvable by SCM.
- **Concrete**: Philosophical or abstract thinking should not be the outcome but only part of the process. Using actual proofs, the solution has to be a concrete product directly usable by companies.

- **Contemporary**: The most recent data available should be used to build an up-to-date product.
- **Empirical**: The product should not be based only on literature but also on interviews with people form the industry.
- **Structured**: The data collected will be the foundation of a framework applicable by any company involved in software development.
- **Simple**: This is the first work trying to regroup a large number of problems related to Configuration Management. For this reason the product will be kept simple. Complexity and completeness could be added in future works.
- **Detailed**: The product should present and describe the different problems and symptoms found.
- **Accessible**: The product has to be understandable by everybody involved in software development without any particular knowledge in Configuration Management. SCM terminology has to be explained.
- **Flexible**: While behind generic in the big picture, the product should be flexible and explain the differences companies could experience.
- **Approved**: The product has to be tested by its users. The method should be explained and the results analyzed. Interviews give the opportunity to picture the needs and questionnaires to check the validity of the design implemented.

As well as designing a solution based on these requirements, this study is the opportunity to verify the several hypothesis. In the next section dedicated to the research method, these hypothesis will be analyzed and the implemented strategy to verify them will be explained.
The symptoms and problems resulting of the interviews depends on:

- **Country**: Symptoms and problems are different depending on the culture of a country.
- **Size**: Symptoms and problems are different depending on the size of the company.
- **Domain**: The domain of activity do not change symptoms and problems.
- **Employee**: SCM experts are more likely to understand their symptoms and try a solution.

3. Research method

In order to create the best solution incorporating these requirements, the research method is divided into two parts, the data collection and the feedback. Such an organization gives the possibility to get a continuous flow of data during the entire study. The new data will make the database of symptoms, problems and mappings more accurate as the project moves forward. This section is focussed on the data collection, the feedback method and results will be exposed in chapter 3. Two information streams have been set up to gather as much useful information as possible.

• Interviews

A set of interviews has been organized with people from the software industry to determine their symptoms and understand the problems behind them. To acquire the knowledge in the best conditions, the interviews were:

- **Anonymous**: Companies and interviewees details will remain secret. To take advantages of the differences between companies and interviewees, they will be divided into different categories. These categories will be presented later in this section.

- **Face to face**: Holding face to face interviews will help to feel how people see their workflows and problems. It surely enhances the interaction to get productive thinking and meaningful information. In these conditions interviewees feel the liberty to talk about any kind of problem without being afraid of the consequences.

- **Semi-structured**: Before starting the interviews a list of question has been prepared and it will help as a skeleton to guide the interview when it could get out of focus or the interviewees have no more ideas about a specific topic.

In these conditions only a small amount of interviews can be organized. During this study eight have been directed.

Interviews have been divided in three phases. This organization gives the opportunity to analyze the data and to use the experience in the following interviews.

To get as many different results as possible, taking into account the relatively small amount of time allocated to the task, interviews have been luckily conducted in an eclectic panel of companies. As stated in the previous section, several hypothesis based on these differences will be verified during the process.

- **Countries**: Working conditions can be different from one country to another. This study could check if the symptoms and problems are the same from one country to another. Being a French student in Sweden, I used this advantage to meet people from both countries: France and Sweden.

- **Size**: Do companies of different size have the same symptoms and problems? Luckily, interviews were held in international corporations as well as small regional enterprises such as startups.

- **Domain of activity**: Does the main activity of the company change its problems related to SCM? Companies proposing totally different products have been selected for the study.

- **Employees' position and background**: Depending on their SCM knowledge, employees can have a totally different point of view on their problems.

• Literature

The books and papers available are often focussed on problems SCM can solve rather than symptoms. While looking for useful literature, it appeared that nothing existed to help find SCM problems based on development symptoms. When reading a document where only the problem is explained, the reader has to deconstruct the problem described, extract the symptoms and build the opposite mapping to check if his organization is subject of the problem.

This study will provide another way to work with this Configuration Management problems which should be easier for the software industry to use.

The literature is a solid starting point for any project. The SCM material available consists in a large database today. However it does not provide all the required information and had to be paired with some interviews.

Interviews, when not conducted with SCM experts, essentially brought symptoms, as they are the visible part of the problem and easier to see for software professionals. On the other hand, literature and Configuration Managers were more helpful with the list of SCM problems.

Coupling both flows of data created meaningful information to the two databases of problems and symptoms which lead to the creation of mappings in between them.

Problem analysis, requirements, research method; this first chapter grouped the different elements required to build a reliable solution.

II. Design

Keeping in mind the requirements defined in the previous chapter as well as the limited amount of time allocated to the study, the quest for the quintessential solution can commence. The first section of the chapter is defining and motivating the choices made for the solution. The following sections will discuss the details of this solution.

1. Solution

This study is neither destined to a specific company nor to handle a particular problem. Its intention is to involve several companies through interviews and to work on the largest number of problems conceivable (as long as they follow the previously stated stipulations). As formerly exposed, the objective of this research is to build a framework treating as many Configuration Management problems as possible. A framework offers a structure usable in every company by applying a customized implementation. In these conditions, the amount of time spent on each element would be as short as possible while bringing a complete solution. Trying to solve a large range of problems has one main drawback consisting in the impossibility to address every aspect in details. A compromise between the quantity of information and the time spent working on each piece of information has to be found and respected.

To offer the possibility to explain Configuration Management problems, the framework obviously has to include a list of them together with a way to help companies understand the ones they are facing. As explained before, the visible part of these problems are their symptoms.

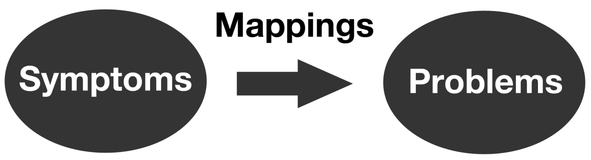
"The reason of a resolution is more to be considered than the resolution itself."
-- Sir John Holt

Logically the first step lies in a journey in the symptoms and problems world, trying to list as many of them as possible. In this chapter, the list of symptoms will be discussed in section 2 and the list of problems in section 3. The complete lists with the description of each element can be found in appendix A for the symptoms and B for the problems.

Once in possession of two lists with enough consistent data (this tipping point occurred when 30 symptoms and 15 problems were gathered, two arbitrary numbers. It was the time to move on to focus on the second part of the work), the next move is to create a link between them. Using this connection, the user can find his SCM problems by submitting his symptoms. This need lead to the creation of a third list relating symptoms with problems. Working between symptoms and problems in both ways is necessary to be able to consider every association possible. As a mat-

ter of fact, it would not be viable to construct the entire mapping working in one way only.
Reflecting on the mappings helped to consider new symptoms and problems. When cogitating on one list at the time, only a limited amount of elements can be found. When considering both lists as well as their relations at the same time, new symptoms are found by asking "what are the symptoms related to this problem?" and vice versa for the problems.
The second step consists in the reflexion on the mappings. This list will be discussed in section 4 of this chapter.

The solution emerges naturally by following these steps and leads to this panacea decoding the initial puzzle. Concessions are necessary in any case; not focussing on any problem and staying relatively superficial for each of them is the compromise made by this study. Due to the innateness of this choice, no other possibility has been seriously considered.



Rejected symptoms and problems have also been included in these sections. They help to understand the choices made for the final framework and where its limits stand. Like the symptoms and problems included, a discussion around them will be held and an explanation for their exclusion will be given. Furthermore once a number had been attributed to a symptom, problem or mapping, it never changed. It explains the gaps in the lists. The missing elements are presented afterwards in the rejected lists.

The three lists are accessible in appendix (A, B and C) for companies willing to exploit them. Their construction is debated in the following three sections. No specific order for their different element is followed. Moreover a better way to present and use this framework has been designed and will be proposed in section 5 of this chapter.

2. Symptoms

Through this section, discussions are held around the symptoms considered and the motivations for selecting them. As for the descriptions they are included in Appendix A, as part of the final framework destined to companies. The list of symptoms is based on both interviews and literature. Each symptom includes a reference to its source of inspiration.

To clarify the difference between symptoms and problems, symptoms are not closely related to Configuration Management but indirectly through their root causes. It creates a clear difference between symptoms and problems which could be difficult to distinguish otherwise. In this part, only software development terminology is used without any mention of the SCM jargon to be understandable by people involved in software development and without SCM knowledge. Nevertheless an exception has been made with the term "branch". The idea of branching is basic in Configuration Management and well integrated in software development nowadays. Using it allows to tackle more symptoms frequently experienced by many companies.

(S1) Is the project on time?

At the beginning this symptom was called: "The project is delayed". The difference is not in the change of the word "delayed" by "on time" but in the fact that the new formulation is a question. The interest is not to answer yes or no but to know if the question can be answered. The symptom is present when the question cannot be answered, it could be called symptom by ignorance. The first idea was slightly different but gathered too many root causes. Almost every problem can end up with delay in the project. This would not have been valuable for the framework. As described now it refers to a specific cause related to project tracking.

The original idea came from several interviewees who pointed it out as a problem. As expected interviewees called problems their surface problems which are called symptoms in this study.

After putting it in the symptoms list, it has been rejected at some point due to its uselessness. After its modification it has been put back in the accepted list. A different symptom number has not been used to be able to explain its origin.

(S2) The product does not meet the requirements.

This symptom has been inspired by interviews conducted with people working in the software industry. As presented in the description, different situations can lead to this symptom, CM can limit the risks. In a non-agile development process, the risk of developing a non satisfying product for the customer is high due to the endless time separating the writing of the requirements and the first shipping of the product.

Such a symptom can have different root causes. It might require to be paired with other symptoms to understand its real origin. The problem leading to this symptom can be outside the scope of SCM as well e.g. the customer changed his mind. Only reasons related to Configuration Management will be addressed here.

(S3) Productivity decreased after a process change.

This symptom has been inspired by interviews conducted with CM professionals. Configuration Managers have to be careful since productivity will decrease for every process change at the beginning. The situation should get better relatively fast if the appropriate training is offered. The Configuration Manager has to make sure the productivity increases rapidly afterwards. If it does not work, the right decision has to be taken which could be to go back to the previous process.

(S4) The program does not work anymore.

This symptom has been inspired by Walter Tichy's work [Tichy88] where he notes "This program worked yesterday. What happened?".

It can have many different origins and will not help to delimit the problem. However it constitutes a good starting point to become aware of the existence of a problem. It will have to be connected with other symptoms to understand the root cause.

(S5) An error cannot be reproduced in a configuration.

This symptom has been inspired by Walter Tichy's work [Tichy88]. He stated: "I can't reproduce the error in this configuration." The system should assist the people in charge of the support to quickly reproduce a bug when reported by providing all the information and files necessary. This is the essence of a businesslike maintenance strategy.

(S6) A change disappeared.

This symptom has been inspired by Walter Tichy's work [Tichy88] who introduced it as "Why did my change to this module disappear?". Who would seriously want to redo what he did yesterday? Nobody?

The symptom was called: "Something disappeared" to englobe more situations. After reflexion the name was changed to be more precise.

(S7) The documentation does not match the program.

This symptom has been inspired by interviews conducted with CM professionals and by Walter Tichy's work [Tichy88] who describe it as "The online documentation does not match the program". Many companies have at least a basic implementation of file versioning. This allows them to create and manage multiple revisions for each file in the system. However they get more difficulties to link these different versions together.

(S8) Do I have the right version?

This symptom has been inspired by Walter Tichy's work [Tichy88] where he asks: "Do we have the latest version?". After reflecting on his question, it appeared that the latest version is not always the one required. Changing the word "latest" by "right" had then been decided to overcome this limitation. It could also have been called: "What version do I have?". Like S1, the purpose is to know if the question can be answered (cf. S1 for more information).

This symptom has been included to show the importance for the programmer to know what he is doing and where he is going. Configuration Management should bring him this information with the concept of configuration identification.

(S9) Has this bug been fixed?

This symptom has been inspired by Walter Tichy [Tichy88] who asked: "Has this problem been fixed?"

Without properly reporting the status of a task, the team cannot know what has been done, what is remaining. Configuration Status Accounting fulfill this need.

(S10) Which modifications have been included in this build?

This symptom has been inspired by Walter Tichy [Tichy88] who wondered: "Which bug fixes went into this copy?"

Like S1, the purpose of this symptom is to know if the question can be answered.

(S11) Has this change been tried?

This symptom has been inspired by Walter Tichy [Tichy88] who asks: "This seems like an obvious change. Was it tried before?"

"Tried" or "tested"? As introduced by Walter Tichy, the word "tried" was used at the beginning. However, after reflection on the details of this symptom, its description ended up focused on tests and the idea brought at first was lost. It has been decided to keep both meanings and create a new symptom for the tests (S36).

Like S1, the purpose of this symptom is to know if the question can be answered.

(S12) Who is responsible for this modification?

This symptom has been inspired by Walter Tichy [Tichy88] who asked the same question.

Facilitating the circulation of information is critical for the success of a project. This kind of information should be available to anybody in the team without having to ask around.

Like S1, the purpose of this symptom is to know if the question can be answered.

(S13) Have these modifications been integrated?

This symptom has been inspired by Walter Tichy [Tichy88]. He wrote: "Were these independent changes merged?"

"Were these changes merged?" was the first title for this symptom. Presented this way, this symptom had been rejected. It did not appear to be relevant enough for the framework. Nevertheless the meaning linked to this idea was interesting. After reconsidering it, the symptom has been reintegrated in the framework in a slightly different way.

(S14) Merge problem - Changes have to be carried out manually.

This symptom has been inspired by interviews conducted with people working in the software industry. When managing variant of a same program, the CM tool should help to report the changes between them easily.

This symptom has been merged with S16 due to their similitude.

(S15) Modifications are not replicated to other branches.

This symptom has been inspired by interviews conducted with CM professionals. This can be the consequence of a system where the changes have to be reported manually (S14).

(S17) How to create a specific configuration?

This symptom has been inspired by interviews conducted with people working in the software industry. A program might require personification through the integration of different components or their parameterization. How can this specification be handled? Where can the different properties be stored? How can these slightly different programs me manage afterwards?

(S18) How these versions differ from each other?

This symptom has been inspired by interviews conducted with people working in the software industry.

Differences between two versions cannot always be known easily e.g. when only a few parameters differ for the needs of a customer. The tool used to compare file will not be useful in this situation.

(S19) Which version of the product has been delivered to the customer?

This symptom has been inspired by interviews conducted with people working in the software industry.

After delivering a specific version or configuration (cf. S17) of a program, these pieces of information have to be accessible to assure the maintenance. This is a symptom related to the identification of a specific configuration in a program deployed for a customer.

(S20) How has this configuration been created? What does it contain?

This symptom has been inspired by interviews conducted with people working in the software industry. Walter Tichy [Tichy88] asked as well: “How exactly this configuration has been produced?”

This symptom can be related to S17; Does a procedure exist to create a specific configuration? Is it then possible to retrieve the information? Why does any record exist?

(S21) Were the regression tests performed?

This symptom has been inspired by interviews conducted with people working in the software industry.

Regression tests are a good indicator to see if the project is progressing in the right direction. Is it possible to know if they have been performed or not?

(S24) Build composition.

This symptom has been inspired by interviews conducted with people working in the software industry.

Nowadays many programs needs to be customized for the customers or sometimes latests changes should not be included in the build. Does the SCM tool or process provide a way to easily choose the content of a build when necessary?

(S26) The unknown branch.
This symptom has been inspired by interviews conducted with CM professionals. Even if it will not create direct problems, it is important to know why branches are created and define a clear strategy. In this case no branch should have an unknown origin.

(S29) Everybody works on his own branch or copy of the program.
This symptom has been inspired by interviews conducted with CM professionals. Unless the division of the work has been designed this way or using a distributed tool where the idea is to have a private version, this should be avoided. Working this way will create more work for nothing.

(S30) The same version of a program has to be modified in different places.
This symptom has been inspired by Wayne Babich's work [Babich86] who is presenting the double maintenance problem. This symptom is the one related to this problem.

(S31) Developers are working simultaneously on the same copy of a program.
This symptom has been inspired by Wayne Babich's work [Babich86] who is presenting of the shared data problem. This symptom is the one related to this problem.

(S32) The program breaks after a successful merge.
This symptom has been inspired by the problem P4 (Logical conflict). This symptom is the visible part of this problem and always come together. The problem and the symptom represent the same idea: the symptom is the visible part and the problem is the underlying concept.

(S36) Were the tests run?
While discussing S11, this symptom has been added (cf. S11 for more information). Like S1, the purpose of this symptom is to know if the question can be answered. This symptom can be related to S21 (Were the regression tests performed?) but it includes further testing than just regression tests.

Rejected symptoms

(S16) Merge problem.
The development team is working on different version of the program, e.g. development or production. When modifications are done, they sometimes have to

be merge in the different version of the product. However the result of the automatic merge is not usable.
This symptom carry the same idea as symptom 14.

(S22) Merge conflict
When a developer commits modified files, they conflict with the latest version on the common repository.
When parallel work is authorized, conflicts are inevitable. They are not a symptom. If conflicts are not easily manageable, S14 should be considered.

(S23) Regression tests are failing.
Regression tests are breaking after introducing new changes in the program.
This symptom has been rejected because it does not indicate a problem in the configuration management system used. Having these regression tests as a protection to deeper problem is definitely a good option. As not having regression tests could create deeper problems, it could be categorized as a symptom. However it would not be totally related to configuration management. Even if it could be part of an automated verification process, this did not lead to a symptom after reflection.

(S25) Bug fix not fixing.
A bug has been reported, then fixed and released. However the customer still experiences the same problem. This situation is very frustrating for the customer. Did the customer get the new version? Has the right bug been fixed?
The support team understood the problem, fixed it and released it. If the customer installed the new version which should correct the right problem, it gets difficult to find the root cause behind the symptom but it will probably not be related to configuration management.

(S28) Continuous integration problems.
This symptom has been inspired by interviews conducted with CM professionals.
Implementing a continuous integration system can have many advantages. Problems can be encountered in the process, configuration management might help to overcome them.
This symptom is too general and does not provide any precise information. It could be used in a more complex framework.

The following symptoms are related to Configuration Management. As explained at the beginning of the section, these kinds of symptoms have been discarded for this framework. Not withstanding the benefit of it, implementing Configuration Management processes is not an easy task and can create new symptoms. After considering some of them, the decision of rejecting them came after remembering the target group of the product: people with no CM knowledge. These symptoms are rejected considering the limited needs of this framework but will probably be a good starting point for a product design for people with more knowledge in configuration management and with Configuration Management already largely implemented in the organization.

(S27) CM processes are shortcut.
Configuration management is implemented for the project. However for diverse reasons, developers or managers do not use them.

(S33) Configuration Management tool ignorance.
Called "Difficulties to find features in the CM tool" at the beginning, the symptom points to the lack of training using the tool. It could be considered as a problem, this has been rejected as well since the idea can be considered as a part of P5.

(S34) Configuration Management guideline.
This symptom was called "CM tasks are not adapted to the project", then "General company guidelines" to finish with the current title. The idea behind this name was always the same: Configuration Management is handled at a high level only (organization). It is a good first step but it needs to be adapted to each project to work in the best conditions.

(S35) Time consuming CM.
Developers loose too much time with CM tasks and/or processes. Moreover they find it difficult to see and understand the perks behind these tasks.
Are the processes necessary? Optimized? Are they helping developers or managers?
This symptom could lead to S27, people will stop doing the tasks.

3. Problems

This section introduces the discussions around Configuration Managements root causes called, as formerly stated, problems. These problems are the hidden origin of the symptoms introduced in the previous part. The pattern is the same as the one used for the symptoms that is to say, presenting the different possibilities and the motivations to selecting them or not. Their description is in appendix with the rest of the framework.

At variance with the symptoms, problems have to be directly related to Configuration Management. The symptoms listed in the previous section often have several root causes, some related to configuration management and others not. Problems from the second category will not be included in the following list as this work is focussed on SCM.

(P1) Double maintenance.
The problem has been introduced by Wayne Babich [Babich86]. This is the first of the three basic problems related to configuration management he presents in his

book. Despite the oldness of the book and the problems discussed, their relevance is unquestionable. Companies unaware of configuration management will most likely face them. They might seem less relevant to companies with more knowledge in software development since they implemented solutions a long time ago to solve them. These comments are applicable for the two following problems as well.

(P2) Shared data.

The problem has been introduced by Wayne Babich [Babich86]. This is the second of the three basic problems related to configuration management he presents in his book (cf. P1 for the discussion).

(P3) Simultaneous update.

The problem has been introduced by Wayne Babich [Babich86]. This is the third of the three basic problems related to configuration management he presents in his book (cf. P1 for the discussion).

(P4) Logical conflict.

If no rules are set for the developers when they commit their modified files, this problem will sometimes occur.

The best solution to cure this problem is probably to use strict long transactions. Developers can also be asked to checkout the entire program (or component at least) before committing new files and to verify no conflict are existing. Using this method corresponds to faking the strict long transaction when the functionality is not offered by the tool. Yet it is difficult to make sure this step is done every time new files are committed.

Although the theory solved the problem, not every CM tool offers the elegant solution.

(P5) Lack of CM training - Developer part.

Initially grouped under the same problem, the lack of CM training has been divided in two parts: developer and manager. Their name may be close but their significance is fairly distant as they address opposite point of views and needs. Developers and managers do not face the same symptoms, the root causes are different too.

This part of the problem has been introduced by Lars Bendix [Bendix00]. He suggests a training method including 3 metaphors to introduce the basic principles of CM more easily. Depending on the needs, more advanced workshops might be required to solve this problem.

(P6) Lack of merge history or powerful merge model.

Denominated at first "Lack of powerful merge tool", the true problem had been missed. Most of the merge tools available are powerful enough to manage merge conflicts in the best conditions. However the merge history kept by the CM tool is often not enough and companies are using too complex branching strategies.

This problem has been created after interviews with software development teams. Developers often try to avoid merges because the results of the automatic merges are not usable and will require much manual work to get the desired outcome.

(P12) Identification problem.

This problem has been inspired by discussions with CM professionals. It regrouped every problem related to Configuration Identification which is also its drawback; it can represent too many problems. Using more specific problems will provide more details but the problem would loose its generic attribute. It has been decided to keep the problem in this form and companies will have to analyze the problem more carefully to find a solution.

(P13) Lack of CM training - Manager part.

(cf. P5 for more information)

After reflecting on P5, it appeared that a separation between the two problems was preferable.

Managers will use CM in a different angle. They want to get feedback on the developers' work to be able to make the right decision.

(P19) Communication problem.

Configuration Management includes many elements helping the communication within the team. By making available many pieces of information, a set of specific questions can be answered without any interruptions required. As Wayne Babich [Babich86] explicates it, as the size of the team increase the communication channels are raising at an exponential rate.

The imprecise nature of this problem make it relatively easy to spot but more difficult to solve as a large number of cures are possible. Getting into details will require a great amount of work and references as W. Babich book already treat the subject.

(P20) Reproducibility problem - Lack of Bill-Of-Material.

This problem has been inspired by interviews conducted with CM professionals.

Providing a complete BOM will solve the reproducibility problem.

(P21) No status accounting.

Parts of configuration status accounting are missing. By making many pieces of information available, the project manager can make the right decisions.

(P22) Bad branching strategy.

When branches are used during a project, it should be for a specific purpose. A clear and simple strategy should be created.

(P23) No checkin rules.

This problem can be related to P4. Using checkin rules such as strict long transaction can lead to the improvement of the code committed.

(P25) Unavailable or undocumented feature in the SCM tool.

Project managers or developers with little knowledge in Configuration Management will probably not directly complain about a feature not included in the tool. However they might have questions or needs that the tool could theoretically answer. The tool might not include this feature or the company's policy might not document it.

Rejected problems

(P7) Bad quality.

According to tests, the quality of the program is not satisfying.

This problem is not directly related to Configuration Management. SCM will in many ways improve the overall quality of the project. Nevertheless the bad quality of the program cannot be cured to a single SCM magic trick. For this reason, the problem has been rejected.

“Bad quality” could be considered as a symptom since it is not directly related to Configuration Management. Find the root cause of such a symptom would be difficult; many different problems can lower the quality of the program and they are not always related to SCM. It would add complexity in the framework with little improvement. It has not been included for this reason but it should be considered in future work to create a more complex product.

Thinking about this problem lead to problem P23, no checking rules which was included in this problem at the beginning.

(P8) Team too large.

After adding developers to the project to finish it on time, the development speed decreased. Leaving aside the need of training and time of integration in the team for the new members, increasing the size of the team creates new communication canals. By complexing the communication network each team member works slower. This problem is broadly covered by F. Brooks [Brooks75].

Configuration Management cannot help in this situation. The manager needs to be aware of the right size for his team.

(P9) Bad balance between time, quality and number of features.

During a project, managers will often try to get too much from each side. Configuration Management will help to follow and control the three elements but in the end the manager only will decide where he wants to compromise.

This problem has been inspired by interviews with SCM professionals.

As well as P7, P9 could become a symptom but the little furtherance would not be worth the increased complexity for this framework.

(P10) Inefficient CM process.

The development lack in rapidity or the quality of the product is not satisfying. This might reveal an unproductive CM process that need to be improved.

This problem implies symptoms related to Configuration Management. Since it has been decided not to include these symptoms, the problem has been rejected too. This is however probably a good problem when trying to improve CM.

This problem was related to symptoms S27 and S35. Both have been rejected due to their closeness to Configuration Management.

(P11) CM for managers.

This problem occurs when Configuration Management is designed and optimized for managers only. It helps them in their work by providing useful metrics. The downside of this situation is the increased workload for developers.

Like P10, the problem insinuate symptoms too close to configuration management. It has been rejected for the same reasons, as well as its symptoms S27 and S35.

(P14) CM is not involved.

CM has not been considered when designing a process. It will not provide the control or feedback normally required.

Too many symptoms can be explained using this problem. It does not bring new information. Saying that SCM should be more involved in a process will not help either. To be really helpful, the framework needs to include more precise problems.

(P15) CM comes too late.

(cf. P14 for more details)

This problem is a variant of P14. It has been excluded for the same reason.

In contrary to P14, this problem could be included in a more advanced framework.

(P16) Too much CM - CM too rigid.

This problem has been inspired by interviews with CM professionals.

Description: The configuration manager implemented processes which are too rigid to be efficient. Managers and/or developers spend too much time working on the CM process. Their actual work does not go as fast as it should.

Like P10 and P11, the symptoms related to this problem are linked to Configuration Management. It has been rejected for the same reasons, as well as its symptoms S27 and S35.

(P17) Centralized CM.

CM is centralized in one department in the company. For the most part configuration managers create guidelines, processes or tasks applicable in every project. This will bring a good structure within the company and interoperability between projects. The downside will be the lack of specificity for every project. Each project is different and has its own needs. Considering guidelines made for the company level, configuration managers should also work on the project level to create the required details.

Again, without including symptoms related to configuration management, this problem has to be rejected too. It was related to S27, S34 and S35.

(P18) CM not optimized for managers.

Opposite to P11 ("CM for managers"), Configuration Management does not help at all managers in their work. Managers need information to be able to control their project in the best conditions, otherwise they end up navigating in the fog not making the best decisions possible.

Problems related to the optimization of CM need elements showing the symptoms currently present in the CM organization.

(P24) No test plan.

The project does not include a test plan. Tests might be sporadically used but no global strategy has been decided and/or is followed. The quality of the product is uncertain.

Configuration Management aims to improve the quality of the product by bringing rules. Beside their importance tests are not part of the discipline. It has therefor be decided not to include this problem.

4. Mappings

As stated at the beginning of this chapter, after overcoming the first step i.e. producing the lists of symptoms and problems, mappings between the two could be created. Considering the needs of the framework each symptom is mapped to its corresponding root causes (problems). During the construction of the list of mappings each problem has also been linked to its symptoms. This move seriously helped to consider all the possibilities between symptoms and problems. Symptoms pointing to the same problem might be related. This connection, when existing, is explained.

A small part of the mappings comes from well documented problems where the relation between symptoms and problem was easy to get. The most part, however, comes from the reasoning on the two previous lists. Tests and feedback presented in the next chapter are trying to demonstrate the legitimacy of these mappings.

• M1: S1 (Is the project on time?)
--> P21 (No status accounting)

What could indicate the status of a project? The notion of status accounting brought by configuration management can offer different indicators such as the status of the project. No other possibilities have been considered for this symptom.

Related symptoms: S9 (Has this bug been fixed?), S11 (Has this change been tried?), S12 (Who is responsible for this modification?), S21 (Were the regression tests performed?), S36 (Were the tests run?).

These symptoms are related to a lack of information due to the non-implementation of Configuration Status Accounting.

S19 (Which version of the product has been delivered to the customer?) has not been related to this symptom even if it is linked to a lack of status accounting. The *reason is that S19 does not directly connected to the development of the program but the follow-up of customers.*

• M2: S2 (The product does not meet the requirements)
--> P19 (Communication problem)

The communication between customers, analysts, developers and testers broke somewhere. Configuration Management can support the transit of information between them.

This symptom was also related to P10 (Inefficient CM process) before it had been deleted. The idea was relatively the same as the one linking it to P19 that is to say the problem to transmit information between different groups. P19 present it as a communication problem while P10 presented it as a CM process not able to give the required information.

Related symptoms: S8 (Do I have the right version?), S10 (Which modifications have been included in this build?), S17 (How to create a specific configuration?), S19 (Which version of the product has been delivered to the customer?), S24 (Build composition).

These symptoms bring more information about the cause of S2. Linked with one of them, its cause could not be a communication problem.

S8: Is the right version tested? Has the right version been sent to the customer?

S10: Were the modifications related to right set of requirements included in the build?

S17: Is it possible to create a configuration specific to the customer's requirements?

S19: Did we send the right version to the customer?

S24: Is it possible to build a specific application for the customer?

• M3: S3 (Productivity decreased after a process change)
--> P5 (Lack of CM training - Developer part)
--> P13 (Lack of CM training - Manager part)

Depending on the kind of process (involving developers or managers), the symptom does not lead to the same problem. As stated in the discussion about the symptom, companies have to be careful because every process change will lead to a drop in the productivity at first.

Related symptoms: S6 (A change disappeared), S8 (Do I have the right version?), S14 Merge problem - Changes have to be carried out manually), S24 (Build composition), S26 (The unknown branch).

These symptoms occur when people did not get enough CM training. They might indicate needs indifferent area of Configuration Management but the idea is the same.

• M4: S4 (The program does not work anymore)
--> P2 (Shared data)
--> P4 (Logical conflict)
--> P23 (No checkin rules)

This symptom can have many origins inside or outside the Configuration Management scope. Combining it with other symptoms will lead to the real cause.
This symptom was also related to P24 (No test plans) which has been rejected (cf. section 3 about the problems).
Related symptoms: S21 (Were the regression tests performed?), S31 (Developers are working simultaneously on the same copy of a program), S32 (The program breaks after a successful merge), S36 (Were the tests run?).
S4 can be related to one of these symptoms to get a better understanding of the root problem since they often break the program.
S21: If the regression tests are not performed, even a successful merge can break other parts of the program.
S31: While not modifying the program, it might break anyway. Surprising? No, developers are not alone in their workspace.
S32: If merges are not carefully handled or tests run after their success, the program might break in an unmodified component.
S36: Are tests regularly run to check if bugs have not been included?

• M5: S5 (An error cannot be reproduced in a configuration)
--> P19 (Communication problem)
--> P20 (Reproducibility problem - Lack of BOM)
Has all the necessary information been transmitted? The person in charge of the support might not understand the bug encountered by the user. To help the communication, the reporting process should be the best possible way to transmit the important data to the support.
Is a complete Bill-Of-Material provided? This symptom can also come from the fact that not all the information required to reproduce the same configuration are stored.
Related symptoms: S17 (How to create a specific configuration?), S19 (Which version of the product has been delivered to the customer?), S20 (How has this configuration been created? What does it contain?).
These symptoms bring questions to spot the root cause.
S17: Do we at least know how to create a specific configuration?
S19: Do we have the necessary information about the version used by the customer?
S20: Where does this configuration come from?

• M6: S6 (A change disappeared)
--> P3 (Simultaneous update)
--> P5 (Lack of CM training - Developer part)
--> P12 (Identification problem)
This is the main symptom of the simultaneous update problem. However it can also come from a mistake made by a developer due to his lack of knowledge in using the CM tool. Another possibility is that the specific file cannot be found because the files' organization is not clear.

Related symptoms: S8 (Do I have the right version?), S15 (Modifications are not replicated to other branches), S20 (How has this configuration been created? What does it contain?), S26 (The unknown branch).

S6 is not a precise symptom and has many origins possible. It will have to be related to one of these other symptoms to understand its root cause.

• M7: S7 (The documentation does not match the program)

--> P12 (Identification problem)

This symptom is due to a more advanced identification problem.

Is the version of both the source file and the documentation file known? If the CM tool does not offer a feature to relate the two files, an identification system has to be implemented to palliate the problem.

Related symptoms: S6 (A change disappeared), S8 (Do I have the right version?), S10 (Which modifications have been included in this build?), S18 (How these versions differ from each other?), S19 (Which version of the product has been delivered to the customer?), S20 (How has this configuration been created? What does it contain?).

These symptoms might be related to an identification problem as well.

S26 (The unknown branch) can be related to an identification problem but is not in the list because it is linked to a problem to identify a branch which should not have any impact in this case.

• M8: S8 (Do I have the right version?)

--> P5 (Lack of CM training - Developer part)

--> P12 (Identification problem)

This question cannot be answered if Configuration Management does not provide the piece of information required. Can I know on which version I am working? If the Configuration Management system gives the possibility to know the answer, the developer probably does not know how to find or understand it. Specific training on the Configuration Identification will then solve the problem.

Related symptoms: S6 (A change disappeared), S7 (The documentation does not match the program), S10 (Which modifications have been included in this build?), S18 (How these versions differ from each other?), S19 (Which version of the product has been delivered to the customer?), S20 (How has this configuration been created? What does it contain?).

These symptoms are related to P5 and/or P12 as well.

Training can cover various area, not always related to this symptom. For this reason, S3 (Productivity decreased after a process change), S14 (Merge problem - Changes have to be carried out manually), S15 (Modifications are not replicated to other branches), S17 (How to create a specific configuration?), S24 (Build composition) and S26 (The unknown branch) were not included.

• M9: S9 (Has this bug been fixed?)

--> P21 (No status accounting)

Status accounting provide the necessary information to answer this question. The bug report tool should bring this feature. This symptom is not related to P5 (Lack of CM training - Developer part) because it is assumed that if a bug reporting tool is available, the developers know how to access it and get basic information about bugs concerning its project.
Related symptoms: S1 (Is the project on time?), S11 (Has this change been tried?), S12 (Who is responsible for this modification?), S13 (Have these modifications been integrated?), S21 (Were the regression tests performed?), S36 (Were the tests run?).
These symptoms are related to a lack of Configuration Status Accounting. For the same reason as in S1 (Is the project on time?), S19 (Which version of the product has been delivered to the customer?) has not been included.

- **M10: S10 (Which modifications have been included in this build?)**

--> P12 (Identification problem)
--> P20 (Reproducibility problem - Lack of BOM)
To answer this question, a proper identification of the build is necessary. What information is available about the build? Is a complete Bill-Of-Material accessible? The lack of Bill-Of-Material is the more probable root cause since it would provide all the required information to know how the program has been built.
Related symptoms: S17 (How to create a specific configuration?), S20 (How has this configuration been created? What does it contain?), S24 (Build composition).
These symptoms are especially related to a lack of BOM since it is the most probable cause of this symptom.

- **M11: S11 (Has this change been tried?)**

--> P21 (No status accounting)
Is it possible to check a file’s history to know its previous modifications? The only possibility to know if a change has been tried or not is to have an accurate status accounting. One of the purposes of status accounting is to provide that exact piece of information.
Related symptoms: S9 (Has this bug been fixed?), S12 (Who is responsible for this modification?), S21 (Were the regression tests performed?), S36 (Were the tests run?).
These symptoms are related to a lack of status accounting.
However due to the specific piece of information required by this symptom, S1 (Is the project on time?), S13 (Have these modifications been integrated?) and S19 (Which version of the product has been delivered to the customer?) have not been integrated.

- **M12: S12 (Who is responsible for this modification?)**

--> P19 (Communication problem)
--> P21 (No status accounting)
Status accounting should spread this information. However it might conceal a deeper problem related to the communication inside the team.

Related symptoms: S1 (Is the project on time?), S2 (The product does not meet the requirements), S9 (Has this bug been fixed?), S11 (Has this change been tried?), S13 (Have these modifications been integrated?), S21 (Were the regression tests performed?), S36 (Were the tests run?).
Like S12, these symptoms are related to P19 or P21.

• M13: S13 (Have these modifications been integrated?)
--> P19 (Communication problem)
--> P21 (No status accounting)
The motivation for creating this mapping are the same as M12. The transmitted information is just different but the problems are identical.
Related symptoms: S6 (A change disappeared), S9 (Has this bug been fixed?), S12 (Who is responsible for this modification?).
These symptoms are related to P19 or P21 and to the management of modifications.

• M14: S14 (Merge problem - Changes have to be carried out manually)
--> P5 (Lack of CM training - Developer part)
--> P6 (Lack of merge history / powerful merge tool)
--> P22 (Bad branching strategy)
This symptom occurs in many different companies. When the automatic merge is not usable, companies often think their merge tool is the problem. It might be the situation. It might not.
To be more accurate the merge history is probably not providing enough information to the merge tool leading which cannot produce good results in this condition. Other factors might be at the origin of the symptom such as a bad branching strategy or lack of training for the developers.
Related symptoms: S6 (A change disappeared), S15 (Modifications are not replicated to other branches).
S6 and S15 can be a consequence of S14.

• M15: S15 (Modifications are not replicated to other branches)
--> P5 (Lack of CM training - Developer part)
--> P22 (Bad branching strategy)
Developers do not know or understand the branching strategy to be able to carry out their modifications where needed. This is due to their lack of training or to a too complex branching strategy.
Related symptoms: S6 (A change disappeared), S14 (Merge problem - Changes have to be carried out manually), S29 (Everybody works on his own branch or copy of the program).
These symptoms are related to P5 or P12 and where selected for their help in defining the root cause of S15.

• M17: S17 (How to create a specific configuration?)
--> P5 (Lack of CM training - Developer part)

--> P20 (Reproducibility problem - Lack of BOM)

A well defined Bill-Of-Material is the key requirement to be able to build a specific configuration. Without a proper training developers might not follow it exactly.

Related symptoms: S5 (An error cannot be reproduced in a configuration), S10 (Which modifications have been included in this build?), S20 (How has this configuration been created? What does it contain?).

These symptoms are related to the lack of BOM which is the main concern in this situation.

- **M18: S18 (How these versions differ from each other?)**

--> P12 (Identification problem)

Are the different versions clearly identifiable? Then it should not be too difficult to compare them.

Related symptoms: S6 (A change disappeared), S7 (The documentation does not match the program), S8 (Do I have the right version?), S10 (Which modifications have been included in this build?), S20 (How has this configuration been created? What does it contain?), S26 (The unknown branch).

These symptoms are related to a configuration identification problem.

S19 (Which version of the product has been delivered to the customer?) has not been included due to the different nature of the identification in this situation.

- **M19: S19 (Which version of the product has been delivered to the customer?)**

--> P12 (Identification problem)

--> P21 (No status accounting)

Is it difficult to identify a version deployed for a customer? Can the customer do it by himself if the product is installed on his site? Can the support act from a distant place? Why does the company not keep track of the version deployed?

Related symptoms: S7 (The documentation does not match the program), S8 (Do I have the right version?), S9 (Has this bug been fixed?), S10 (Which modifications have been included in this build?), S13 (Have these modifications been integrated?).

These symptoms are related to P12 or P21 as well. They were selected for the additional pieces of information they can bring compare to other symptoms linked to these problems.

- **M20: S20 (How has this configuration been created? What does it contain?)**

--> P12 (Identification problem)

--> P20 (Reproducibility problem - Lack of BOM)

To be able to get this information about a configuration, a clear identification should be possible first. Then to know more about its content and the way it has been created, a complete BOM has to be available.

Related symptoms: S5 (An error cannot be reproduced in a configuration), S10 (Which modifications have been included in this build?), S17 (How to create a specific configuration?).

These symptoms are related to the lack of BOM which is the main concern in this case.

• M21: S21 (Were the regression tests performed?)

--> P21 (No status accounting)

Status accounting will allow the team to follow the progress of the different tests and more precisely in this case the regression tests.

Related symptoms: S12 (Who is responsible for this modification?), S13 (Have these modifications been integrated?), S36 (Were the tests run?).

These symptoms are related to a lack of status accounting.

However due to the specific piece of information required by this symptom, S1 (Is the project on time?), S11 (Has this change been tried?) and S19 (Which version of the product has been delivered to the customer?) have not been integrated.

• M24: S24 (Build composition)

--> P5 (Lack of CM training - Developer part)

--> P25 (Unavailable or undocumented feature in the SCM tool.)

An increasing number of programs require the possibility to create specific configuration depending on the customer. However this feature might not be available in the SCM tool or the developer might not know how to use it.

Related symptoms: S10 (Which modifications have been included in this build?), S17 (How to create a specific configuration?), S20 (How has this configuration been created? What does it contain?).

This symptom are related to a lack of information concerning the creation or the identification of a build.

• M26: S26 (The unknown branch)

--> P5 (Lack of CM training - Developer part)

--> P12 (Identification problem)

--> P22 (Bad branching strategy)

Branches should be created with a specific purpose. The developer who created this branch probably did not understand the branching strategy.

If the branch has been created with a specific purpose, the required information to identify it should be accessible. It indicates that the branching strategy is probably bad as well.

Related symptoms: S8 (Do I have the right version?), S18 (How these versions differ from each other?).

This symptom is specific. By collecting information about the branch it should be understandable by itself. For this reason, it does not have many related symptoms.

S8 and S18 can help identify the branch more precisely.

• M29: S29 (Everybody works on his own branch or copy of the program)

--> P1 (Double maintenance)

--> P22 (Bad branching strategy)

It might stop the conflict between developers for some time but when the different parts will have to be merged, it will be too complicated for it to be done.

Related symptoms: S15 (Modifications are not replicated to other branches), S26 (The unknown branch), S30 (The same version of a program has to be modified in different places).

These symptoms are related to P1 or P22.

• **M30: S30 (The same version of a program has to be modified in different places)**

--> P1 (Double maintenance)

As W. Babich pointed out, when changes have to be reported in different copies of a program it is the signal for the double maintenance problem.

Related symptoms: S29 (Everybody works on his own branch or copy of the program).

S29 could explain why the same version of the program has been copied.

• **M31: S31 (Developers are working simultaneously on the same copy of a program)**

--> P2 (Shared data)

The symptom of the shared data problem introduced by W. Babich.

Related symptoms: S4 (The program does not work anymore).

S4 is the only other symptom related to P2. When companies experience this symptom, their developers will most likely face an unexpected broken program from time to time.

• **M32: S32 (The program breaks after a successful merge)**

--> P4 (Logical conflict)

If the merge was successful, the program should run properly. This symptom shows that an apparently unrelated component created a conflict. This is a logical conflict.

Related symptoms: S4 (The program does not work anymore), S21 (Were the regression tests performed?), S36 (Were the tests run?).

S4: the only other symptom related to P4 and the consequence of this symptom.

S21: Running regression tests will reveal S32.

S36: Like S21 (Were the regression tests performed?), running tests will help to discover this symptom.

• **M36: S36 (Were the tests run?)**

--> P21 (No status accounting)

What wan edentate the status of the test? Configuration Status Accounting brings this piece of information.

Related symptoms: S9 (Has this bug been fixed?), S11 (Has this change been tried?), S12 (Who is responsible for this modification?), S21 (Were the regression tests performed?).

These problems are related to the lack of status accounting.

S1 (Is the project on time?), S13 (Have these modifications been integrated?), S19 (Which version of the product has been delivered to the customer?) are not related to this symptom because of the different kind of information required.

5. Product

The three previous parts constitute the discussions about the raw material of the framework. The actual product containing the description of each part and each element can be found in appendix. By exploiting these results, a company can check its symptoms, then find and understand its Configuration Management problems by using the mappings. The outcome already includes most of the requirements formerly defined. The only one missing is the approval from the industry which will be treated in the next chapter. Is the framework presented this way (divided into three parts) the best package possible? The division brought simplicity as well as completeness in the product structure. The downside of providing it in this form consists of its usability. The user has to switch between the lists all the time and the results are not easily readable. Even if the content might interest people working in the industry this aspect might stop them from using the product.

During the interviews, people were asked how they would imagine the product. Many of them asked for a program that could present the related CM problems by providing the symptoms. While being attractive, this idea would have required much work and time which could have been used to improve the core data of the framework. Considered as a side objective, the workload has been focussed on the data leaving aside the presentation of the product at first.

The turning point has been the need of feedback. As explained in detail in the next chapter, a website has been created to present the unfinished framework and to ask different software professionals to give their feedback. At the same time, the opportunity of using the website to diffuse the final framework to the world appeared clearly. The website would have a double use, feedback at first and support for the framework after. The time required to build the website would not seem as lost anymore due to the needs for feedback.

The website presents the product in a more interactive way. The user just has to select his symptom in a list, the mapping is automatically done, and the results can be checked immediately. The problems regrouping the larger number of symptoms are displayed first due to their possible greatest importance. Every problem is explained in details to help the user to understand them and implement a solution responding to his needs. A screenshot of the results displayed by the website is available in appendix E.

During the research, the website was available at www.cmcheckup.com. The product and data was part of a personal project. However after the end of the project, it has been decided to use www.cmcheckup.org. There are only 3 letters different but the philosophy behind is not the same. Now the project has to be shared with the world to encourage people to improve it with their knowledge. This way the product will have the possibility to continue its evolution even after the end of this research.

Multiple hypothesis have been proposed in the previous chapter. Conducting interviews in different kinds of companies offered the possibility to get an opinion about these hypothesis. Considering the small number of interviews organized, the conclusions cannot be totally ensured.

Symptoms and problems do not depend on the country where the company is based. A similar company would have similar SCM problems everywhere in the world (or at least between France and Sweden).

The size of the company matters. Large companies tend to have spread guidelines through their projects. They avoid major problems by acting this way. However they might lack in customization inside their projects. Smaller companies are more flexible but sometimes implement exotic solutions, probably due to their lack of theoretical SCM knowledge.

As long as a company is involved in software development, its main activity domain does not influence the kind of symptoms and problems faced.

Depending on the interviewee, the symptoms and problems discussed are not the same. It might also be influenced by the problems faced by the company or the solution chosen. Nevertheless a configuration manager, a project manager or a developer do not talk about the same symptoms and problems. Most of the time they can see what they are facing but not what their colleague might endure. It shows again the importance of the communication especially between people exercising different functions in the project.

III. Discussion

The third and last chapter of this work will explain and discuss the method used to validate the results brought by the work done in the previous part. Reflexion about this study will be held including a comparison with other works and the possibilities for future research.

1. Feedback

Feedback has been organized to corroborate the product designed and developed in the last chapter. Receiving comments about the work would help to improve it as well as assure that the solution satisfies the industry.

Just after the first prototype was created, a questionnaire has been sent to a few people working with software projects, mainly including the people previously interviewed. The purpose of this questionnaire was to verify if the needs of the companies had been well understood. It included the list of symptoms and problems available at the time. People were only asked to add or remove symptoms and problems. Moreover descriptions for three symptoms and three problems were asked to be commented.
Unfortunately this questionnaire has not been successful. Only a small percentage of the people sent back their answer. After observing the enthusiasm during the interviews, this non-reaction was surprising. People who received the questionnaire and did not reply were contacted again to clarify their behavior. The main reason was the difficulty to answer a questionnaire presented as a classic document with empty fields to fill out with their answer. The users experience had not at all been considered when creating the survey. This aspect, presumed not important, led to the failure of the questionnaire.

Feedback was remaining fundamental. With the experience gained from the failure of the questionnaire, a new way to get constructive comments and critics had to be found.
How can the user experience provided by the questionnaire be improved? What should changed? To overcome these limitations a obvious change was to switch from a static document to a dynamic application. The effort made by building such an application would be quickly rewarded by an increased interaction with the users and a higher rate of response. The decision to make it a web application came from the need to diffuse the product more easily.
As expected the results obtained with this new approach were more satisfying. The interaction brought by the web application made the difference.

The comments resulting from the feedback were relatively satisfying. Most of the people seemed pleased with the product. Since it was mostly the people interviewed during the study, they already had an idea about the work and the possible outcome. The overall results met the expectations they got after discussing about the product during the interviews. They often had some comments about details in the different parts: incomplete descriptions or explanations resulting in the non understanding of some results. After complementary information was given about these matters, only few objections were left.

The feedback could have been more relevant if more software professionals, or at least people not previously interviewed, had been contacted. Too little time was remaining when the first version of the website got published online to ask for feedback. It would have taken too much time to find the people and especially to analyze the data resulting from the surveys.
Asking for feedback the people who brought the data at the beginning is another problem. It might look like the solution had been designed for their needs and it was meant to please them. Keeping in mind that possibility, an important point is also that data is coming from several people at the beginning. These people had a different point of view on the project and its potential solutions. The same will happen for the feedback.
For the last interviews the design of the product was already decided and the interviewee could already comment on it.

It is however the next step to communicate more broadly about the framework to get more reactions and improve it.

2. Comparison with related works

As stated in the first chapter, no similar study has ever been done before (or has, at least, not be found). A similar study designates a research meant to help companies find their SCM problems based on their software development symptoms and regrouping a large number of problems. However a comparison with the currently available literature (mainly about SCM problems) is possible. It would help to understand the advantages and drawbacks of the decision made during the study.

This literature is the one used as reference or background for the research. A part of it refers to symptoms, but the largest part is related to CM problems.
In the introduction of his paper, Walter Tichy exposes symptoms companies have. On the other side, Wayne Babich presents and explains problems. By analyzing his work, symptoms can be extracted from the problems description. Most of the literature studies treated SCM problems and the same method as the one used for Babich book could be used.

The main interest and advantage of this study is its broad aspect. No other work found gather and explain a large number of SCM problems.
The other advantage is the ease of use. By listing numerous symptoms, companies just have to pick them and find their underlying cause. Again no other seems to use this approach.

The main drawback of the product the superficiality of the symptoms and problems exposed. Literature focussed on few problems give more details about them. Due to the limitations and the purpose of this study, this fact is acceptable.

The source of inspiration of every problem has indicated. People willing to fulfill their potential need of details just have to read the corresponding reference. However problems inspired by interviews have not always been related to the literature. The description for these problems have been written to include all the necessary details and exclude the need of a reference.

3. Self-evaluation and future work

The first challenge was to define the needs of different types of companies involved in software development to design the framework. Doing such a study has one advantage: almost all companies have problems related to configuration management, the more difficult part was sometimes to find them.

As stated in the first chapter, one requirement for the framework was simplicity. Symptoms, problems and mapping had to stay simple to offer a finished product even if it is not totally complete. Some of them, such as “bad quality” could have been included but the complexity added did not seem interesting. The discussion could help future researchers decide to include them or not.

Moreover the choice of focussing on improvement or measurement was possible at the beginning. In the end the framework is concentrated on improvement rather than measurement to provide assistance to a larger number of companies and satisfy more needs. Including metrics in the work is surely an interesting and promising aspect as it would be the next natural step to complete the work.

When, at first, problems were encountered to get feedback, the reaction of providing a new way to get it seems appropriate now. However it reduced the time and possibilities to gather many comments. Improvements have been made based on those received, though organizing a larger study to validate the results more broadly would be relevant.

Despite these future improvements, the purpose of providing, for the first time, a tool to help many companies understand SCM through their problems seems fulfilled.

Conclusion

The product resulting from this study consists of a framework divided in the three following parts:

- A list of symptoms encountered by companies involved in software development. Each symptom includes a title, a description and a discussion. These symptoms can be understood by people with no knowledge in configuration management.
- A list of SCM problems. These problems are related to configuration management because they can be solved by it. Each problem includes a title and a description. Most of them are largely described in the Configuration Management literature.
- A list of mappings. Once symptoms and problems were gathered the key element of the study was to relate them. Mappings link the two groups, describe and motivate the different choices made.

Interviewing software professionals helped to create a collection of up-to-date software development symptoms. Gathering knowledge from various SCM books supported the construction of a list of configuration management problems. By mixing these pieces of information, creating groups of symptoms and mapping them with problems, a framework satisfying the stipulated requirements has been built step by step. Constructing on these sources of data partly legitimizes the results, at least for the origins of the symptoms and problems. Moreover feedback has been organized and distributed through the website. The replies given by different companies reenforced the validity of the product and showed some possibilities for future improvements.

Companies now have a framework to help them understand their problems related to Configuration Management. The product is accessible in the appendices or online at www.cmcheckup.org. Companies can freely use it and Configuration Management experts can collaborate to improve the current results. Today the product explains a large number of problems, that could be extended in the future with further studies.

Appendices

Appendices A, B and C are respectively the list of symptoms, the list of problems and the list of mappings forming the three parts of the framework. Symptoms and problems are accompanied by their descriptions which were not included in the discussion. Mappings are included to provide a complete framework even if they are already available in the design. These parts are placed in appendix due to their irrelevance in the discussion carried on in chapter 2. Nevertheless they represent the product resulting of the study and the pertinent part for companies.
The same pattern is followed for the three lists; each element is composed by an identifier, a name and a description. However no specific order is followed inside the lists.

Appendix A: Symptoms

(S1) Is the project on time?
The development team (or the project manager at least) has to know the status of the project to be able to take relevant decisions when needed. Plans might change when a project is late: the project manager could decide to refocus on critical tasks, postpone secondary tasks or features…
What is missing to have these information? Do the developers report their work? Is the information easily accessible?

(S2) The product does not meet the requirements
According to tests or to customers, the application does not match the requirements. Were the requirements well defined at the beginning? Did the developers understood them? Were did the communication break?
This symptom is not related to Configuration Management in every case.

(S3) Productivity decreased after a process change.
A configuration manager decided to change a process after monitoring it and evaluating it could be improved. This is part of his job. However the productivity decreased after the change. Why did the productivity decreased? Was the previous process better? Did the team get enough time the learn the new process? A productivity drop after such a transformation is normal, by getting the appropriate training, it might become better in the future. The configuration manager has to judge quickly in the situation to take the appropriate decision.

(S4) The program does not work anymore.
Developers rarely produce perfect code at their first try. This is not a problem as long as they do not share their modifications before testing them and make sure it does not create immediate problems. Otherwise it might compromise the work of the other members of the team.

This symptom can be detected by picking the latest version of the program and trying to build and run it.

(S5) An error cannot be reproduced in a configuration.
Bugs should be fixed fast. With an accurate bug report, the problem should be easily reproducible by selecting the right version of the product in cause and by following the instructions to reproduce the bug. Here the person in charge of the support will loose much time trying to recreate it. A piece of information is missing in puzzle. Which one?

(S6) A change disappeared.
A developer made a change but cannot find it anymore. He cannot spend time looking for something he made before or, even worse, doing it again. Once modifications are done and submitted to the common repository, an easy way to track them should exist.

(S7) The documentation does not match the program.
Versioning has been organized for sources files and documentations. The development team can select easily the right version of one file. It is however difficult to relate the right revision of the source code of a file or component with its corresponding documentation. A manual follow up often takes place. What is the use of the documentation nobody knows to which version of the program it refers? How can Configuration Management support this issue?

(S8) Do I have the right version?
Developers do not know on which version of the program they are working on. Depending on the current task, the required version might not be the usual or the latest.
What version do I have? Is it possible to answer the question?

(S9) Has this bug been fixed?
The support team has a bug report but do not know if it has already been corrected, i.e. the status of the task is not available.
The symptom occurs when the status of the bug fixes cannot be known.

(S10) Which modifications have been included in this build?
The content of a specific build cannot be known. Nevertheless the team needs this information to answer several questions: How can the new features be communicated to the new customer? Should the program be rebuild to integrate these changes or is it already included?

(S11) Has this change been tried?
Can the team know the answer of this question? The possibility to know if a specific modification has been tried is not offered. Is a historic of the different modifications available?

(S12) Who is responsible for this modification?

How can a team member know the identity of the person in charge of a modification? Who should people refer to if this information is not available when they need more information about the modification?
Is there a plan available for future work? Are these information easily accessible for the entire team?

(S13) Have these modifications been integrated?

This symptom occurs when the integration is not directly done by the developers. The status of the integration process is not available. This is especially a problem when the process takes more time than expected; somebody could have forgotten to integrate a feature or a problem delayed the integrator's work.
Can the status of the integration process be known?

(S14) Merge problem - Changes have to be carried out manually.

The team is working on several versions of the program, e.g. development or production. Modifications made on a branch have to be reported manually to other branches. It creates much unnecessary work. Why is this task done manually? Are the results of the automatic merge unusable? Is the merge tool not good enough? Is the merge history usable? Is the branching model too complicated? Will the developers always remember to copy every change? What will happen if they forget?

(S15) Modifications are not replicated to other branches.

Developers do not copy their modifications in the other versions of the program when they should. Does the automatic merge create problems which discourage developers to report their changes? Is the branching model to complicated to always know where modifications have to be reported?

(S17) How to create a specific configuration?

The program developed offers the possibility to be configured under a user request. Depending on its needs, different components or parameters for these components can be used. How can this specific configuration be created? Does a process exist to guide the team to easily build the required program?

(S18) How these versions differ from each other?

The difference between two version of a file or a program cannot be known easily. This information needs to be accessible by the team to know on which one they should work or to be able to tell their customer which features are included. The system used can help to know the version of files or programs but not to compare them.

(S19) Which version of the product has been delivered to the customer?

In order to understand the problems encounter by a customer or to tell him the features added in the new version compare to his, the current version of the product he is using has to be known. It is a required information to assure the maintenance for this customer as well.

(S20) How has this configuration been created? What does it contain?
After building a configuration, someone would like to know the conditions in which it has been created and what has been included e.g. to understand the problem faced by an user.
Has it been compiled using the right environment and parameters? Which version of the source code has been used? Are these details available?

(S21) Were the regression tests performed?
Regression tests should be run before submitting changes to the common repository. Is the status of this task available? Were there successful? Is the quality of the product assured by using this kind of tests? Does the team want risks of further problems want to be limited?

(S24) Build composition.
The team might not want to include all the latest modifications when they are compiling the program. Can they be excluded? Is it possible to easily select the components or features to include in a build?

(S26) The unknown branch.
Why does this branch exist? This symptom does not create direct problems since the branch can be left alone. It shows the confusion within the project and might hide deeper troubles concerning the branching strategy. Can this branch be safely deleted? What is the branching strategy? Has it been respected when creating this branch? Why cannot its origin be known?

(S29) Everybody works on his own branch or copy of the program.
Each developer is working on his private branch without paying attention to his colleagues ‘work. Many conflicts will surface during the integration of the different parts.
This situation should be avoided unless the system has been designed around this principle (e.g. distributed version control system).

(S30) The same version of a program has to be modified in different places.
Developers have to copy their changes in different places for their coworkers to get them.

(S31) Developers are working simultaneously on the same copy of a program.
Developers are working directly on a shared copy of the program hosted on a server. This situation will disturb developers as the program they are working on will be modify without them knowing how, where or when.

(S32) The program breaks after a successful merge.
The program breaks after a successful merge.

(S36) Were the tests run?
Tests have been created. Good! Are they used? Do we know their status?

Appendix B: Problems

(P1) Double maintenance
The problem occurs when the same version of a program, component or file has to be maintained in different places. More information can be found in W. Babich book [Babich86].

(P2) Shared data
The problem occurs when several developers are working on the same copy of a program and their modifications are interfering with each other. More information can be found in W. Babich book [Babich86].

(P3) Simultaneous update
Two developers checkout a component. They both work on it. The first one commit his modifications. When the second checkin his modifications, he will erase the one made by the first developer. More information can be found in W. Babich book [Babich86].

(P4) Logical conflict.
When changes are committed, a component or part of the program that has not been modify stops working.

(P5) Lack of training - Developer part.
Developers do not follow the implemented processes because they do not know or understand them.

(P6) Lack of merge history or powerful merge model.
The merge history is not complete or the merge model is problematic. This results to problems when doing merges. The merge tool could do better job using a tool which record all the required data for the process. If the merge model is too complex, it will probably creates problems too. The merge model should stay simple.

(P12) Identification problem.
Problem related to Configuration Identification. It can be that nothing has been implemented or the team do not apply parts of it...

(P13) Lack of CM training - Manager part.
Managers are doing mistakes, are not using CM correctly because it has not been explained to them and they do not understand it.

(P19) Communication problem.
The communication broke within the team or with the customer resulting to a lack of information. This will create further problems.

(P20) Reproducibility problem - Lack of Bill-Of-Material.
A part of the information required to reproduce a specific configuration is missing. A Bill-Of-Material is the set of information required to build the same configuration. The company need to define one.

(P21) No status accounting.
The team does not get enough information and feedback on their work. Using Configuration Status accounting will provide the required information.

(P22) Bad branching strategy.
The branching strategy used is too complex. It creates difficulties to know the purpose of each branch or how they should be merged. This complexity will also lead to merge problems.

(P23) No checkin rules.
Giving a set of rules to the developers before they checkin their changes in the common repository will limit the risk of spreading bad or broken code.

(P25) Unavailable or undocumented feature in the SCM tool.
The team requires a feature that is not available in the tool currently used. Another solution can be found, but in the end if the tool is really lacking in performance, getting a new tool might be the right solution.

Appendix C: Mappings

• M1: S1 (Is the project on time?)
--> P21 (No status accounting)
Related symptoms: S9, S11, S12, S21, S36.

• M2: S2 (The product does not meet the requirements)
--> P19 (Communication problem)
Related symptoms: S8, S10, S17, S19, S24.

• M3: S3 (Productivity decreased after a process change)
--> P5 (Lack of CM training - Developer part)

--> P13 (Lack of CM training - Manager part)
Related symptoms: S6, S8, S14, S24, S26.

• M4: S4 (The program does not work anymore)
--> P2 (Shared data)
--> P4 (Logical conflict)
--> P23 (No checkin rules)
Related symptoms: S21, S31, S32, S36.

• M5: S5 (An error cannot be reproduced in a configuration)
--> P19 (Communication problem)
--> P20 (Reproducibility problem - Lack of BOM)
Related symptoms: S17, S19, S20.

• M6: S6 (A change disappeared)
--> P3 (Simultaneous update)
--> P5 (Lack of CM training - Developer part)
--> P12 (Identification problem)
Related symptoms: S8, S15, S20, S26, S33.

• M7: S7 (The documentation does not match the program)
--> P12 (Identification problem)
Related symptoms: S6, S8, S10, S18, S19, S20.

• M8: S8 (Do I have the right version?)
--> P5 (Lack of CM training - Developer part)
--> P12 (Identification problem)
Related symptoms: S6, S7, S10, S18, S19, S20, S33.

• M9: S9 (Has this bug been fixed?)
--> P21 (No status accounting)
Related symptoms: S1, S11, S12, S13, S21, S36.

• M10: S10 (Which modifications have been included in this build?)
--> P12 (Identification problem)
--> P20 (Reproducibility problem - Lack of BOM)
Related symptoms: S17, S20, S24.

• M11: S11 (Has this change been tried?)
--> P21 (No status accounting)
Related symptoms: S9, S12, S21, S36.

• M12: S12 (Who is responsible for this modification?)
--> P19 (Communication problem)
--> P21 (No status accounting)
Related symptoms: S1, S2, S9, S11, S13, S21, S36.

• M13: S13 (Have these modifications been integrated?)
--> P19 (Communication problem)
--> P21 (No status accounting)
Related symptoms: S6, S9, S12, S13.

• M14: S14 (Merge problem - Changes have to be carried out manually)
--> P5 (Lack of CM training - Developer part)
--> P6 (Lack of merge history / powerful merge tool)
--> P22 (Bad branching strategy)
Related symptoms: S6, S15.

• M15: S15 (Modifications are not replicated to other branches)
--> P5 (Lack of CM training - Developer part)
--> P22 (Bad branching strategy)
Related symptoms: S6, S14, S29.

• M17: S17 (How to create a specific configuration?)
--> P5 (Lack of CM training - Developer part)
--> P20 (Reproducibility problem - Lack of BOM)
Related symptoms: S5, S10, S20.

• M18: S18 (How these versions differ from each other?)
--> P12 (Identification problem)
Related symptoms: S6, S7, S8, S10, S20, S26.

• M19: S19 (Which version of the product has been delivered to the customer?)
--> P12 (Identification problem)
--> P21 (No status accounting)
Related symptoms: S7, S8, S9, S10, S13.

• M20: S20 (How has this configuration been created? What does it contain?)
--> P12 (Identification problem)
--> P20 (Reproducibility problem - Lack of BOM)
Related symptoms: S5, S10, S17.

• M21: S21 (Were the regression tests performed?)
--> P21 (No status accounting)

Related symptoms: S12, S13, S36.

• M24: S24 (Build composition.)
--> P5 (Lack of CM training - Developer part)
--> P25 (Unavailable or undocumented feature in the SCM tool.)
Related symptoms: S10, S17, S20.

• M26: S26 (The unknown branch.)
--> P5 (Lack of CM training - Developer part)
--> P12 (Identification problem)
--> P22 (Bad branching strategy)
Related symptoms: S8, S18.

• M29: S29 (Everybody works on his own branch or copy of the program.)
--> P1 (Double maintenance)
--> P22 (Bad branching strategy)
Related symptoms: S15, S26, S30.

• M30: S30 (The same version of a program has to be modified in different places.)
--> P1 (Double maintenance)
Related symptoms: S29.

• M31: S31 (Developers are working simultaneously on the same copy of a program)
--> P2 (Shared data)
Related symptoms: S4.

• M32: S32 (The program breaks after a successful merge.)
--> P4 (Logical conflict)
Related symptoms: S4, S21, S36.

• M36: S36 (Were the tests run?)
--> P21 (No status accounting)
Related symptoms: S9, S11, S12, S21.

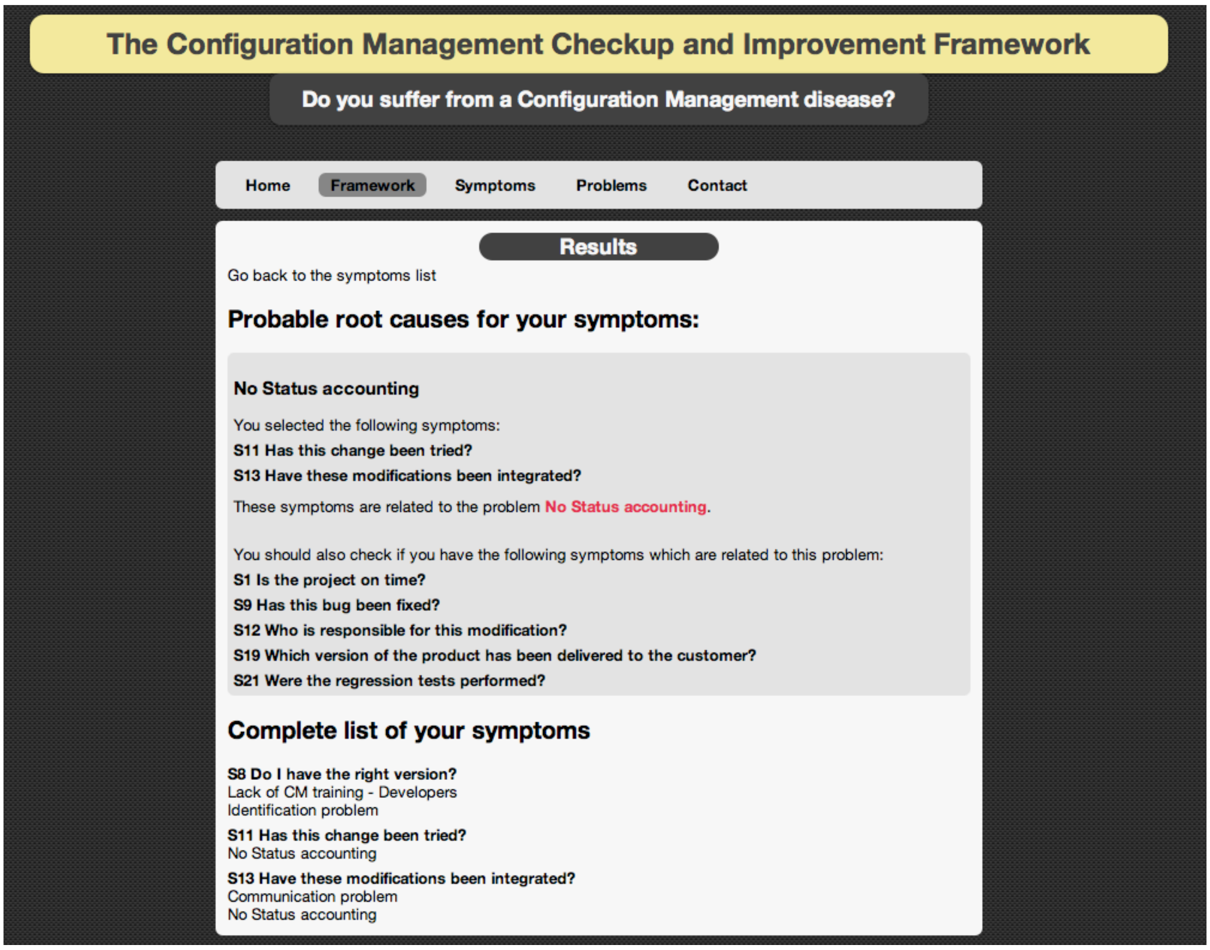
Appendix D: User manual

The framework can be used online at www.cmcheckup.org or with the material provided in the previous appendices. As explained before the website makes the framework easier to use. Moreover in the future, improvement may be brought on the website only and will not be reported in this paper.

The use of this framework is very simple. A company has to check its symptoms from the list provided. A careful job should be done on that part by considering each symptom which should be well understood to make sure they are encountered or not. The company has to ponder their importance, privileging the most important. A symptom in the framework might not be relevant to cure in a specific situation or company. This analysis is the difficult part.

Then the mappings can be used to reveal the root causes of the different symptoms. The application takes care of this process on the website when it will be a long task on paper. The underlying problems will then be available and can be analyzed using the short descriptions given to facilitate their understanding. An interesting feature is the incorporation of related symptoms that can help to find other symptoms that were not selected the first time and complete the analysis.

Appendix E: Website


The Configuration Management Checkup and Improvement Framework
Do you suffer from a Configuration Management disease?
Home
Framework
Symptoms
Problems
Contact
Results

Go back to the symptoms list

Probable root causes for your symptoms:

No Status accounting

You selected the following symptoms:

S11 Has this change been tried?

S13 Have these modifications been integrated?

These symptoms are related to the problem **No Status accounting**.

You should also check if you have the following symptoms which are related to this problem:

S1 Is the project on time?

S9 Has this bug been fixed?

S12 Who is responsible for this modification?

S19 Which version of the product has been delivered to the customer?

S21 Were the regression tests performed?

Complete list of your symptoms

S8 Do I have the right version?
Lack of CM training - Developers
Identification problem

S11 Has this change been tried?
No Status accounting

S13 Have these modifications been integrated?
Communication problem
No Status accounting

References and background material

- [Babich86] Wayne Babich, *Software Configuration Management: Coordination for team productivity*, 1986
- [Brooks75] Frederik Brooks, *The mythical man-month*, 1975
- [Tichy88] Walter Tichy, *Software Configuration Management Overview*, 1988

The following literature helped the reflexion but are not directly used as references. It can be good references for future work.

- [Bendix98] Lars Bendix, *How to introduce maturity in SCM*, 1998
- [Bendix00] Lars Bendix, *Continuous improvement of the configuration management process*, 2000
- [Bendix01] Lars Bendix, *A disciplined approach to change management*, 2001
- [Bendix05] Lars Bendix, Lorenzo Borracci, *Toward a suite of software configuration management metrics*, 2005
- [Berczuk97] Steve Berczuk, *Teamwork and Configuration Management*, 1997
- [Berczuk02] Steve Berczuk, Brad Appleton, *Software Configuration Management Pattern*, 2002
- [Borracci05] Lorenzo Borracci, *A return on investment Model for Software Configuration Management*, 2005
- [Brown99] William Brown, Hays McCormick, Scott Thomas, *AntiPatterns and Patterns in SCM*, 1999
- [Colville08] Ronni J. Colville, *Establishing Processes for Configuration Management*, 2008
- [Frühauf99] Karol Frühauf, Andreas Zeller, *Software Configuration Management: State of the Art, State of the Practice*, 1999
- [Iizuka06] Masashi Iizuka, *Request and Main Problems Regarding Configuration Management in Open System Development - Management in short-term, congested development*, 2006
- [Jalote02] Pankaj Jalote, *Software Project Management in Practice*, 2002
- [Kern00] Harris Kern, Stuart Galup, Guy, Nemiro, *IT Organization: Building a World-class Infrastructure*, 2000
- [Klibor98] Heinz-Uwe Klibor, Heinz-Eckhar Gödecke, *Evaluation of the Impact of SW Configuration Management as a key facto for Improved SW Quality*, 1998
- [Larsen98] Jens-Otto Larsen, Helge Roald, *Introducing ClearCase as a Process Improvement Experiment*, 1998
- [Moreira99] Mario Moreira, *The 3 Software Configuration Management Implementation Levels*, 1999

- [Vanhanen97] Jari Vanhanen, *Improving configuration management processes of a software product*, 1997
- [Visconti02] Marcello Visconti, Rodolfo Villaroel, *Managing the improvement of SCM Process*, 2002
- [Wasson] Jack Wasson, *CM for the 21st century*
- [Watts09] Frank Watts, *Configuration Management Metrics*, 2009