Coding conventions

Version	Date	Author	Description
1.4	13.4.2003	Tomas Backas	Updated: conclusions DE
1.3	21.3.2003	Tomas Backas	Updated: practical implementation, conclusions I3
1.2	6.2.2003	Tomas Backas	Updated: conclusions I2
1.1	1.12.2002	Tomas Backas	Updated: practical implementation, conclusions I1
1.0	30.11.2002	Tomas Backas	Updated: practical implementation
0.9	27.10.2002	Tomas Backas	First version

Table of contents

Introduction

Teamwork in software projects requires many things to run smoothly and free from problems and complications. One important thing is a consistent and clear coding convention. This means that the people who participate in the coding of the software all "speak the same language", i.e. they write similar code. This is useful when it comes to improving the performance of the actual writing of code. When people have the same idea of what the code should look like, it is easier for them to read their own and code written by other coders. Benefits of using a coding convention in a project are besides anybody being able to read (and edit) anybody's code; people who don't have a personal coding style learn the common way quickly and don't have to come up with an own style, and they make fewer mistakes in the code. Maybe the strongest argument for coding conventions is that 80% of the lifetime cost of a piece of software goes to maintenance, which speaks strongly for clear code. [1] Disadvantages with a coding convention are; people who have their own style may have difficulties adopting the new style, and coding conventions reduce creativity and might prohibit certain programming styles. [2]

Coding conventions are not necessary but will definitely make the project more efficient.

Application in this project

In our case, we have seven students with different coding experience, working part time on this project. The project is run by the students, and is not structured as professionally as many commercial/professional products. This will most likely lead to people having to help one another with their code, which might depend on some of their code etc. Here we have the perfect ground for a coding convention implementation. We must define a convention and enforce it, so it will help when these situations occur. The coding conventions are also very important since this particular project will most likely be further developed by the customer, by other people than the original developers. A structured code makes the transfer of the development less painful.

Basic rules

1. The rules will be agreed upon by the group.
2. When the decisions are made they are documented in this document.
3. All code in this project must follow the rules stipulated in this document.
4. The rules are enforced to meet the rules.

Influencing factors

- Tools used for development; tools sometimes have incorporated styles that may or may not be changed. If they are not changeable it must be taken into consideration when deciding on the rules. If changeable, what's the effort, what is the most efficient style?
- Common styles; are there any styles that are familiar to a majority of the programmers? Should it be used?

Agreed rules

All source code should be written according to Sun Microsystems, Inc's Code Conventions for the Java™ Programming Language [1] with the addition of normal comments throughout the source code, when possible to make difficult code clearer with them. Javadocs [3] will be produced from the code, and due to that programmers should follow the Javadoc documentation style.

Practical Implementation

Responsible person and coordinator for the practical implementation of the method "Coding Conventions" is Tomas Backas. The method is implemented through the following interactions:

- Group discussions; Wishes, problems, and thoughts on the subject coding conventions are handled during group meetings and through the information channels, mainly Phorum.
- Private discussions with the coordinator; Small problems and wishes are handled directly by the coordinator of the implementation of this method. Small decisions are also handled in this way.
- Group decisions, voting if necessary; Major decisions are made by a majority vote or decision inside the group. A unanimous decision is the goal, if necessary voting will decide.
- Code examination; The code is examined and evaluated by the coordinator for reporting purposes.

Conclusions

Phase I1

There was some group discussions on coding styles during this phase, see more about it below. The code was examined by the coordinator.

At this moment (30.11.2002) the project contains about 3500 rows of code (according to Burana), of which 1500 are comments. The large number of comments is mainly a product of using Javadoc, which we agreed to use, and which will come in handy when writing the documentation for further development. The size of the code will still grow a lot during this phase, but there are already some conclusions to be made from the code that is available today.

The programmers are very consistent with the agreed Java style, and the code they produce is consistent. Main observations made during this phase:

1. Different programmers produce surprisingly similar code. The code (style) the different programmers produce is pretty much alike. This is probably thanks to a very standardized programming language, i.e. Java.

2. One programmer has difficulties adopting the Java style, and prefers to use “C style” programming, e.g. with the block brackets on separate rows.

After discussing the usage of “C style" in the project, we agreed that it would not impact the other programmers’ work at all, and hence should be allowed on the condition that it is changed to “standard” style before the end of the project. This was evaluated as having minor impact on the project when making the decision, and after examining the situation retrospectively we can see that the decision was not bad.

We can make the conclusion that programming in a specified way is not always done in a minute. We will still try to make it a habit rather than an awkward experience. During the next phase the “standard” style will perhaps be more common than in this phase.

Phase I2

This phase was pretty much dominated by writing code, which was our intention from the beginning. Thus we can start to see a trend how things are working out with our plans on following a predefined coding convention to make things smoother. The code was examined by the coordinator 6.2.2003, during our “code freeze”. The project contained 7900 lines of code at that point. Of those, 1600 lines were comment lines.

We can still conclude that the programmers are writing very similar code, and it is hard to make out who wrote which part, if we don’t analyze the structure of the code itself merely the design or the layout of it, with a few exceptions.

Main observations:

- One programmer still has not adopted the Java block style, with the brackets on the same row as the leading statement.

- Javadocs are totally missing from some classes, and partly missing from others.
- Usage of inline comments is satisfactory.
- Old or “commented out code” exists in source code, totally natural during implementation phases.

The programmer who writes C style code may continue with this as planned, as this makes his work much more efficient. Javadocs are not crucial at this moment, but they are to be written during next phase. Old code fragments is not a problem, they are easily cleaned during the development in the next phase.

We will continue the project with confidence that it will grow into a product that is manageable by the post developers.

Phase I3

In this phase we had to focus on coding even more than in earlier phases since we were slightly behind schedule. This is best realized from the fact that, while in the last phase we used about 43 % of total work hours on coding, in this phase the number is closer to 65 %. In this phase we also had more use of a predefined coding convention, since we had to help each other and work together on same parts of the system more than before, especially when performing debugging and fixing of bugs.

Interesting is the gradually and unknowingly introduced way of reporting minor bugs to each other. When the programmer encountered a minor bug caused by another programmer or in another programmer’s area of responsibility, he coded a warning message into the system that let everybody know that the bug existed. Burana was still of course used parallel to this method, especially for bugs of greater importance.

The code

The code was examined by the coordinator 21.3.2003. The project now consists of about 17200 lines of code, of which 3300 are comments. This means that the number of lines has more than doubled during this phase. The comment percentage is approximately 19 %, and it has decreased during previous phases, but is now seemingly stabilizing.

The code still looks very consistent, throughout all the classes. The biggest differences are the use of in-line comments and Javadocs. In-line comments are used pretty often, by everyone, but in some places there are very many in-line comments that in practice are commented out lines, for testing and debugging purposes. These extra debugging lines will naturally be removed as the product is finished. Javadocs are still missing from some classes and methods, although we decided to add these during this phase. They will most likely appear shortly.

There is also some deprecated and commented out code in the files that will be removed.

Phase DE

Lines of code (total / comments): 24200 / 4600.

This method was most of all applied to the project in the beginning, when we decided how to go about with the coding in general. We decided to try using a standard coding style so that it would be easy to work with each other's code segments, if that was needed. We also decided upon this for the reason that it would be easier for the customer to continue developing the product after we had done our job.

This worked pretty well, and since we had chosen a programming language that is very straight forward (Java), the outcome was very pleasing. We had only one major difference, which was one of the coders intuitively writing old C style blocks. This was such a small difference, so we decided it could be dealt with later, if using the old style could speed up this particular programmer's coding. Another difference we noticed later on in the project was indentation. Since we used at least three different editors, which all had different default values, the outcome were slight differences in indentation for some users. To be wise after the event, we should have considered this and decided on an indentation style that everyone should have used.

At the end of the project we had to add lots of Javadocs, since people didn't do this during coding. I guess this is because lack of Java experience. Some wrote Javadocs consistently, while others forgot them almost completely. Luckily they could be added afterwards.

This method is maybe not so obvious when applied to a project like ours. There are lots of things you could define in a coding convention plan, but the implementation of this plan is not that easy when the project group consists of individual students who generally don't have mush experience these things. The coding convention could be quite effective; in our case it was focused mostly on bug fixing within the group and the possibility to develop the product further. All in all the method wasn't the most useful of the methods, but the use of it gave a good picture of what it could do for you in another situation. I think this method could be useful in future projects like this, it was after all quite ok, and I learnt the idea why it can be effective. I wouldn't have thought of it before.

References

[1] Sun Microsystems, Inc, Code Conventions for the Java™ Programming Language, 1995-1999, http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html

[2] Todd Hoff, C++ Coding Standard, 1995-2002, http://www.possibility.com/Cpp/CppCodingStandard.html

[3] Sun Microsystems, Inc, Javadoc, 1995-2002, http://java.sun.com/j2se/javadoc/index.html