Lightweight Management - taming the RoboCup development process

Tijn van der Zant and Paul G. Plöger

FhG Institute of Autonomous Intelligent Systems,
Schloss Birlinghoven,
53754 St. Augustin, Germany
tijn@aisland.org
paul-gerhard.ploeger@ais.fraunhofer.de

Abstract. RoboCup projects can face a lack of progress and continuity. The teams change continuously and knowledge gets lost. The approach used in previous years is no longer valid due to rule changes and specialists leaving the team leave black boxes that no-one understands. This article presents the application of a recent software development technique called eXtreme Programming to the realm of RoboCup. Many common problems typical for teams of students seem to be solvable with this technique. It also gradually spreads out in professional software production companies. Students mastering it are of high use for their further career after having left the university. The strategy is being tested on a real RoboCup Mid-Size and an Aibo league project and produces very promising results. The approach makes it possible to modularize scientific knowledge into software that can be re-used. Both the scientist/expert, who has the knowledge, and the software development team benefit from this approach without much overhead on the project.

1 Introduction

1.1 RoboCup

RoboCup is an extremely difficult to approach problem and the complexity makes it impossible for single individual researchers to tackle it. It seems mandatory to attempt a solution in a team. In many teams members change frequently and the interest in RoboCup may vary substantially. Devotion of all work power to a single well defined topic exclusively is a rare exception. These discrepancies pose some of the following questions which are quite typical for many RoboCup teams:

– How can one avoid starting anew each tournament, i.e. how to support systematically the enhancement of a existing behavior system? Every new team member needs to get acquainted to the source code. The documentation is hard to maintain and can lack the clarity needed to start programming.

– How can one address the problems caused by permanent changes of the rules? Active and regular participation in tournaments requires a never-ending and constant improvement. This is very much akin to try to meet the demands of an external customer, who is not only never satisfied, but constantly changes the requirements.
– How can one cope with frequent team turn-overs, i.e how to avoid a significant drop in performance or the resign of the team if some key system designer graduates, leaving the team behind. How to either fill or avoid a knowledge gap?
– How can one foster cooperation in the team, i.e. how to lower the danger of a complete dependence on the expertise of a single member?

1.2 Correlations with the industry

If we compare this to the software producing industry these questions sound familiar and they can easily be paraphrased. The first might read as: how to distribute knowledge about huge bodies of existing code to all members of project teams in order to raise the overall productivity? Secondly, changing of rules can be interpreted as a permanent change of customer specifications resulting in an endless number of engineering change requests. Inherent to this lingers the demand for "faster, better and cheaper", which is known as well from other robotics customers like NASA[11] as from other big SW producing companies. Thirdly the life cycle of the project usually lasts longer than the individual career of a single project engineer. The goal of the whole project will be endangered if the team of programmers lack cooperation. Solving a large problem in common constitutes also a problem amongst *humans*.

1.3 Experience matters

Both authors have a long experience of more than six years in different RoboCup teams. They have extracted the problems described in this article from experience and conversations with many other teams. The authors thank the openness of members of other teams. It made them realize that many teams face very similar problems. No teams are mentioned specifically because the nature of the problems is a general one. It suggests to investigate all viable solutions and especially to look for successful techniques from software engineering. It is important to write down these experiences for the next generations.

This article is structured as follows: in the next section a problem description is given. It suggests a mapping to some solution. Section 3 constitutes the core of this paper, it defines the favored SW engineering method and reasons why it is very well suited for this problem domain. It also indicates some open problems that are partially addressed but need a more complete solution in near future. Then follows the results on actually applying the suggested methods in running RoboCup projects and closes with some outlook on future extensions.

2 Problem Description

2.1 The development process

In late 2003 it was decided to take a move from the RoboCup team formerly known as GMD-Musashi, being rooted at Fraunhofer research institute AIS[6], to become an internship project only operated by students in a masters program at the University of Applied Sciences of Bonn-Rhein-Sieg. During this migration process we faced a number of problems.

Facing the blend of some very elaborate and difficult to program problems students easily get overwhelmed. To keep them motivated one can use early successes in spite of the complexity of the underlying task. Traditional SW production models, such as the waterfall or spiral model, have a too slow turnaround for this, so we chose eXtreme Programming [7] (XP) as an underlying programming paradigm. It allows a jump start to code production and has a high promise of early success. Secondly we introduced a visual programming suite called 'Iconnect'[10]. It contains many different kinds of either standard or user defined modules dedicated to signal processing tasks in real-time systems. The programming paradigm is a synchronous data-flow architecture which eases the programming tasks for a robot very much. Although the first steps are simple, the library reaches all the way up to vision algorithms like scaling, clipping skeleton building, smoothing kernel filters and Lens calibration.

To foster collaboration, trust in the whole code has to be generated. This is established through extended automated testing, so called test case driven design or unit testing [7]. Every student is allowed or even encouraged to change the code of other group members, but to generate the necessary trust to dare so the changed code has to be accompanied by many test cases. The positive completion of all tests ensures that the functionality of the code remained invariant. Collective code ownership is essential and tightly bounded teams becomes possible. These procedures allow for micro architectural code transformation also known as refactoring [9].

2.2 RoboCup

The use of standard platforms, such as the VolksBot[1] and the Aibo[2], eases the development process in RoboCup. There can be a substantial waste of time in the maintenance of shaky robots being constructed from far too many parts, which can be tamed by using these basic robots. This change yields an encapsulation of all micro-controller related issues, so the SW team can concentrate on other issues, sometimes called high level SW. The knowledge of an expert is in the micro-controller, ready to be used.

A good choice is to drastically cut down on the vast many number of choices in the design space. Sacrificing here pays off in a much higher productivity, see section 4. By prescribing a VolksBot as HW and Iconnect as the low level SW exchange of modules becomes possible. Prescribing the SW development methodology of XP adds many advantages which RoboCup seems to demand.

It supports small teams optimally with a range of techniques, without giving too much overhead. One integral part of XP, the overall testing of the complete behavior system, is difficult to automate completely for the given case of behavior based robot control programs. But generally it is believed that regular unit testing leaves less than 20% of the whole SW system uncovered (our experience).

The expectations of the management may be met by a fast proof of capabilities achieved by a purchase of some robot, instead of the time-consuming process of building one. The start-up of new teams can focus on its organization and actively building up its structure. The growth and decrease of a team is no problem. Lazy students may get motivated by doing XP since it is fun to do and early running tests are tempting. Finally pair programming makes all people in the team cooperate and distributes the knowledge. This reduces what is known as the 'truck' factor in XP [7], which is the number of people that can be run over by a truck without endangering the project. If this is 1, for any part of the project, the project might be in trouble.

3 Approach

3.1 Top-down approach to bottom-up robotics

Some of the methods of XP have to be adapted to the RoboCup environment and some do not really seem to work at all. The authors have been on a few (non-robotic) XP projects and there the technique works quite well, although it is only a starting point and not the holy grail which solves all the problems. A list of practices used or aimed for is given with some explanations why we do it and whether it works as expected or not.

Borrowed techniques from XP

Short releases: This prevents software-drift. There is always a fully working version in CVS (or SVN). If there might be an integration problem between modules, it is detected in an early stage and easy to solve. A release happens every 4 to 6 weeks, and contains a fully working system, though it does not have all the functionality of the end-product. This works very well and motivates the team and assures the boss (professor or team-leader) that the project is on track. If it is not on track appropriate steps are undertaken without having too much damage.

Simple design: A complex design is hard to change. It is also impossible to explain to new people coming into the project. It slows the speed of development and discourages the exploration of alternative solutions. Worst case scenario is a project that has virtually come to a full stop, nobody dares to change much in the code and programming on the robot means, in practice, that the person is mostly debugging.

Testing: An essential feature to get more certainty that the complex system will actually work. If a test fails because something has changed the mistake is easily found and fixed. The software grows and if it is uncertain if the basic systems work as expected or not, there is no way to predict the behavior of the software in the future.

Pair programming: Pair programming contains immediate code refereeing and learning from each other without too much explicit training. One of the problems is that in a too small group there are several specialists working and a pair can quickly turn into one code-warrior and one viewer without learning.

Collective code ownership: Everybody is allowed to change everything. Waiting (hours, days) for someone to change a piece of code which could be fixed in minutes by oneself is frustrating, slows down progress and gives rise to friction in the team.

Continuous integration: To be certain that errors or incompatible modules do not occur, continuous integration is an essential part of a professional agile way of working. Persons can work for long periods of time on their own island only to find out that during integration something very important was very different than expected.

On-site expert: In XP an on-site customer is preferred as the expert on what the end-product should look like. In scientific projects as RoboCup this is more difficult and usually there is no customer. An on-site expert is recommended to keep the group on track. Questions about algorithms, architectures and planning issues are quickly resolved.

Steering: A complicated project has to be steered. In XP it is compared to driving a car: one cannot point the nose into a certain direction and pay no more attention to the driving. Steering is done continuously with small adjustments all the time. This might result in uncertainty because there is no grand/final plan while working. On the other side, such a plan is usually adjusted many times and only gives superficial certainty and a lot of overhead.

Coding Standards: To be able to read all the code as if it was your own, coding standards are needed. Today this is often an automated procedure in the programming environment.

Coaching: The coach aids persons with the adopted way of working. Frictions will arise due to different working habits, and the coach eases the transfer from a naive approach to a structured one. The coach is not the bogyman, but usually talks in general terms. The coach does not punish a team member if something goes wrong but is instead looking how to solve the problem. Nobody is to blame, instead everybody works on the solutions.

Strategic vs. implementation decisions: A big difference exists between these sort of decisions. They can be made by the same persons but it is good to separate them and explain explicitly what sort of decision is being made. Inexperienced persons can make (some) implementation decisions, but only the experts in the team can make the strategic decisions regarding overall architecture, the algorithms to be used and hardware changes for example.

Practices from experience in robot projects

Self-monitoring of the robots: Together with reliable software one should pay attention to reliable hardware. If one is programming a behavior and the robot starts to shake after running the code, the first thing to do is check the new code. This reasoning is not always valid and can be plainly wrong. There are many possible problems with the hardware and software. Fully automated checks of the important systems should give the user a hint whether it is the new code causing the problems or something else. The monitoring can be automated and on-line, which decreases debugging time. It also helps with the control of a robot, if it can diagnose by itself what is wrong.

Round-trip engineering: Build modules and behaviors according to the specifications, test them in simulation, transfer them to the real robot and test them again. If the result is not close enough to what was expected the simulation has to be adapted or the implementation rechecked. A research question is how to automate the simulator tuning.

Active project management: Most of the people working on RoboCup are students. They join the team for a certain amount of time and leave the project. This is the ideal recipe for a failing project. Most of the knowledge leaves with the students and the new students have to learn everything anew. This causes a lack of progress on the long run.

3.2 Tools

Standardized hard- and software Preferably all the hard- and software should be of-the-shelf. The robot used is a commercial product, unless the robot itself is the research topic. The setup of the robot is standardized. One of the tools we build is an installer for the robot and development software, an automatic update procedure (one(!) button) for the drivers and the fully automated control software update procedures. *Every time* one of these buttons is pressed somewhere between five minutes (driver update) and a few hours (complete installation) is saved. Because it is easy to update the software it is more likely that during the games all the robots have the same version running.

Just a few years ago a standardized, of-the-shelf, component based system was lacking[14] and one of the aims of the robotic community. The standardization of the hardware is progressing appropriately, though on the software side it is still lacking. The approach used in modern software engineering is a visual based, modularized and agile[4] one. The visual approach forces to build modules and to standardize all the components. It becomes easier to test the software. To cite Manuela Veloso, who talked in Padova about RoboCup, she was "getting bored by the lack of progress", it was "time to do something new", the RoboCup community should "surprise her" and "be less conservative". The authors think it is due to the approach used in the different RoboCup teams that the RoboCup community is showing a slow-down, and not due to technological incompetence or another reason. The lack of progress is a social problem, not a technological one!

4 Results

The results of the proposed management approach are convincing but hard to convey. The development speed is very high while the work pressure remains constant. There was a steep learning curve due to the new robot, a new and unknown software tool and a new way of working (visual based and XP). To some of the problems we encountered we have some solutions. For the simple design the solution we use is extreme modularity. Everything we do is programmed in modules (in C++). The interfaces are rigidly defined and strong typing is used. In case of the visual programming environment, the graph built in this way *is* the architecture. It supports hierarchy in cases where it is necessary. This ensures flexibility and design overview. No design documents are used as the design *is* the real-time system itself, in case of the visual tool.

The problems with pair-programming is solved by getting more persons on the project. To be certain that modules are not messed up by unexperienced programmers we work on the problem together with the expert if the change is rather large. The on-site expert works really well. The rapid feedback of the expert ensures that if a mistake is made or a problem has to be solved it is done very swiftly. The steering done by the expert helps to keep the project on track. The coaching eases communication in the team. Nobody has a special place and everybody is treated as equal, which aids in the cohesion of the team and speeds up the development process. The approach of dividing the strategic and implementation decision is used to the maximum and it is a good working practice. Students get certainty about the project and leaders are assured no big mistakes are being made.

The estimated speed-up is hard to measure. In the mid-size league, a period of just a few months, with a handful of people working a couple of days a week resulted in our case in a working team that got through the first rounds. We lost to the European Champion and the second on the world ranking list of that year (2004), which is not a bad result for such a short period of working. In comparison, other projects of us, without the methodologies written down in this article, took one-and-a-half year *(three times as long)* to get the same quality.

5 Future Developments

In the Netherlands a research project is ongoing which tests these methodologies between research groups. Questions arise about how it works when dozens of people are working on a robotic project, in half a dozen different research groups. One group is researching how to make a 'virtual laboratory'. How can we work in different physical places and still have a decent project. One extra practice is a gate keeper on the versioning system (CVS/SVN). There can be many unstable branches, where groups or individuals can work, but there is only *one* stable branch. This branch has a gatekeeper who checks whether the code is according to the standards. If not then it does not go into the stable branch.

Another method is nightly updates and builds. All the unstable branches are updated with the stable branch code. If merger problems arise it is reported in the daily morning mail. Also all the branches are compiled from scratch, to check for problems. Everything is automated and the system emails one email in the morning to everybody on the email list with all the details.

6 Summary

In this article we presented the application of a recent SW development technique called eXtreme Programming to the realm of RoboCup. Some of the techniques do not seem applicable to working with robots, but other one are working out very fine. The result is a team that is expandable and has a high development speed. We advise other groups also to pay attention to the social aspects of complex high-tech projects such as RoboCup. This might be the first steps toward the maturation of the robot-industry.

References

1. http://www.volksbot.de.
2. http://www.us.aibo.com.
3. http://www-2.cs.cmu.edu/tekkotsu/.
4. Special report: Software goes extreme.
5. A. Bredenfeld, Th. Christaller, H.Jaeger, H-U. Kobialka, P. Schöll. Robot behavior design using dual dynamics. Technical report, GMD Report, 2000.
6. A. Bredenfeld, V. Becanovic, Th. Christaller, I. Godler,G. Indiveri, K. Ishii, J. Ji, H-U. Kobialka, N. Mayer, H. Miyamoto, A.F.F. Nassiraei, P-G. Plöger, P. Schöll, M. Shimizu . Ais-musashi team description paper.
7. Kent Beck. *Extreme Programming Explained*. Addision Wessley, 2000.
8. Silvia Coradeschi and Jacek Malec. How to make a challenging ai course enjoyable using the robocup soccer simulation system. In *RoboCup-98: Robot Soccer World Cup II*.
9. Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addision Wessley, 1999.
10. Roland Mandl and Bernhard Sick. *Messen, Steuern und Regeln mit ICONNECT*. Vieweg, 2003.
11. Howard E. McCurdy. *Faster, Better, Cheaper: Low-Cost Innovation in the U.S. Space Program*. Johns Hopkins Univ Press, 2001.
12. Peter Stone. Robocup as an introduction to cs research.
13. Peter Stone. Multiagent competitions and research: Lessons from robocup and tac. 2002.
14. B. Werger. Ayllu: Distributed port-arbitrated behavior-based control, 2000.