Software Process Improvement Using Groupware

Supporting Distributed Cooperation in Software Development

Gustav Nordenskjöld
Harm Andries Kingma

Department of Software Engineering and Computer Science
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
This thesis is submitted to the Department of Software Engineering and Computer Science at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full time studies.

**Contact Information:**

Authors:
Gustav Nordenskjöld
Contact: gustav@nordenskjold.com

Harm Andries Kingma
Contact: hakingma@hotmail.com

University advisors:
Stefan Johansson
Department of Software Engineering and Computer Science

Yvonne Dittrich
Department of Human Work Science, Media Technology and Humanities

Department of Software Engineering and Computer Science
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

Internet : www.bth.se/ipd
Phone : +46 457 38 50 00
Fax : + 46 457 271 25
Abstract

This master thesis describes our work with Team Sweden and how we have attempted to perform a Software Process Improvement using groupware. Team Sweden is a distributed academic software development organization doing research on artificial intelligence and robotics. Such an organization has a lot to gain from the introduction of a more structured process model. We have investigated the possibility of supporting the development processes of a small software development organization by introducing groupware.

Only introducing a groupware tool or technique is not enough to improve the processes of an organization. Process improvement requires an infrastructure and groupware could be a useful tool for establishing it. Improved communication and knowledge sharing through groupware can be beneficial for establishing a process infrastructure. Groupware should be used in conjunction with other process improvement measures in order to be effective and improve the capability of the targeted organization.

**Keywords:** Team Sweden, Groupware, Software Process Improvement, Computer Supported Cooperative Work.
Acknowledgments

First of all we would like to thank our advisors, Stefan and Yvonne, for their help and guidance. Their expertise has been a great help to us throughout this project.

We would also like to thank Kevin and Alessandro for answering all our questions and taking the time to help us, despite their busy schedule.

Thank you Ted for supplying us with the initial version of the Code Standard. This saved us a lot of time and effort and improved the quality of the document.

And last but not least, we would like to extend our gratitude to the rest of Team Sweden. We hope that they will benefit from our work on this project.
CONTENTS

1 INTRODUCTION...................................................................................................................1

2 BACKGROUND....................................................................................................................2
  2.1 RoboCup.........................................................................................................................2
  2.2 Team Sweden..................................................................................................................2
  2.3 The Robots.....................................................................................................................2

3 THESIS FRAMEWORK........................................................................................................4
  3.1 Problem Description....................................................................................................4
  3.2 Hypothesis.....................................................................................................................4
  3.3 Thesis Outline...............................................................................................................4

4 THEORETICAL BACKGROUND..........................................................................................5
  4.1 Software Process Improvement....................................................................................5
    4.1.1 The Software Process..............................................................................................5
    4.1.2 Software Process Improvement..............................................................................5
    4.1.3 Software Process Assessment................................................................................6
  4.2 Groupware....................................................................................................................6
    4.2.1 Disparity in Work and Benefit..............................................................................6
    4.2.2 Critical Mass...........................................................................................................7
    4.2.3 Local Optimization..................................................................................................7
    4.2.4 Disruption of Social Processes and Current Way of Working.......................8

5 CURRENT SITUATION........................................................................................................10
  5.1 Organizational Structure..............................................................................................10
  5.2 The Software Architecture..........................................................................................10
  5.3 Development Process..................................................................................................11
  5.4 Organizational Maturity..............................................................................................12
  5.5 Current Process Infrastructure.....................................................................................13

6 RESEARCH METHODOLOGY, PLANNING AND EVALUATION.........................................14
  6.1 Action Research..........................................................................................................14
  6.2 Interview Methodology...............................................................................................15
  6.3 Project Plan..................................................................................................................15
  6.4 Data Collection............................................................................................................16
  6.5 Documentation of Collected Data..............................................................................18
  6.6 Evaluation of Research Methods.................................................................................18
    6.6.1 Action Research.....................................................................................................18
    6.6.2 The Unstructured Interview..................................................................................18

7 PROBLEMS AND SOLUTIONS..........................................................................................20
  7.1 Positive Aspects............................................................................................................20
  7.2 Configuration Management.........................................................................................20
7.2.1 Problem Description........................................................................................................21
7.2.2 Support Proposal.............................................................................................................21
7.2.3 Implementation..............................................................................................................21
7.2.4 Feedback and Opinions..................................................................................................22
7.3 Communication..................................................................................................................22
7.3.1 Problem Description........................................................................................................22
7.3.2 Support Proposal.............................................................................................................22
7.3.3 Implementation..............................................................................................................23
7.3.4 Feedback and Opinions..................................................................................................23
7.4 Source Code Documentation and Taxonomy....................................................................24
7.4.1 Problem Description........................................................................................................24
7.4.2 Support Proposal.............................................................................................................24
7.4.3 Implementation..............................................................................................................25
7.4.4 Feedback and Opinions..................................................................................................25
7.5 Knowledge Management..................................................................................................26
7.5.1 Problem Description........................................................................................................26
7.5.2 Support Proposal.............................................................................................................27
7.5.3 Implementation..............................................................................................................27
7.5.4 Feedback and Opinions..................................................................................................28
7.6 Other Problems..................................................................................................................28
7.6.1 Developer Turnover..........................................................................................................28
7.6.2 Time Management..........................................................................................................29
7.6.3 Complex Development Dependencies...........................................................................29
8 Discussions..........................................................................................................................31
8.1 Embracing Change..............................................................................................................31
8.2 Introducing and Conducting an Improvement Project......................................................31
8.3 Distribution of Effort..........................................................................................................33
8.4 Increasing Productivity.......................................................................................................33
8.5 Groupware and Team Sweden...........................................................................................34
8.5.1 Disparity in Work and Benefit.........................................................................................35
8.5.2 Critical Mass..................................................................................................................35
8.5.3 Local Optimization.........................................................................................................36
8.5.4 Disruption of Social Processes and Current Way of Working.....................................36
8.6 Metrics and Measurements...............................................................................................36
8.7 Applicability to Other Areas.............................................................................................37
9 Future Work..........................................................................................................................38
9.1 Continuous Process Improvement.........................................................................................38
9.2 Open Source Organizations.................................................................................................38
9.3 Personal Software Process..................................................................................................38
10 Conclusion............................................................................................................................40
## APPENDICES

Appendix A  “Team Sweden Code Standard and Developer Guidelines”

## LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>AASS</td>
<td>center for Applied Autonomous Sensor Systems</td>
</tr>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>AIBO</td>
<td>Sony Entertainment Robot</td>
</tr>
<tr>
<td>CMM</td>
<td>Capability Maturity Model</td>
</tr>
<tr>
<td>CVS</td>
<td>Concurrent Versions System</td>
</tr>
<tr>
<td>FAQ</td>
<td>Frequently Asked Questions</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>HTML</td>
<td>Hyper Text Markup Language</td>
</tr>
<tr>
<td>OSS</td>
<td>Open Source Software</td>
</tr>
<tr>
<td>PSP</td>
<td>Personal Software Process</td>
</tr>
<tr>
<td>SEI</td>
<td>Software Engineering Institute</td>
</tr>
<tr>
<td>SEPG</td>
<td>Software Engineering Process Group</td>
</tr>
<tr>
<td>SPI</td>
<td>Software Process Improvement</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

Developing software products is a complex and demanding process. Systematic practices are needed, from idea to final product, in order to produce a high quality product. Qualitative software is produced by applying a systematic, disciplined and quantifiable approach to the entire development process. The larger the organization, the higher the demand for a disciplined development process. This includes both technical and managerial aspects of the organization.

Any software process can be controlled, measured and improved. Software process improvement is performed through planning, developing and implementing changes to the development process. An improved software process should lower costs, increase productivity and result in higher quality products. Software process improvement is commonly performed in an commercial setting within larger companies where the benefits of an improved software process are obvious and profitable. This is the reason why most software process improvement models are targeting large commercial organizations. But even smaller software organizations can benefit from a more structured development process, both in terms of efficiency and quality of the work. We believe that it is possible to improve and support the development processes of smaller organizations and increase both the productivity and efficiency of their work. In order to prove our thesis we have been working with a small distributed academic organization called Team Sweden.

Team Sweden is a Swedish national effort to produce a team of soccer playing physical robots able to participate in the four-legged league of the RoboCup international competition. The team is a cooperative effort between several universities in Sweden; Örebro University, Lund University and Blekinge Institute of Technology. They are also collaborating with the University of Murcia in Spain. The goal of the team is to develop general principles and algorithms for autonomous robot operation in unpredictable environments that can be re-used in different robots and different domains.

Team Sweden utilizes a loose organizational structure without clearly defined processes. A large turnover of developers, combined with the distributed collaboration between four different locations have created complications during their work. A complex organization such as Team Sweden would have a lot to gain from the introduction of a more structured process model. The topic of this thesis is to investigate if it is possible to, within a limited time frame, introduce a number of tools and techniques that would benefit the work of Team Sweden and improve their efficiency. One way of doing this would be to introduce and implement groupware tools and techniques that are commonly used in Computer Supported Cooperative Work. This thesis describes our work with Team Sweden and how we have attempted to perform a software process improvement using groupware.
This chapter will give some background information about RoboCup, Team Sweden and the robots used by the team.

2.1 RoboCup

RoboCup (originally called the Robot World Cup Initiative) is an international research and education initiative. It is an attempt to foster Artificial Intelligence (AI) and intelligent robotics research by providing a standard problem where a wide range of technologies can be integrated and examined. For this purpose, RoboCup chose to use the soccer game as a primary domain, and organizes the Robot World Cup Soccer Games and Conferences. In order for a robot team to actually perform a soccer game, various technologies must be incorporated. These include for example design principles of autonomous agents, multi-agent collaboration, strategy acquisition, real-time reasoning, robotics, and sensor-fusion. RoboCup also offers a platform for research on the software aspects of this problem domain [URL-RC].

2.2 Team Sweden

Team Sweden is a Swedish national effort to produce a team of soccer playing physical robots able to participate in the four-legged league of the RoboCup international competition. The team is a cooperative effort between several universities in Sweden and Spain. The participating entities in 2003 are Örebro University (coordinating node), Lund University, the Blekinge Institute of Technology and the University of Murcia. Team Sweden approaches the RoboCup problem as an entertaining way to do team research work involving important scientific challenges. Of particular interest are the problems of:

- How to cope with the large uncertainty that is inherent in this domain.
- How to integrate higher level cognitive processes with lower level sensory-motor processes.
- How to integrate action and perception in a complex and dynamic environment.

The solutions that are investigated are intended to be generally applicable to any system presenting the above characteristics. The goal of the team is to develop general principles and algorithms for autonomous robot operation in unpredictable environments that can be re-used in different robots and different domains [URL-TS].

2.3 The Robots

The robot used by the four legged league is the Sony Entertainment Robot (AIBO), which is depicted in figure 1. Developed and built by Sony, the AIBO family of entertainment robots includes the ERS–110, ERS–210 and the new 210A-series. The ERS–110 was the first consumer model sold by Sony and the version used in the 1999 and 2000 RoboCup tournaments. In late 2000, the ERS–210 was released, with improved CPU and updated operating system. In 2002 the 210A was released, which doubles the processor capacity.
The hardware platform of the AIBO robot is a challenging and exciting one. The robot is equipped with several sensors, including – but not limited to – several pressure sensors, a camera, microphones, acceleration sensor, temperature sensor and vibration sensor. The ERS–210 is modularly built, with all four legs and head easily removable for replacement [URL-AI].
3 Thesis Framework

This chapter describes the thesis framework, the problems addressed in this thesis and the research questions. We will also give a short summary of the structure of the thesis.

3.1 Problem Description

Team Sweden utilizes a loose organizational structure, without clearly defined processes and practices. A large turnover of developers, combined with the distributed collaboration between four different locations have created complications during their work. The team also suffers from a lack of resources, has complex development dependencies and deals with advanced research problems. Although the current way of working has been effective in the past, the team would benefit from improved development processes. A complex organization such as Team Sweden has a lot to gain from the introduction of a more structured process model.

Increasing the maturity level of Team Sweden would require a lot of time and effort. Supporting a structured software process would also require an extensive process infrastructure, something that the team does not have. We realized that it would not be possible for us, within the scope of a master thesis project, to perform an extensive process improvement on Team Sweden. However, it would be possible to introduce a number of tools and techniques that would benefit their work and improve the efficiency of the team. We decided that the best way to do this would be to support the cooperation and communication within the team. Groupware tools and techniques could be a solution but would not be sufficient. We would also have to define a set of development processes that could be followed and enforced. These terms and others will be described in chapter 4 – Theory.

3.2 Hypothesis

It is possible to improve the development processes within a distributed software development organization, such as Team Sweden, through the introduction of groupware support solutions.

3.3 Thesis Outline

This thesis is structured in three abstract parts. Chapter one through five will give some theoretical background of our work and the Team Sweden organization. These chapters explain the concept of software process improvement and link it with groupware related issues. We will also give some in depth information about the organizational structure of Team Sweden and their current development processes.

Chapter six and seven will describe our work and results. These chapters discuss the problems we identified in the Team Sweden organization and our proposed changes.

The last part consists of chapter eight through ten and discusses our work with the team. We will also give some recommendations for future improvements and finish with a conclusive summary of our work.
4 THEORETICAL BACKGROUND

This chapter will give a theoretical background for the ideas and concepts that are applied in the thesis and also explains the terminology that is used. We will first explain software process improvement - how and why it is employed. In order for a software process improvement to be successful, it is important that the communication and cooperation within a distributed organization are improved. Computer Supported Cooperative Work and groupware could be one possible way of accomplishing this. Groupware tools and techniques will be described through the perspective of some of the problems that are common when introducing them into an organization.

4.1 SOFTWARE PROCESS IMPROVEMENT

This section describes the software process and how it can be improved to increase the quality and lower the cost of software products.

4.1.1 THE SOFTWARE PROCESS

A software process is defined as a set of activities, methods, practices and transformations that people use to develop and maintain software and the associated products [SEI93b, A-19]. This includes the entire development cycle, from design and implementation to quality assurance and a finished product. Process thinking is different from the conventional way of thinking as it aligns the behavior and activities of a group towards a common goal [ZAH98, p. 4]. It brings consistency and uniformity to the organization and should result in improved capability and better quality in the products. A disciplined process is performed in an ordered and consistent pattern of behaviors. The process needs to be established and institutionalized into an organization in order to be effective [ZAH98, p. 7]. This means the group will perform the process as a team, rather than individuals. The capability of the group will be higher than the sum of all the individual members.

The organizational and management infrastructure of a software organization should cover the responsibility of establishing, monitoring and enforcing the processes [ZAH98, p. 71]. This is usually performed by a Software Engineering Process Group (SEPG) which is responsible for coordinating and maintaining the processes. This could be done either as a part-time or a full-time task depending on the resources of the organization.

4.1.2 SOFTWARE PROCESS IMPROVEMENT

Software Process Improvement (SPI) is used for planning, developing and implementing changes to the software process. The basic principle is that any software process can be controlled, measured and improved [HUMP90, p. 4]. An SPI program is an ongoing task and should be performed in continuous cycles [ZAH98, p. 213]. Each SPI cycle should improve the software process and increase the maturity level of the organization. Humphrey has defined six steps that an organization can use to increase the capability of their software [HUMP90, p. 4].
1. Understand the current status of their development process or processes.
2. Develop a vision of the desired process.
3. Establish a list of required process improvement actions in order of priority.
4. Produce a plan to accomplish the required actions.
5. Commit the resources to execute the plan.

There are a number of standards and initiatives for software process improvement. One of the more popular ones is the Capability Maturity Model (CMM) developed by the Software Engineering Institute (SEI) of Carnegie Mellon University. The SEI promotes the evolution of software engineering from an ad hoc, labor-intensive activity to a discipline that is well managed and supported by technology [URL-SEI]. CMM describes the principles and practices underlying software process maturity and is intended to help organizations improve the maturity of their software processes in terms of an evolutionary path from ad hoc, chaotic processes to mature, disciplined software processes [URL-CMM]. The CMM is organized into five maturity levels: Initial, Repeatable, Defined, Managed and Optimizing. The model is not prescriptive and it does not tell an organization how to improve. It describes an organization at each maturity level without prescribing the specific means of getting there [SEI93a, p. 14]. The CMM is aimed at larger commercial companies but most software development organizations, regardless of size and situation, should be able to benefit from the model.

4.1.3 SOFTWARE PROCESS ASSESSMENT

Software process assessment is defined as an appraisal to determine the state of an organization's current software process, to determine the high-priority software process-related issues facing an organization and to obtain the organizational support for software process improvement [SEI93b, A-19]. A process assessment will help a software organization to identify their critical problems and establish improvement priorities.

4.2 GROUPWARE

In this section a number of problem areas associated with the introduction of groupware will be discussed. This section is based on information from the articles “Eight Challenges for Groupware Developers” [GRUD97] and “Why Groupware Applications Fail - Problems in Design and Evaluation” [GRUD89]. Before being able to identify the different problem areas associated with the groupware domain, a definition of groupware has to be formulated. The term groupware refers to programs that help people work together collectively while located remotely from each other. Groupware services can include the sharing of calendars, collective writing, e-mail handling, shared database access, display information to others, electronic meetings with each person able to see and other activities.

4.2.1 DISPARITY IN WORK AND BENEFIT

In an ideal situation, the introduction of groupware is beneficial for each group member. In reality this is seldom the case since groupware applications never
provide the same advantage to each group member. The actual benefit depends on individual preferences, prior experience with groupware, the role within the team and the type of assignment. This has to be taken into account when introducing any type of groupware. Groupware is meant to be beneficial to everyone involved. However, groupware applications have to be maintained. The people assigned to the task of maintenance might resist against its introduction because they only see the extra work they will have to do. The disparity in work and benefit problem does not limit itself to the people responsible for maintaining the software, but also to the regular users. If they have to do extra work in order to keep the application running, for example the input of several kinds of information in the system, they will also be reluctant to adopt the groupware solution. It is therefore essential that, before the decision is made to introduce a groupware solution, the work versus benefit problem is considered. There should be an obvious benefit compared to the current situation. Otherwise the future users of the system might be reluctant to adopt the solution, which will most likely lead to its failure.

4.2.2 Critical Mass

Groupware is only useful if it is used by all, or at least most, group members. In other words; a critical mass of users has to be reached. The critical mass of users is the minimum amount of users needed to make the groupware application successful. When a groupware solution is rejected by a number of employees its usefulness will be severely reduced. To give an example: When an electronic meeting calendar is not used by one or more project members, it will not function correctly. The main idea behind such a calendar is to schedule meetings at a time most convenient for all participants. However, when some people schedule appointments without registering these with the meeting calendar, the system might plan meetings at times when these people are occupied somewhere else, thereby rendering the system useless. In this case the critical mass of users consists of all employees. Other groupware solutions, like for example a knowledge database, are less sensitive when it comes to the critical mass problem. Nevertheless it is a major factor when it comes to success or failure of groupware solutions.

4.2.3 Local Optimization

When groupware products are used within an organization, so called local optimization type problems can arise. People utilizing the local optimization strategy tend to act in a way that is in his or her own best interest. This could result in a worse situation not only for the group but also for each individual. This will be clarified by relating to the prisoner's dilemma problem as described in [LURA89, p. 94-95]:

Two prisoner's are waiting for their trial. Each prisoner has two options; confess to the crime or not. The prisoners are separated and not able to discuss the problem with each other. The choice made by both prisoners can result in three different situations, namely:
1. Both do not confess, which results one year in prison for each.
2. Both confess, which results in eight years in prison for each.
3. One confesses and the other does not do so, which results in three months in prison for the confessor and ten years for the other one.

This problem can be related to the groupware domain. The example of a knowledge database will be used to do so. In this example there are two people using the knowledge database. When both people are spending some time and effort to update and maintain the database this can be related to the situation of both prisoners confessing to the crime. The one year sentence can be compared to the little extra work both persons have to do. If, however, one person chooses to optimize his own situation, this person will stop updating the database and only freeload information from it. The resulting situation corresponds to one prisoner (the freeloader) confessing and the other not. The freeloader is in the best possible situation (three months) and the other one in the worst (ten years) because of the extra work he has to do. The final situation is that both persons stop with updating the database (both confess), which results in a very bad situation for both people (eight years each) because the database is rendered useless as the information in it becomes obsolete and incomplete. Of course it could be argued that, when the only person that is updating the database stops doing so, the situation improves for this person rather than get worse. The reason being that this person still has all the information available after stopping with updating and has much less work to do. However, from a theoretical point of view, the prisoner's dilemma is a good illustration of the local optimization problem.

From the perspective of an organization wanting to achieve the biggest benefit from the introduced groupware, the preferred situation would be that both people do their share of updating the database. Only in that case the best work versus benefit would be achieved, together with ensuring an equal distribution of effort.

4.2.4 Disruption of Social Processes and Current Way of Working

Groupware may also fail because it interferes too much with the current way of working, or even worse, with the existing social processes. If a groupware application requires a radical change in the way people work, the person involved will only see the extra work coming forth from its introduction and not see the benefits. Therefore they might be reluctant to adopt the solution.

Within organizations, there exist ways of communicating between different employees which have evolved over time. These include formal and informal communication lines. Informal communication lines include for example discussion during coffee and/or lunch breaks. An example of a formal communication line is a weekly status report to the management of the company. The social processes within an organization are closely linked to the existing informal communication lines. If it is customary within the organization to solve problem by conducting informal face to face meetings, the employees will for example be reluctant to accept a video conferencing application to replace the
informal meetings because the social aspect of the meetings will be lost. This example illustrates how groupware can disrupt the social processes within an organization.
5 CURRENT SITUATION

In this chapter a general description of the current situation of Team Sweden will be given. First the organizational structure of the team will be discussed. Then the current software development process of Team Sweden will be described. Finally, the organizational maturity and process infrastructure will be discussed.

5.1 ORGANIZATIONAL STRUCTURE

We will now shortly describe the current organizational structure of Team Sweden, which is depicted in figure 2. Team Sweden is currently divided over four different geographical locations, namely: Örebro, Lund, Ronneby and Murcia. These locations are referred to as nodes. The team leader, located in Örebro, is responsible for coordinating the team effort in order to reach the goals of the team. Each node has a node coordinator, which is responsible for managing and coordinating the team effort at that location. The team leader also functions as the node coordinator for Örebro. Finally, each node has a number of master and post graduate students doing their thesis work for Team Sweden.

![Organizational structure of Team Sweden](image)

Figure 2 - Organizational structure of Team Sweden

5.2 THE SOFTWARE ARCHITECTURE

In order to clarify the software development process of Team Sweden, the software architecture used in the robots will now be described shortly [URL-TS03]. The Team Sweden software architecture is layered for autonomy inspired by the Thinking Cap, an autonomous robot architecture based on fuzzy logic in use at the center for Applied Autonomous Sensor Systems (AASS) of the Örebro University. The architecture is depicted in figure 3.
The lower layer (Commander module) provides an abstract interface to the physical functionalities of the robot. The middle layer is responsible for maintaining a local representation of the space around the robot (Perceptual Anchoring Module), and for implementing a set of robust tactical behaviors (Hierarchical Behavior Module). The higher layer maintains a global map (Global Map module) of the field and makes real-time strategic decisions (Reactive Planner module). At all levels, fuzzy logic is used to cope with the inherent uncertainty in the domain.

Because of its modular nature, the software architecture is well suited to the Team Sweden organization. This enables the team to distribute the development of the software modules over the different nodes of Team Sweden. Currently the Hierarchical Behavior Module and Commander module development is done in Örebro. The Reactive Planner is developed partly in Lund and partly in Ronneby. The Perceptual Anchoring Module is developed in Örebo and Lund. The Global Map Module is also developed in Örebro. Murcia is working on the software for a goalkeeper.

![Figure 3 – The Team Sweden software architecture](image)

### 5.3 Development Process

Each year several team meetings are held. The purpose of the first meeting is to determine what work has to be done that year. Additionally, the identified work is divided into separate tasks. Within the team, decisions are made by means of reaching a consensus. No one has the authority to decide who has to do what. Tasks have to be done on a voluntary basis. Therefore the identified tasks are assigned to team members volunteering to complete the task in question. After identifying and assigning the different tasks, a plan is made describing who is doing which task and when (parts of) the tasks have to be finished.
After this meeting, the team members more or less work on their own until the next meeting. Each developer works on one or more modules involved in accomplishing his or her task. It is rare that more than one developer works on the same module. During this time the interaction between the different team members consists mainly of peer-to-peer communication. Whenever a developer has a problem which needs the input of another team member, he discusses the problem with that person in order to solve the problem. This discussion is usually conducted through email when the involved parties are located at different nodes. Discussions between people from the same node are conducted by telephone, email or a face to face meeting.

Some time before the competition the team meets again for a whole week. During this week the different modules are integrated and the planning is updated according to the current status of the project. When this meeting has ended, the team has a few additional short meetings before the the competition. The purpose of these meetings is to make some final adjustments to the software in order to make it suitable for use in the competition.

The first thing that is done by the team when they have arrived at the competition site is to calibrate and fine-tune the dogs in order to make them work at the competition site. Also, last minute changes are made to the software and remaining problems are solved if possible. During the actual competition it may well be that some parts of the software fail to operate correctly. If possible, the team will attempt to fix these errors on site between the matches. During the competition the performance of the robots is logged in order to identify future areas of improvement and new strategies and approaches. Additionally, the competitions of other teams are monitored in order to determine how they perform and to try and discover other approaches and ideas concerning the robot soccer domain. After the competition not much work is done until the start of next season, which starts in January.

5.4 ORGANIZATIONAL MATURITY

Although CMM is designed for use in large commercial organizations, most of it is applicable to a smaller academic research group such as Team Sweden. Many of the Key Process Areas mentioned in CMM are as useful for a small organization in an academic context as for any large commercial software company. However, Team Sweden has some specific conditions that make other Process Areas redundant.

Team Sweden has experienced problems due to the current organizational structure and the lack of an institutionalized development process. We have assessed that they are at the Initial maturity level according to CMM. Their processes are ad hoc and success is dependent on individual efforts rather than the combined effort of the entire team. The team handles a crisis through abandoning the plans and instead resorts to coding and testing [SEI93b, O-14]. They utilize a loose organizational structure, without defined processes and practices. A large turnover of developers, combined with the distributed collaboration between four different locations has also created complications during their work. The team also suffers from a lack of resources, complex development dependencies and advanced research problems. These problem areas and others will be further described in chapter 7.
Although the current way of working has been effective in the past, the team would benefit from improved development practices and a more structured process. Increased process discipline in the organization would be likely to improve the quality of their work [ZAHR98, p. 24]. It would also increase the predictability of the organization and any problems that might arise will be more maintainable. The total capability of the team would improve if the development processes would align the members towards a common goal [ZAHR98, p. 21].

5.5 **Current Process Infrastructure**

A process infrastructure is necessary to enable and facilitate the activities and to support process-related roles and responsibilities [ZAHR98, p. 83]. Problems related to poor development processes can be overcome through a focused effort on building an effective process infrastructure based on engineering and management practices [SEI93b, O-1]. At Team Sweden's current maturity level, it is lacking an infrastructure that supports the software process. In order to maintain an effective software process within Team Sweden, a process infrastructure needs to be established.

An infrastructure consists of tools and techniques as well as organizational and managerial aspects. Since most organizational and managerial aspects of Team Sweden are outside the scope of our project, we decided to introduce tools and techniques that will support the process infrastructure. Our aim is to improve and support the Team Sweden process infrastructure by introducing some groupware tools and techniques.
6 RESEARCH METHODOLOGY, PLANNING AND EVALUATION

In this chapter a description will be given of the research-methodologies and -planning used throughout the project. First, the research methodology chosen for the project will be discussed. This discussion will also motivate why this method was seen as the most suitable one. The following section describes the interview methodology which we used will be introduced, including the reasons for choosing this method. Then the research plan used throughout the project will be explained. After that a detailed description will be given of how we collected the data that was used during the project. The next section will discuss shortly how the results of the data collection process were documented. Finally, our research methodologies will be evaluated.

6.1 ACTION RESEARCH

We have used Action Research as our research methodology. The purpose of Action Research is to influence or change some aspect of whatever the focus of the research is [ROBS02, p. 215-217]. A central element of the action research methodology is a focus on improvement through involvement with the studied organization. The aim of this type of research is to actively try to change and improve the current situation, while at the same time documenting and analyzing the effects of the changes. By reflecting on these effects, additional changes can be performed.

Given the above definition, Action Research is suitable for use in our project. We are attempting to change the software development processes of Team Sweden in order to increase their efficiency. Also we are actively involved with the team. Our aim is to try to improve the current situation while at the same time documenting and analyzing the effects of the introduced changes. Therefore, we have chosen to use this research methodology for our project.

Bassey in [ROBS02, p. 218] has written a detailed description of the different stages of the Action Research methodology. We have slightly modified this description in order to make it suitable for our project. This resulted in the following stepwise iterative research plan, which we have used as a guideline for conducting our project:

1. Define the inquiry. What is the issue of concern. What research questions do we ask? Who will be involved? Where and when will we do our research?
2. Describe the situation. What are we required to do here? What are we trying to do here?
3. Collect evaluative data and analyze it. What is happening now, as understood by the various participants? Using research methods, what can we find out about it?
4. Review the data and look for possible problem areas. What differences are there between what we would like to happen and what seems to happen currently?
5. Try to improve the current situation by introducing change. By reflecting critically and creatively on the identified problem areas, what change can we introduce which we think is likely to be beneficial?
6. Analyze evaluative data about the change. What is happening – as understood by the various participants - as a result of the changes introduced? Using research methods, what can we find out about it?
7. **Review the change and decide what to do next.** Was the change worth while? Are we going to continue it in the future? What are we going to do next? Is the change sufficient?

8. **Determine which additional improvements can be done based on the results of the first set and continue with step 5 until a satisfactory amount of improvement is reached.**

### 6.2 Interview Methodology

We chose to use the unstructured interview [ROBS02, p. 271] technique to obtain the necessary input from the team members. Doing an unstructured interview means that no specific questions are prepared before the interview is conducted. The interview is started with and initial question about the area of interest of the interviewer. From there on the interview develops in a similar fashion as an informal conversation would. The interviewee has the opportunity to discuss freely about the topics of interest. The reason for this choice is that in [ROBS02, p. 271] a number of situations are listed in which this interview technique is useful. Most of these situations apply to our project. We will now quote two of these situations, which convinced us to chose this technique for our project. Also, a short explanation will be given about why they apply to our project.

1. *Where a study focuses on the meaning of particular phenomena to the participants.*  
   Our study includes examining the opinion of the team members about the current way of working. Additionally we want to find out what the opinion of the team members is concerning the support measures we propose. Therefore in our case the 'particular phenomena' are the current way of working and the support proposals offered by us to the team. The participants are the members of Team Sweden.

2. *Where individual perceptions of processes within a social unit – such as a work group, department or whole organization – are to be studied prospectively, using a series of interviews.*  
   In our case the social unit is the Team Sweden organization. The processes are the software development processes of Team Sweden. Finally we are using a series of interviews to obtain the individual perceptions of the team members concerning the development processes and our support proposals.

### 6.3 Project Plan

In this section we will discuss our project plan. Before we could write our project plan, we had to determine exactly what kind of information we needed to collect for our project. To answer this question, we started out by discussing with our advisor what the aim of our project should be exactly. We combined the input from this meeting with information from theory on the subject. This resulted in the following step-wise project plan, which is based on our research methodology (see section 6.1):

1. Examine the current way of working.
2. Identify the major problem areas.
3. Look for solutions to these problems by doing research and if possible interviewing team members.
4. Write a support plan to help Team Sweden improve their software development process.
5. Write a stepwise implementation plan.
6. Focus on solutions that have a low impact on the current working situation while at the same time benefit the team as much as possible.
7. Implement the solutions accepted by the team.
8. Based on the results of the introduced changes, look for and implement additional improvements.
9. Reflect on the introduced changes.

The steps 1, 3 and 8 of this plan involve the need to collect data. Information about the current way of working had to be supplied by the team members. Additionally, our support proposals had to be accepted by the team in order to have a chance of being implemented and used. Therefore we wanted to know if the team members themselves had any improvement suggestions, which we could use to tailor our preliminary support proposal to the wishes of the team. Also we needed the approval of the team members on our preliminary support proposal in order to finalize it. In short; it was vital for the success of the project to get input from the team members. In the next section we will give a detailed description of how we collected the data needed for our project.

### 6.4 Data Collection

In order to find out the current way of working and what kind of problems that were associated with it from the point of view of the team, we chose to interview the node coordinators. The reason for this choice was that the node coordinators were the only team members with sufficient experience of the development processes of Team Sweden. The students currently participating in the team had just started their work with the team. They were therefore unfamiliar with working within the team and as a result also unaware of existing problems associated with this way of working.

We first booked a meeting with the node coordinator in Ronneby in order to interview him about his view on the current way of working of Team Sweden and the existing problem areas. This meeting was successful and supplied us with useful information. Because it was not possible at that moment to travel to Örebro and Lund, we chose to use e-mail to communicate with the node coordinators in these cities. We sent them a mail in which we shortly described our plan and asked what their view of the current working situation was. Additionally we asked if they could identify any problem areas and possible solutions. This turned out to be a big mistake. Even though we wrote the mail as politely as possible, we did not get the expected reply. The mail we got from Örebro implied more or less that they were perfectly happy with the current way of working and that they therefore did not have any time for us. In other words; the best thing we could do was to stop immediately with our project because is was totally useless. From Lund we never received a reply.

We have discussed about why this e-mail did not deliver the expected result. We reached the conclusion that the content of the mail was to demanding. Our mail could
be interpreted as follows: Describe the current way of working, identify the problems associated with it and provide solutions to these problems. If our mail was interpreted this way, this would imply that the addressees would think that they would have to spend a lot of time and effort in order to write a complete reply. This would make them reluctant to cooperate, which was obviously not our intention. Of course we needed to know about the current way of working and if they saw any possibilities for improvement. However, we reached the conclusion that email was not the correct medium to use for retrieving this information.

Around this time we received the information that the first team meeting of this year was going to be held within a month in Ronneby. We realized that this was our only opportunity talk to the node coordinators and other team members in person. We therefore asked if we could attend the meeting in order to interview the team members when the schedule allowed it. This request was granted. We gave a lot of thought to how we could present our project in such a way that the team was willing to cooperate with our project. We decided to interview the team members in an informal way in order to give them the opportunity to talk freely about their work for Team Sweden. A detailed description of the used interview technique is given in section 6.2. Whenever sufficient information on this topic was gathered, we changed the topic to another area of interest. At the end of the interview we asked the interviewee if he had any additional comments or suggestions in order to make sure that we had not overlooked or forgotten to discuss important topics.

We prepared ourselves for the meeting by writing a document describing an initial development support plan using the information we gathered from the interview we already conducted with the node coordinator from Ronneby. This document also included a short description of our project. Additionally we prepared a short oral presentation about our project. To obtain a more extensive knowledge base for writing the support proposal, we did an extensive literature research on software development support solutions and tools. We sent our proposal to all node coordinators before the meeting in the hope that they would be willing to read it before the meeting.

We conducted the interviews as planned during the first official meeting of Team Sweden. This meeting totally changed our opinion on the team. It turned out that most team members had read our document describing our project and the initial support proposal. After we had explained to them who we were and what we wanted to achieve with our project, they were willing to cooperate with us. They did not agree with all our proposals, but that was to be expected. A face to face discussion with the team members took away a lot of misunderstandings.

We used the input we gathered from the team during this meeting to finalize our support proposals. We then implemented the accepted proposals before the next meeting in Örebro a month later. That gave the team members the opportunity to look at and try out the implemented proposals. During the meeting in Örebro we asked the team members for their opinion on the implemented measures. We needed this feedback to fine-tune the introduced changes to the needs of the team. This meeting concluded the data collection phase of this project. Measurements of the usage and usability of the implemented measures fall outside the scope of this project.
Because of a lack of time we were not able to implement additional improvements based on the results of the introduced changes. Future development support projects for Team Sweden can address this issue.

**6.5 Documentation of Collected Data**

Our main data collection method was the unstructured interview. During the interviews we did not make any notes, because this would interfere too much with the discussion with the interviewee. In order to prevent losing any of the information we received during the interview, we wrote down the collected interview data immediately after the interview. This was done in the form of short notes about each topic of interest. After all interviews were completed, we compared and analyzed the noted of the different interviews. This resulted in a document comprising all information extracted from the interviews.

**6.6 Evaluation of Research Methods**

In this section the research methods - action research and unstructured interview - that we used in this project will be evaluated.

**6.6.1 Action Research**

We consider the choice of action research as the research method for this project as largely successful. The steps one to seven of our project plan (see section 6.3) were performed according to the plan. However, it would be better if we would have been more involved with the team. Also, it would have been beneficial if more time had been available for interviewing the team members. This would have given us a better insight in the way the team worked. Because we were not doing any development for the team, we had to rely completely on indirect information we got from the team members.

Additionally we had some difficulties getting sufficient information and feedback from the team members, because they had very little spare time. Because of the limited time we had to conduct our project in, we were not able to proceed with step eight: Look for and implement additional changes based on the results from the first set of changes. We have therefore reached the conclusion that, in order to really benefit from this method, it is necessary to use it over a much larger time span. This would enable multiple iterations of the change/improve cycle of the action research method, thereby maximizing the effect of the introduced improvements.

**6.6.2 The Unstructured Interview**

The choice of using unstructured research interviews turned out to be suitable for this project. During the meetings there was very little time available for us to conduct our interviews. Therefore, we had to rely on the breaks between the official parts of the meeting to find some time to speak to the team members. By using the unstructured interview technique, we did not have the need for a strict interview schedule. We could just walk around and talk to the different people in an informal
way about their view on our topics of interest whenever they had time. Because the interviewees had the opportunity to speak freely, we were able to get a good idea of the point of view and opinions of the different team members. Because of this flexibility this interview method was ideally suited for this situation.

The only difficulty of this method is that it is fairly complex to analyze the results. First of all, we did not have the time or opportunity to make notes during the interviews. We therefore had to be very attentive during the interview in order to prevent that we would fail to remember important information. Because we did not have an actual list of questions, we also had to deal with a lot of duplicate and even conflicting information. Therefore it took a fair amount of effort to analyze the interview results and extract the important information. However, this cannot be seen as a disadvantage of this method. Whenever duplicate information occurs, this indicates that the opinion or topic in question gains a higher importance and/or truth value. Contradicting information reveals that further research is necessary on that topic.
7 Problems and Solutions

This chapter describes a number of problem areas that we have identified in Team Sweden's current software development process. We have divided the problems into six different categories, namely: Configuration Management, Communication, Source Code Maintenance, Knowledge Management and Other Problems. We will also present our suggested solutions to the problems and, if applicable, our implementation of the support proposals. Any measurements and feedback that we have received is also included in this chapter.

7.1 Positive Aspects

Even though this thesis will focus a lot on the problems that Team Sweden has, it is also important to emphasize the positive issues. Any positive aspects of an organization should be maintained and reinforced so that they are not lost.

Despite a large turnover of developers, there are still a number of permanent members that have dedicated a few years of their research to the Team Sweden effort. They are highly skilled and with an extensive knowledge about robotics and AI. Their commitment to the team and their field of research has made it possible to maintain the team and continue its work. Team Sweden would not have survived without them. And even though the team relies on new master and postgraduate students every year, they always manage to attract a large number of developers every year. Even though the students usually quit Team Sweden after the tournament, they help increase the knowledge of the team with new ideas and solutions to the complex research problems that the team faces.

A major reason to the success of Team Sweden is their ability to maintain a high team spirit. The research aspect of the organization and the many different topics makes it easy for new and old members to find areas they find interesting and appealing to discover more about. This kind of spirit of discovery has made it possible to function as a team, despite a complex, distributed organization. The team's ability to adopt to the needs and interests of many different contributors has made it possible to maintain the team for so long.

7.2 Configuration Management

This section describes how the configuration management is handled within Team Sweden and what problems that are associated with it. It also includes our support proposals and implementation of the proposed changes. We define Configuration management according to the IEEE Standard Glossary of Software Engineering Terminology [IEEE90]; a "...discipline applying technical and administrative direction and surveillance to; identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status...".
7.2.1 **Problem Description**

Supplying the entire code base to the rest of the team has been a problem in the past. A common repository on a File Transfer Protocol (FTP) server is available but it is only updated before deadlines, which are not too frequent. Sometimes the deadlines fail because they are missed or even ignored. This has sometimes created problems for other developers because of the various dependencies of the code. It is often difficult to access the latest version of the source code. Tests and integration are sometimes a problem because of the poor availability of the latest code-base.

Communication between the different software modules is done using predefined interfaces for calling the different functions. Traditionally the interfaces change very little over the years. However, problems have arisen in the past when the behavior of commonly called functions are modified, affecting other modules. Team members sometimes make assumptions about the functionality of other parts of the software without discussing these with the other developers. An example of this was a change in the visual controller that affected the angle of the AIBO's head. The changed perspective affected the positioning algorithm and made it difficult to maneuver the field. This problem, and similar, are fairly easy and quick to fix but they can create problems if the changes are applied to close to the competition.

7.2.2 **Support Proposal**

A Concurrent Versions System (CVS) could be used to store, manage and integrate the code. The CVS repository can be made accessible through the Internet, thereby enabling the entire team to access the code at any given time and place. This also works on local networks, for example during the competition, provided the CVS server is installed on one of the machines. Regular updates to the repository would ensure the availability of complete and up-to-date code to every team member. A CVS would make testing and integration considerably easier by allowing developers to try their code at any given time. Whenever the software is stable enough to be used, a release could be made. This ensures that the most recent working version is always available.

7.2.3 **Implementation**

The CVS was implemented by setting up a CVS on the Team Sweden server located in Ronneby. This server was made accessible through the Internet. The second step was to set up a CVS repository to store the Team Sweden source code in. To ensure the correct functionality of the CVS several tests were performed. The procedure of setting up the CVS was fully documented in order to ensure the process was repeatable. We also set up a web-based maintenance tool to simplify the maintenance of the CVS server. Although there are free graphical CVS client application available, we decided not to make use of them. After having tested the usability of the different tools, we reached the conclusion that they would not be beneficial to the team. Some tools did not offer enough functionality, while other crashed frequently. We deemed it better to write a number of simple UNIX shell scripts to simplify the execution of the most used CVS client side commands.
7.2.4 Feedback and Opinions

The use of a CVS was seen by team as over extensive. Even though they recognized that they have problems with configuration management, they also did not believe that the use of a CVS could solve any of the complications they have had in the past.

Increasing the visibility of the source code would not solve the problems they have had with integration. There is no particular need to look at other modules since the predefined framework works fairly well. Additionally, the team did not feel that there was a particular need to change and modify code that they were not personally assigned to. They are content with only having one developer working on a specific file. The team expressed a reluctance on allowing other people modifying their code. A additional source of reluctance was some members' previous experiences with using a CVS. A common opinion was that it requires a lot of extra work, without any obvious benefits.

7.3 Communication

In order to support and improve the collaboration within Team Sweden, the communication also needs to be improved. This section describes how the communication within Team Sweden has been handled in the past and our support proposals for possible future changes. We will also discuss our implementation and the feedback we received on it.

7.3.1 Problem Description

Team Sweden consists of a lot of developers, distributed over many different locations. Communication within the team has been a problematic area in the past. The problem is a result of the distributed organization which makes the information flow difficult to maintain and support. The team relies almost exclusively on e-mail to communicate changes, news and decisions. Unfortunately, e-mails do not always succeed as a communication tools since they are sometimes ignored and not answered. Information is lost and the development is sometimes stalled as a result. An other problem is that communication is usually handled peer-to-peer between single developers which means that the information is not distributed to the rest of the team.

7.3.2 Support Proposal

The communication within the team could be supported through the use of dynamic mailing lists supporting message threads. Issues arising during development, including their solutions, could be discussed and stored on the appropriate list. By doing this, the whole team is kept up to date on what is going on in the team and, in addition, can provide valuable input for solving the problems. For security reasons, it should not be possible for people that are not members of Team Sweden to subscribe to the mailing lists. Subscribing new people to the list should be handled by the person maintaining the list. The mailing lists should have the option of being password protected, to prevent unauthorized persons from accessing it.
An example of a useful type of mailinglist is a Frequently Asked Questions (FAQ) list. Whenever an answer to a frequently asked question is posted, it can be sent to this list. Doing this will gather these questions in one place. This will result in a FAQ-list without the need to write separate tutorials. New team members can be asked to browse this list before asking the other team members for an answer. Only when the answer is not available on the FAQ list, the answer has to be provided by the senior team members and sent to the list. Therefore, by implementing this list, one can avoid the same questions being asked over and over again. This list should be accessible through a browser so that the posts are easy to find and read. Additionally, the list can be made searchable, thereby further increasing the usability.

Other types of mailinglists dedicated to a specific topic could include, for example, 'administration' and 'development'. This would structure the relevant information in different categories making it easier to find, sort and manage the messages.

7.3.3 IMPLEMENTATION

The team already had a mailing-list that is used for discussing team and developer specific issues. It is not frequently used but as the competition approaches, we expect to see a lot more activity on the list. The technical solution for the list is a simple UNIX alias-file, containing the e-mail addresses of the the entire team. When an e-mail is sent to the Team Sweden user, it is forwarded to all the addresses in the alias-file. This is a simple solution that doesn't require a lot of extra effort, besides that the team must contact the administrator when the list should be updated.

We looked at two different mailing-list programs, Majordomo [URL-MD] and Hypermail [URL-HM] but neither one of the offered enough useful features as to motivate installing, learning and maintaining an entire application. Instead, we decided to keep their original UNIX alias-file and only to install the mailing-list forum application. We looked around for a number of clients but the one that looked the most promising was a mail-to-HTML converter, called Mhonarc [URL-MH]. It reads a UNIX-style mailbox and converts it to the Hyper Text Markup Language (HTML), with support for threads and chronological sorting. It gave us the option of displaying the entire mail-discussion in format, readable by an ordinary browser on any computer, given that you can authenticate yourself. A simple cron-job makes sure that the list is updated every hour so that the newest mails are added to the web-archive.

7.3.4 FEEDBACK AND OPINIONS

The mailinglist in combination with the forum software was received very well. The fairly easy set-up was also appreciated since it requires little effort to maintain and use. Our solution was accepted by the team but they also recognized that it would be difficult to measure any real results in the near future. Whatever topics that are discussed on the mailinglist will be read be the team and there is no real use to look and search the forum until after long time when the information has been forgotten. They also believed that it would be particularly useful for the next
generation of developers that needs to have their questions answered. But although they liked our solution to their communication problem, they also recognized that it will not solve all their problems, but it is a good, first start.

7.4 **SOURCE CODE DOCUMENTATION AND TAXONOMY**

This section describes our suggestions of how the source code could be documented and classified within Team Sweden. It handles both our support proposal as well as the implementation of the proposed changes. We have defined *Taxonomy* according to the IEEE Standard Glossary of Software Engineering Terminology [IEEE90]; a “... scheme that partitions a body of knowledge and defines the relationships among the pieces. It is used for classifying and understanding the body of knowledge.” It also defines *Documentation* as “... any written or pictorial information describing, defining, specifying, reporting, or certifying activities, requirements, procedures, or results.”

7.4.1 **PROBLEM DESCRIPTION**

Due to time constraints and lack of interest, it is often difficult to make the team members document their source code. This is a big problem since the developers usually quit the team effort after the tournament and their code is left unmaintained. It is very extensive to update and add new functionality if the code is poorly documented. The lack of a common naming policy also makes the code difficult to understand since every developer has their own personal way of naming constants and variables which makes it difficult to distinguish between them. In the past, this has resulted in a lot of double work since the old code is difficult to maintain and have had to be rewritten. It also increases the time it takes for new developers to get acquainted with the source code. A lot of time could be saved if the team could agree on some common guidelines for documentation and taxonomy.

7.4.2 **SUPPORT PROPOSAL**

Students could be encouraged to document their code to a greater extent. This includes both high-level design and low level in-line comments. One way of doing this would be to require that the source-code of their project is included in their thesis paper. Having the code as an appendix would increase the accessibility and might convince people to further structure and document it.

Older, undocumented, code could also be documented if people using it would add comments. This would support future development, since not every developer working on old code would need to spend time and effort trying to understand it.

A further addition would be to introduce a code standard for the students working with Team Sweden. This standard could address issues like naming conventions, change logs, authors and contact information. The introduction of such a standard would make the code considerably easier to read, understand and to modify for other/new developers not familiar with the source-code.
We would like to note that we have refrained from defining any rules for sorting or maintaining the source code since this would be a major change to the current source code maintenance and outside the scope of this project. Our effort has been focused only on taxonomy and documentation in order to increase the readability and support future development. However, we also advise that future SEPG of Team Sweden should look further into this.

7.4.3 IMPLEMENTATION

We made a study of the current source code of Team Sweden in order to distinguish patterns and similarities between different modules with different developers. Our study was focused on existing terminology, consistent naming, well-known abbreviations and the relationships between classes. The rules and procedures that were most commonly used were gathered into a code standard handling mainly taxonomy and documentation. We also talked to developers of Team Sweden in order to get their opinion of what could be beneficial for the team. Their additional changes and modifications were also included into the code standard.

The code standard was intentionally kept short and with a small amount of rules and regulations. The reason for this was because the current code base of Team Sweden is huge and contains a large number of different undocumented standards and conventions. It would be impossible for us to produce a code standard that comprised all of them. It was also difficult for us to write a standard for Team Sweden since we had not done any development for them and we did not know what was needed and what would be superfluous. Instead of writing an extensive document, we decided to limit ourself to a smaller set of rules that would be easily incorporated by the team. We feared that if we made it to big, the team would ignore it and not use it. Another reason for keeping it short was because it would make the standard easier to maintain and update.

We expect the standard to grow over time to include more conventions that will benefit the team. In the beginning, it is easier to use a smaller standard and add rules as the need for them becomes apparent. We believe that this course of action will increase the chances of the standard being used. This is also the reason why we only included naming conventions and guidelines for documentation, since any other rules would be a major change to the current development processes. We also avoided any semantics such as indentation and bracket placement since these are purely ethical choices and not relevant to the understanding of the code.

The complete Team Sweden code standard (version 1.0) is available in Appendix A.

7.4.4 FEEDBACK AND OPINIONS

Unfortunately, we do not have any data on how frequently the code standard and developer guidelines are being used since we did not have sufficient time or means of performing such measurements or collecting the relevant metrics. However, the document was received well by the team leader and node coordinators and they
thought it was a good first version. The need for a common development standard has been apparent to them for quite some time but they have not had the time or energy to make one yet. But they believed that it would be a lot easier to add new conventions and standards now when they have an official document to add the rules to.

They had some minor grammatical changes to the code standard but otherwise it was approved without any major revisions. The changes were mainly minor details such as prefixes and postfixes on variables, constants and parameters. They also felt that the guidelines for documenting the code could be very beneficial as long as they were enforced. Any change that could increase the amount of documentation to the code base was a welcome change.

7.5 Knowledge Management

This section describes our suggestions of how the knowledge management processes and information transfer could be improved within Team Sweden. It handles both our support proposal as well as the implementation of the proposed changes. The concept of knowledge management handles issues such as supporting innovation, reusing information, fostering knowledge sharing, capturing experience and making it available [LEVE00, p. 258].

We would like to note that this section handles mainly knowledge transfer related to research and development, rather than organizational and management related issues. Our definition of knowledge management handles lasting knowledge (e.g. research articles and papers), rather than continuously changing information (e.g. travel arrangements and updates to the code). Communication and information that is subject to constant change are handled in section 7.3.

7.5.1 Problem Description

Knowledge is said to be an organization's greatest competitive advantage [DAVE98, p. 13]. In the competitive RoboCup tournament, where the current code base and technology is continuously evolving and building on top of older, previous versions, it is extremely important that the knowledge within the team is preserved, shared and updated. If Team Sweden want to remain in the tournament, they need to improve the current knowledge management. Although the benefits of a knowledge management system, such as quality and efficiency increase, is difficult to measure, the advantages of effective knowledge management within an organization are obvious [DING02].

Knowledge sharing between different sites is vital in a distributed organization. Given the large turnover of people and resources of Team Sweden, additional measures should be taken to preserve the knowledge within the organization. Currently, a lot of knowledge is lost every year when the old developers are exchanged with new ones. The work is usually time limited and few members remain in the team for longer than half a year. It would be beneficial to preserve the knowledge that the older generations of the team has learned and make it available to the following generations of developers. It is not possible to keep the developers
but it could be possible to preserve their knowledge through a knowledge management system. The current implementation of a knowledge database does contain some useful information but it is outdated, irrelevant and does not really fill any function. Even though the knowledge is available, it is not really transferred if it is not absorbed [DAVE98, p. 101].

### 7.5.2 Support Proposal

One solution to the problem could be a knowledge database containing information such as documents, FAQ's, articles and thesis work. Each node should be made responsible for updating their part of the content. By giving all the node coordinators permissions to add/remove information, it would be easier to keep it up-to-date. Such a dynamic database would ensure that the knowledge is stored and freely available for future generations of developers and participants of Team Sweden. This would save a lot of time and effort in the long run, particularly for new students joining the team effort.

The usefulness of the knowledge database is dependent on the amount of information that is stored in it. To be beneficial, it needs to contain a lot of documents and information on various topics. This will be a problem in the beginning, since the information needs to be gathered and uploaded. However, the information will accumulate over time, thereby increasing the usefulness of the database.

There are other measures that could be taken to increase the amount of knowledge sharing within the team. Possible solutions could include more team meetings, encouraging members to further document their work and to increase the transfer of information within the team. Unfortunately, these measures are all time- and resource intensive and therefore not possible to implement at the moment. Any effort that could increase the total knowledge of the team would be beneficial but should be weight against potential costs of implementing them.

### 7.5.3 Implementation

We created an extensive directory structure to be used for the knowledge database. This would be an big improvement over the current database on their homepage, which only contains a few research articles. The knowledge database was then moved to the private FTP server of the team, which requires an authentication to access. This also gives all the team members the ability to download and upload information and documents the same way as they have access to the source code. Since the entire team has the ability to add information, we hope that more information will be available in due time. This is an important improvement from the previous version which restricted the upload access to the one person responsible for managing the database. The authentication will also make it possible to add information that is restricted knowledge for anyone not with the team.
The structure in which the information was to be stored was important because, once it is implemented, it is difficult to do any major restructuring in the database. The information also needs to be easy to locate so that information is not hidden away and unusable. It must also be easy to update. Even more important was to make it possible for the database to contain a lot of information since it will, hopefully, grow over time. It should be as easy to browse and use with a few files as with a hundred files. We tried to avoid making the directory structure to deep since this would make it difficult to locate the information. Also, we tried to make the catalogs as unique as possible so that there would not be any doubts what should be put where.

In order to make the database dynamic, we had to avoid any files that describe the catalog structure and content since they would have to be updated. We limited ourselves to a few, short text files that were general enough not to require any updates.

7.5.4 Feedback and Opinions

A more extensive knowledge database was seen as being a good solution for storing and sharing the knowledge of the team. The only real objection was from one node coordinator that worried that it would not be updated and therefore become as outdated as the current one. But if the members of the team could be encouraged upload information and papers, this would not be an issue. The new and improved structure was otherwise seen as an improvement since it is easier to locate and to share information due to the increased permissions of the team members.

However, the success of the database is dependent on if the members are encouraged to upload research articles, papers and other kind of information. If the members do not commit to upload and share their information, the database is bound to be outdated and useless. Therefore, if the database was a success or not is not possible to determine at the moment.

7.6 Other Problems

The problem areas that we have mentioned earlier in this chapter are not the only ones. There are a number of different issues that the team is facing but which are too complex to be solved by any simple means. For the following problems to be dealt with, the team needs to do some major reorganizations that are not possible to perform in the short time frame of our research, if possible at all. However, we do have a few suggestions on how the problems could be handled.

7.6.1 Developer Turnover

Team Sweden suffers from a high developer turnover and very few members stay with the team more than a year. Since most developers are doing their thesis projects with the team, they quit the team when they are finished. Traditionally, only the node coordinators and a few postgraduate students remain in the team after the competitions. Unfortunately, this causes a large problem because a lot of the
knowledge is lost when the members move on. If more team members would stay with the team, then the need for documenting their code would decrease.

It would be beneficial to keep the members in the team for more than one season. Unfortunately, this would be difficult to accomplish since they do not have the sufficient fundings to employ developers. One solution would perhaps be to increase the number of postgraduate students within the team since they tend to stay longer than students doing their masters thesis. However, these students also need funding which might be problematic because of the limited resources available to the team.

7.6.2 Time Management

The team could benefit from better time management. A lot of the problems with communication and missed deadlines are a result of lack of time. It is difficult to increase the resources spent on the team but if the members would dedicate time in their schedules for Team Sweden activities then it would be easier to achieve a structured development process. For example, mails are often ignored and not answered properly because the members are occupied with other work, not related to the team. The e-mails that are answered are usually not as extensive as they should have been, which is a cause of misunderstandings and misinformation.

A possible solution to this problem would be for the members to dedicate one or two days a week to Team Sweden, depending on their current commitment. This would assure their complete attention to the team and ensure that it got the time it requires. If the members were allowed to focus their entire attention on the problems facing the team, it is quite possible that the solutions would appear faster. It would be beneficial if the members could devote more time to the Team Sweden project but this is not possible due to other equally important commitments, but it is likely that a better time management would benefit the team effort.

7.6.3 Complex Development Dependencies

This section refers mainly to the dependencies concerning the development aspect of Team Sweden. The code base of Team Sweden is very large and the feedback during execution is scarce and cryptic. The amount of information that can be discerned from the AIBO robots is limited to what actions and movements they perform, which are extremely dependent on the situation and other factors. It requires an extensive knowledge about the dog and the Team Sweden code base in order to make an accurate prediction and isolation of a particular behavior in the robots. And even the best predictions are uncertain at best. Fortunately, there are a number of debug printouts that can give some hint of what causes a particular behavior but they are also difficult to read and make any accurate predictions from. The large amount of different developers on distributed sites makes this an even larger problem.

The team would benefit from an improved feedback from execution runs and debug printouts. Any tool or technique that could help the developers to isolate issues and runtime errors would be a huge improvement. The traceability of the current
implementation is difficult and sometimes non-existent - a lot could be done to support the developers. This problem is difficult to resolve without being a developer in the team and therefore we will not offer an actual solution to the problem. However, they have assigned this task to a developer and he will hopefully be able to increase the traceability of the code.
8 DISCUSSIONS

This chapter will discuss our experiences with Team Sweden and the conclusions we have drawn from our work on this thesis. This chapter might be useful for people performing similar work related to both groupware and Software Process Improvement.

8.1 EMBRACING CHANGE

Both our research on the topic and our experiences with Team Sweden testifies on an organization's reluctance to embrace change. Humphrey talks about the human nature of accepting and embracing change but also an almost universal resistance to it [HUMP97, p. 261]. Change and improvement is commonly considered a necessity, but paradoxically, there always seems to be a reluctance on adopting it. For example; Team Sweden demonstrated a strong resistance against changing their configuration management using a CVS. They had convincing arguments against using it but also admitted having doubts solely based on a reluctance of trying out a new, unknown method. However, they did agree to look at it but made it fairly clear that they would not use it, even before trying it. It seems as though an organization and it's participants are more inclined to reject changes than to embrace them.

A reluctance to change can sometimes be derived from the inherent uncertainties that changes unavoidably bring. Older and reliable methods are less intimidating than any new and unfamiliar methods. Any changes will convey risks that in some way or another might disrupt normal work and development [HUMP97, p. 262]. This is certainly a valid assumption but not a reason to resist change. Rather than to be discouraged from making changes, this should instead influence the way they are presented and implemented.

Another potential reason why organizations and individuals might be reluctant to adopt change is the misconception that any improvement proposals are considered as criticism. Any suggestion that internal or external individuals might propose to an organization might be perceived as critic. By questioning their methods and the way they work, you are questioning the individuals and this could be taken personally. These issues and other should be taken into consideration when conducting an SPI.

8.2 INTRODUCING AND CONDUCTING AN IMPROVEMENT PROJECT

Humphrey suggests that resistance to change should be treated as a symptom rather than as a problem [HUMP97, p. 262]. Reluctance to embrace change is natural and should be considered when conducting an SPI. But this reluctance can cause problems when introducing and conducting an improvement project for an organization. We have attempted to draw a few conclusions and general guidelines on the topic based on our experiences with Team Sweden.

Changes to an organization and its processes must be carefully motivated and justified. Any changes that are performed should be improvements rather than impairments and if this can not be proved and substantiated, then the changes should not be implemented. This was apparent to us when we tried to get our proposals
approved by the node coordinators. The CVS was difficult to motivate to the team because the advantages were not as obvious as the other proposals. It would have been a relatively large change to the development processes but without sufficient advantages to motivate the change.

In order to be successful with an SPI, there must be a high degree of participation and involvement from the organization. Changes that are not supported by the members of an organization will surely fail [HUMP90, p. 31]. Participatory planning will ensure that the changes are motivated and viable. This will also ensure that the correct decisions are made [HUMP97, p. 261]. An SEPG is not always familiar with the development processes which makes it difficult to make the correct assessment and appraisal [ZAHR98, p. 71]. For example; our project would have failed without extensive help from the team leader, the node coordinators and the students. It is also likely that our results would have been better if we had had further feedback and participation for the team.

Members of an organization might be inclined to reject proposals that originate from external sources. The reason is that any valid and valuable improvement suggestion requires knowledge about an organization and its processes, but this knowledge is difficult to achieve without being a part of the organization. However, this is not always a valid assumption since the proposals can still result in improvements. Any external SEPG will have to prove its knowledge and expertise before their suggestions are considered. For example; Team Sweden demonstrated a obvious reluctance at the beginning of our project. This changed over time to a more open attitude, something we attribute to a number of different reasons. One reason might have been increased support from our advisor, something that increased our credibility with the other involved team members. We also met the participants which made future cooperation considerably easier since it increased our understanding of their processes.

The results of an improvement effort should be made visible to the participants. In order to keep motivation up, the results and progress should be made openly available [ZAHR98, p. 194]. This will also make people aware about what is being done and increase the chances of useful feedback. Unfortunately, the lack of time and resources within Team Sweden made this difficult to accomplish. The distributed structure of the team limited the feedback and updates to what we were able to communicate through e-mail and the few meetings they had.

The first contact with the organization is very important. It will determine the success or failure of an improvement effort. This became apparent for us when we made the initial contact. The mail we sent at the beginning of the project was a big mistake (see section 6.4). The first meeting was scheduled relatively late which forced us to make initial contact before we could meet them in person. This resulted in the mail that was partly ignored and partly criticized. We would have had a better success if we had acquainted the node coordinators with the concept and potential benefits before asking them to contribute. The need for the project should have been better substantiated before we demanded that the team should spend time and resources on it.
It is futile to get written feedback through abstract, extensive and open questions [ROBS02, p. 274-275]. Questions like “what can be improved?” are difficult to answer and will require too much time and effort to reply. Usually, the subject of an improvement effort does not have the sufficient time to spend on questionings. If you intend to get relevant feedback then the questions need to be focused on the issues and easy to answer. Questions should result in concrete and easy to compile answers, while maintaining the content so that the result is still usable. This became apparent to us when we tried to get feedback at the beginning of the project. We would have received better replies if we had structured and divided the inquiries in a better fashion.

Any resistance to change can be avoided if the changes are divided into smaller increments, making each change less intimidating [HUMP97, p. 262]. Smaller, continuous improvements will also make the results easier and more accurate to measure and reflect upon. This was difficult to do in our project because of time limitations, and we feel that there are many more changes and improvements that could be performed within the team.

8.3 Distribution of Effort

The Team Sweden homepage is rarely updated and the knowledge database is outdated and only contained a few articles. A contributing reason for this might be that only one person has access to them. We believe that the time between updates would decrease if this responsibility would be shared between more team members. If the team would distribute the effort of maintaining the homepage and databases, the updates and amount of information would most likely increase. The more members that have access to the server, the higher the chances of maintenance and regular updates.

This is the reason why we suggested that the entire team should have access to upload information to the knowledge database. If everyone shared the responsibility of maintaining the information that is stored, then the number of articles and papers in the knowledge database would be likely to grow faster. This should work for the database and FTP but might cause problems for the homepage. The content on the homepage is more crucial, partly because it has a user interface and partly because it is publicly available. It is perhaps better to limit the access on the homepage to a few selected members in order to avoid any disputes on its appearance and content. If all of the node coordinators were to share the responsibility of updating the homepage, then it is likely that it would be updated more frequently and they would also avoid any unnecessary conflicts. Distributing the effort should be beneficial for most assignments related to maintenance and other continuous, repeatable tasks.

8.4 Increasing Productivity

A requirement for our process improvement project was that the changes we implemented would not interfere with the teams work. Team Sweden is limited in time and resources, so whatever effort they spent on our project would decrease their productivity rather than increase it. Whatever changes we propose would have to be implemented by us and not create any extra work for the team. If we were to introduce
a groupware or a new technique, it would have to result in an almost immediate increase in productivity, otherwise it would not be worth the effort.

Any changes performed in an organization will result in a temporary decrease of productivity and efficiency [DEMA97, p. 161]. The reasons for this vary but can be time spent on adopting the change, training in the new practices and establishing the new processes. It is likely that the larger the change, the more productivity will decrease in order to adapt to the changed processes. This is an issue to consider when conducting an SPI.

Since the rules of the RoboCup tournament require the participants to fulfill a number of challenges in order to qualify for the next year, it is never certain that Team Sweden will qualify for the following tournament. This means that any improvements done during our project will have to result in a rapid productivity increase in order to be useful.

Performing a continuous process improvement seemed like the best approach for our project (see section 6.4). It was never an option to carry out any major restructuring of the organization, nor was it possible to convince them to implement any changes that required to much time and effort. By splitting up the proposals into smaller increments, we were hoping to increase the productivity with as little negative impact on the current way of working as possible. We wanted to minimize the time it took for the team to institutionalize our changes. This meant that we had to limit our changes to the ones that only required a minor effort to learn and utilize. The longer it takes to introduce a new tool, the longer before the investment will be returned.

The lack of patience for new changes was a contributing reason to why the CVS failed. The team would not take the time it required to learn how to use the CVS and we did not have enough time to create sufficient confidence in the technique. It was also difficult to convince the team to adopt proposals that forced them to change. Any tool or technique that required the team to change their old processes was not well received, but the groupware that gave the team an option of using it was acceptable. For example, if the CVS was to be used, the entire team would have to adopt to a new way of working. The mailinglist archive, on the other hand, offered the developers the option of using the pages, or simply ignoring them. If a SEPG wishes to introduce groupware, make sure that the people who are subject to the change have an option of using the tool or not. The resistance to the change will increase if they perceive the change as something that is forced upon them. Unfortunately, if the members are not forced to use the new groupware, it is uncertain that it will be used at all and the effort spent on introducing them will be lost.

8.5 Groupware and Team Sweden

In this chapter a description will be given of how the groupware problem areas described in the groupware theory section (see section 4.2) relate to our work with Team Sweden.
8.5.1 Disparity in Work and Benefit

This problem area will be related to our work with Team Sweden by discussing the introduction of a CVS in the team and why this failed. When a CVS is introduced, developers will have to get used to uploading their code to the repository. In order to keep the repository up to date, this has to be done fairly frequently. This implies that developers need to spend extra time and effort in order to supply the CVS with up to date code. Developers, which are working on code sections being modified simultaneously by other developers, are more likely to be willing to spend this extra effort than those not in that situation. They will appreciate the automatic integration and version handling offered by a CVS. Developers working on code that is not modified by others will most likely not see the benefits of a CVS. They do not have the need for automated integration of different code sections. Therefore, for them the introduction of a CVS will result in extra work without any benefit.

Another issue is the learning curve the group members have to go through in order to get up to speed with using the CVS. If the majority of the team have prior experience with using a CVS, this curve will be short. On the other hand, if the team is inexperienced, the learning curve will be much longer. Since, in a company, time equals money this is a big hurdle which can result in the rejection of a CVS.

Finally, the CVS server has to be maintained. For example; old versions of the software have to be removed from the repository. Also, when a faulty piece of code is committed to the repository, this change has to be undone. This can sometimes be complicated. The person assigned to this task has to spend time maintaining and monitoring the CVS repository. Therefore he or she will be reluctant towards the introduction of a CVS.

The opinion of the team was that the extra work required by the introduction of the CVS did not match the benefits of the CVS. In other words; the disparity between work and benefit was too big. They did not want to spend time on maintaining the CVS. Additionally the main benefit of the CVS, namely the automatic version handling and code integration, was seen as unnecessary. There was no need to integrate different parts of modules, because there is only one developer working on a specific module at a time. In short; the powerful features of a CVS are not needed by a small scale team like Team Sweden. This type of groupware is more suitable for larger and more complex software development organizations.

8.5.2 Critical Mass

This critical mass problem will be related to Team Sweden by discussing the introduction of the knowledge database. Since the team is distributed over four nodes, the main advantage of the knowledge base is that it can function as a centralized storage space for archiving knowledge. However, it is essential that each node provides important information about the projects being performed at that location. In other words; at each node key team members have to be willing to supply that information. These key team members form the critical mass of users needed to ensure the success of the knowledge database.
8.5.3 Local Optimization

The local optimization (see section 4.2) problem will now be considered. This type of problem also applies to a knowledge base. In order to keep the knowledge base up to date, at least one person has to be willing to update the database. In the case of Team Sweden, each node has to supply a person responsible for updating and maintaining a part of the stored knowledge. This is not in the best interest of the persons responsible for doing this. Therefore, if they choose to pursue their own best interest, the local optimization problem occurs which can result in failure of the knowledge database.

8.5.4 Disruption of Social Processes and Current Way of Working

The introduction of the mailinglist archive will be used to relate this problem area to our work with Team Sweden. As described in section 4.2, the introduction of groupware may lead to disruption of the existing social processes within the organization or group. In this case the introduced groupware is the mailinglist in combination with the HTML mail archive. In order for the mailinglist to be effective, the team members have to get used to sending all their Team Sweden related mail to the mailinglist. However, there is a possibility that some team members will be reluctant to do so, because for some reason they do not want the whole team to read their mails. This is an example of the disruption of the social processes within the team. Another reason for not using the mailinglist could be that the team members have difficulties getting used to sending their mails to the list instead of to a specific person. If the above situations occur within the team, this would severely diminish the effect of the mail archive, because its main benefit is that the whole team is kept up to date about what is going on in the team. In other words; this type of groupware is only beneficial when it is used by every group member of Team Sweden.

Fortunately, this problem did not arise during the introduction of this measure. All team members shared the opinion that this measure would be beneficial to the team. Of course, only time can tell if the mail archive is going to be a success. It could be that some people will not make use of the list anyway. In order to obtain a definite answer to this question, extensive usage measurements have to be done.

8.6 Metrics and Measurements

Metrics are vital when performing a SPI endeavor [MATH01, p. 287]. Without any metrics, it is difficult to know how and where the organization needs to improve. Measurements are needed to support any decisions on whether or not the organization would benefit from a changed process. It is also difficult to determine the success or failure of any improvement cycle without any metrics to compare the new method with the old one. It is important to, as soon as possible, initiate a data collection and analysis process. This has an even higher priority than having a theoretical foundation for determining what data needs to be collected and how. The effort of sorting the metrics can be postponed to the point where a working data collection process has been institutionalized [MATH01, p. 290].
Historically, Team Sweden has never worked with collecting and measuring data. This is a distinguishable feature of their current maturity level. In order to enforce a structured development process in the organization, they need to institutionalize the process of collecting and analyzing metrics. They have currently no data on update frequency, communications, percentage of source code turnover or failure statistics. These, and other data, would be useful for determining what areas the organization needs additional improvement on. The quote from Humphrey applies; “If you don't know where you are, a map won't help” [HUMP90, p. vii].

Unfortunately, we did not have time to collect any metrics on our improvement proposals and implementations. Even though measurements on our efforts would be helpful for future SPI's, we did not have time to collect any data on the groupware or the current development processes. The time frame of the project did not permit it. However, all of our support measures will permit future data collection and analysis. For example: The update frequency of the knowledge database can be collected by looking at the dates when the files are added. The code standard (see Appendix A) can be used to approximate how well the code base conforms to the standard. It would be fairly easy for future SEPG's to gather data on our work and to determine their success or failure. These metrics will only cover a relatively small aspect of the Team Sweden organization but they might inspire to perform additional support measures.

8.7 Applicability to Other Areas

The use of groupware support solutions is not limited to distributed academic organizations such as Team Sweden. Any software development organization making use of teams to perform projects can benefit from these solutions. Inherently, the groupware problem areas and their solutions also apply to other software development organizations. For example; software companies with the need to maintain large amount of non-volatile knowledge will benefit from an efficient knowledge sharing system. Any software development organization would benefit from improved communication and increased knowledge sharing, because of the complexity of the software process. Groupware could be useful for achieving this.

The need to introduce change is not confined to the distributed academic software development domain. Humphrey [HUMP97, p. 261] talks about the human nature of accepting the need to embrace change while at the same time having an almost universal resistance against it. This symptom is applicable to all types of software companies, because it is a human nature problem instead of a specific business related issue. Therefore the problems and solutions discussed in this report are also applicable to organizations outside the distributed academic software development domain.

Finally, distribution of effort is also applicable to non academic software organizations. Whenever repeatable tasks - such as maintaining a mailinglist archive - are assigned to one person this can cause problems because of the extra workload of this person. Therefore he or she will be reluctant to do the job. By distributing the maintenance over several different people, the individual workload will decrease and also be fairly divided. This will diminish the reluctance against the task and improve the quality and frequency of the updates.
9 FUTURE WORK

Our work with Team Sweden is finished but a lot of effort remains in order to introduce a more structured development process. Most of our work was focused on establishing a process infrastructure using groupware and to introduce process thinking into the organization. This chapter discusses additional ideas and concepts that could be useful for future SEPG’s working with Team Sweden.

9.1 CONTINUOUS PROCESS IMPROVEMENT

Future SEPG’s would have to evaluate our results and to make an analysis on how the changes were received and if they worked. We did neither have the time nor the possibility to do this but it could offer useful information for future improvements. The biggest benefit with our project was that the tools and techniques that we introduced support measurements. Most of what we implemented can be evaluated using metrics gathered from logs, source code analysis and communications. These metrics could be used to support future changes to the Team Sweden organization.

We think that the team should attract other masters or postgraduate students to perform future process improvements for the organization. It would require very little time and resources from the team and the changes could result in increased productivity and quality. Team Sweden and its organizational structure is an interesting environment to perform research on and we believe it would be beneficial for future SEPG’s as well as for the team. An extra effort should be spent on improving the process culture and discipline of the organization.

9.2 OPEN SOURCE ORGANIZATIONS

The term Open Source Software (OSS) describes all software that is developed and released under a license that allows inspection and reuse of the product’s source code [CROW02]. Open source organizations have proved to be successful over the past few years and delivered products like Linux [URL-LI] and The Gimp [URL-GI]. Some characteristics of OSS are a high user and developer input, availability of the code and extensive use of different groupware tools. Even though the communities only exists on-line and the developers are distributed over many different locations, they are still able to produce high quality and competitive products. The work performed by Team Sweden is not suitable to be released as open source - the required hardware is expensive and the work is confidential - but the team might benefit from applying some tools and techniques that are commonly used by open source organizations. We recommend that a further study should be performed to analyze the methodology used by OSS organizations to determine if Team Sweden could benefit from them.

9.3 PERSONAL SOFTWARE PROCESS

A possible future improvement that could be introduced into the Team Sweden organization would be to adopt the Personal Software Process (PSP). The practice is based on the CMM and aims to guide the individual developers to a disciplined and structured approach to software development. Further information about PSP can be
found at the SEI homepage [URL-SEI]. A lot of research and measurements have been performed on the process model, indicating a large increase in productivity using the PSP [FERG97]. The entire model is more suitable for a commercial setting, unlike Team Sweden, but it could still be beneficial in terms of increased quality in the product and higher efficiency.
10 CONCLUSION

Changing the infrastructure and existing processes of an organization is not an easy task and requires a lot of time, effort and support. Process improvement is a continuous task and each change should be carefully analyzed and measured. It is not possible to change the entire organizational structure and processes through a single improvement cycle. Change should be done in small, manageable increments so that the changes are less intrusive and can be properly institutionalized.

It is always important to be aware of the potential problem areas related to SPI and groupware. It is easier to make informed decisions when introducing these techniques if you are familiar with the problems that can occur. Metrics and measurements can also be helpful for assessing failure or success of introduced changes. These metrics are useful for determining the correct course of action for future improvements. Performing an extensive analysis of the situation is a good way to lower costs and increase efficiency.

Simply introducing some new groupware tool or technique is not enough to improve the processes of an organization. Changing the organizational structure and practices requires more since process improvement involves all aspects of an organization, both on a managerial and a technical level. However, process improvement requires an infrastructure and groupware could be a useful tool for establishing it. Improved communication and knowledge sharing through groupware can be vital for establishing a process infrastructure. Groupware should be used in conjunction with other process improvement measures in order to be effective and improve the capability of the targeted organization.
BIBLIOGRAPHY


ONLINE REFERENCES


Appendix A – Team Sweden Code Standard and Developer Guidelines

Team Sweden
Code Standard
and
Developer Guidelines

Version 1.0

This document describes the Team Sweden code standard and developer guidelines. This standard will ensure that the files have a uniform appearance that allows for easy navigation and understanding of the source code. The guidelines will also help new developers to understand the old code and decrease the amount of time required to get acquainted with the system. All new source files added to the Team Sweden project, as well as changes to existing source files, should apply these standards.

Last updated 04/14/2003 (Chapter 6. Version History)

Table of Contents
1. File Introduction
2. Commenting Policy
3. Old Code
4. Naming Policy
4.1 Constants and defines
4.2 Variables
4.3 Global Variables
4.4 Functions
5. Initializations
5.1 Constants
5.2 Variables
5.3 Pointers
6. Version History
1. File Introduction

Each file should start with the following lines of documentation. The text written in uppercase should be filled in.

//====================================================-*-Mode: C++-*-==
// FILE_NAME.EXTENSION (i.e. Pam.cc)
// MODULE_NAME (i.e. Perceptual Anchoring Module)
// DESCRIPTION........................................................
// ....................................................................
// .......................(i.e. Task-based active vision and multiple.)
//
// Copyright (C) START_YEAR – CURRENT_YEAR
// NODE_LOCATION and Team Sweden, Sweden
// All Rights Reserved.
//====================================================

//====================================================
// Change log (for major changes)
//
// yyymmdd ACRONYM: Did something
// yyymmdd ACRONYM: Did something else
// yyymmdd ACRONYM: Finished something important
//
//====================================================

//====================================================
// To Do list
//
// - List something that should be done.
// - Another important thing you should do in the future
//
//====================================================

Please make sure that the acronym is generally recognizable and that developers can come in contact with you if necessary.

2. Commenting Policy

We recommend that students doing their master thesis for Team Sweden attach their source code in an appendix to their thesis. This should improve the standard of the code and increase the documentation.

The code should be documented as far as possible. New developers joining the team effort should be able to understand the code without relying solely on the source code. Please write the documentation as soon as a file, class or function is completed. The documentation should describe the purpose and the functionality of the code in such a way that an understanding of the code can be achieved without too much effort.

Due to time constraints before deadlines, it is understandable that documenting the code gets deprioritized. Therefore it is very important that the documentation is done as soon as the code is written. It is, usually, a lot easier to update existing comments than to write new ones. If you make any updates to the code, make sure that the documentation is accurate. After the deadlines and competitions, when there is more time, please spend some time documenting your code. It will be greatly appreciated by future developer looking at your code.

Files should be documented at the top, before the source code (please refer to chapter 1). Functions should be documented thoroughly, describing their purpose and functionality. Please remember to comment parameters and return values also. Variables and constants should be documented when initialized. Algorithms should include comments in order to increase the understanding of what they are doing.
Each file has a developer attached to it, but if someone else makes any changes, make sure to note this in the code. This is usually done with a short in-line comment (//-- ) consisting of an acronym, a date and the changes that were made. Please note that this only applies for developers that are not directly responsible for that particular file.

Example:
//--KL 030410 Changed this value to a better one.
int changed = 34;

AND MOST IMPORTANTLY - ALL COMMENTS SHOULD BE WRITTEN IN ENGLISH! THIS ALSO MEANS THAT DEVELOPERS SHOULD AVOID "NASTY" CHARACTERS SUCH AS É, Å, Ä AND Ö. NO EXCEPTIONS GRANTED!

3. Old Code
When old, unmaintained files are used, please make an effort to document the source code. A lot of code is not documented and therefore difficult to understand. If you are going to use old files, add comments whenever you feel that you have understood it. Please follow the same standard and guidelines described here.

4. Naming Policy
Names of classes, functions variables and so on should be logical and it's purpose should be made clear. Always use descriptive names in English. Avoid abbreviations unless you are certain that it is common knowledge for the members of Team Sweden. For example; names like "othcopf" does not comply with the naming policy. However, "ObjectToHandleCalculationsOnPotentialField" is accepted and a lot easier to understand.

If you absolutely have to use abbreviations or short variables, please make sure that they are properly documented when initialized. Also make sure that the name gives at least some clue to it's purpose.

4.1 Constants and defines
Constants and defines should have a short prefix based on what module it is created in, followed by an underscore '_'. The name should also be in capitalized letters.

Example:
int EFA_BOOSTINGFACTOR = 2;
int PAM_VERYIMPORTANTVARIABLE = 9315;

4.2 Local variables and parameters
The local variable- and parameter names should be in lowercase. For longer names, it is recommended to use underscores in order to make it easier to read.

Example:
float anothervariable = 2;
int a_good_variable_name = 1; // Also possible.

4.3 Global Variables
Global variables should end with an underscore '_'. Otherwise they follow the same rules as ordinary variables. Please note that global variables should only exist within a module, not the entire system.

Example:
int aglobalvariable_ = 1234;
float noticetheunderscore_ = 4321.0;
4.4 Functions
The name should, as far as possible, describe what the function does. This will make the code easier to understand and decrease the amount of documentation. The first letter of every word should be capitalized. This is very important since that makes it possible to separated a function from a variable.

Example:
`int SetCorrectNamingPolicy ();
float YetAnotherFunction ();`

5. Initializations
Please initialize each variable and constant on a new line in order to increase readability. No variables should be initialized with a function.

5.1 Constants
Please initialize all constants at the same place in the beginning of the file. Otherwise, the same rules apply.

Example:
`int PAM_INCREASEFACTOR = 2;`

5.2 Variables
All variables shall be initialized with a default value, usually 0 (or 0,0).

Example:
`int nice_name_for_variable = 0; // Correct!!!
potential = GetPotential(); // Wrong. Avoid function-initialization`

5.3 Pointers
Pointers should always be initialized to NULL (or 0) and assigned a reference later on. Again, please remember to use a new line for each initialization.

6. Version History
2002
Created by Ted Samuelsson

March to April 2003
Rewritten and modified by Gustav Nordenskjold and Harm-Andries Kingma

04/14/2003
Version 1.0 - first official release.