Fakulteten för ekonomi, kommunikation och IT

Report

Software Engineering
Statement C

Date: 2006-10-02
Name: Henrik Bäck
Course: DAVC19
Instructor: Tim Heyer

Table of contents

Introduction

When communication is needed during software development different approaches are possible. If it's necessary to communicate with a person who is not familiar with programming language the use of some design artifacts may be a good idea. These documents should only be used for communication and should be discarded afterwards.

We need to communicate

There are several times through software development when communication is very important. If documents are used or not – it is always necessary to communicate in the developing group. This according to [1].

Familiar with programming language

According to [1] very much of the information that usually will be written in the design documents instead could be communicated through the group by social mechanisms. Only documents that contribute to the code and tests are valuably according to [2]. Therefore it is, according to [1] and [2], possible to make a smaller amount of documents and still make the system correct. If all the members of a group are familiar with the programming language and how to write tests these documents are enough for describing the design of the system. It is therefore not necessary to generate more documents than needed, because they are not going to be used. The only thing that really matters, according to both [1] and [2] is that the system is being built correctly.

With help of others

As said, the use of tests and programming code as the only design artefacts may be devastating for group members who are not familiar with those. Mathematicians and physicists are examples of group members who may not be familiar with programming languages. In these cases, according to [5], it works well to communicate through design artefacts and as described in [2] design artefacts may be produced if it is necessary. Therefore it is a good idea to communicate thorough design artefacts with persons who are not familiar with programming language.

Communicating with costumer

Customers are often not familiar with programming language and software development. As described in [3] the uses of use-case is a good way to capture requirements from a costumer without involving the use of programming language. It is here a good idea to use design artefacts only to improve communication, as described in [5].

Discarding made artefacts

It's is important to use the artefacts that are made to communicate, otherwise they should not be created; all though it can be a good idea to not update them after they have for filled their duty. As can be read in [4] there is not worth the effort to continue to update the artefacts. It is also said that if hey become out-of-date they can be misleading. Therefore it can be a very good idea to discard the artefacts made to avoid problems after they have been used.

References

[1] Extreme Programming Explained, Embrace Change
KENT BECK, CYNTHIA ANDERS
ISBN: 0-321-27865-8

[2] Benefits build [software development]
Knoernschild, K. Source: Software Development, v 13, n 3, March 2005, 44-6
ISSN: 1070-8588 CODEN: SDEIEV
Publisher: CMP Media Inc, USA

Abstract: What's the single most important activity your team can perform to get your project moving along? An automated and repeatable build! Why? Because an automated and repeatable build produces the only artifact that really matters to everyone associated with the project: the executable application. Certainly, a team may produce and use other valuable artifacts. But all those pretty documents will be quickly forgotten if you don't deliver the final product. Embodied in extreme programming's continuous integration practice, an automated and repeatable build can work wonders for your development team. First and foremost, it forces you to integrate early and often. You're guaranteed to always have a system that works. Of course, you need to follow two rules. First, all compile errors that surface must be resolved immediately. Since you should be using a version control system, such as CVS, this means that the CVS projects comprising your application should always be free of any compilation errors. Second, any unit test that fails must be immediately fixed. Unit and acceptance tests should be run after each change you make, and you should treat any test failure with the same urgency you'd give a compile error. While simple, these two rules are key elements because they're part of what defines an automated and repeatable build.

[3] The Unified Process, Explained
KENDALL SCOTT
ISBN: 0-201-74204-7

[4] Streamlining the agile documentation process test-case driven documentation demonstration for the XP2006 conference
Brolund, Daniel (Agical AB); Ohlrogge, Joakim Source: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), v 4044 LNCS, Extreme Programming and Agile Processes in Software Engineering - 7th International Conference, XP 2006, Proceedings, 2006, p 215-216
ISSN: 0302-9743
Conference: 7th International Conference on Extreme Programming and Agile Processes in Software Engineering, XP 2006, Jun 17-22 2006, Oulu, Finland
Sponsor: Exoftware;Philips
Publisher: Springer Verlag

Abstract: In far too many software projects the value of the documentation delivered is not high enough to motivate the effort spent to write it. An outdated

document can be as misleading as a good, up to date one can be helpful. This demonstration will show how unit tests complemented with descriptive comments can be used to generate documentation that is constantly up to date. Its demonstrated by example how both the static and dynamic features of a software system can be salvaged with very little effort to be presented to a bigger audience as relevant, readable documentation. © Springer-Verlag Berlin Heidelberg 2006. (2 refs.)

[5] Describing software architecture with UML
Hofmeister, C. (Siemens Corp. Res. Inc., Princeton, NJ, USA); Nord, R.L.; Soni, D. Source: Software Architecture. TC2 First Working IFIP Conference on Software Architecture (WICSA1), 1999, 145-59
ISBN: 0 7923 8453 9
Conference: Proceedings of WICSA1: 1st Working IFIP Conference on Software Architecture, 22-24 Feb. 1999 , San Antonio, TX, USA
Publisher: Kluwer Academic Publishers, Norwell, MA, USA

Abstract: The paper describes our experience using UML, the Unified Modeling Language, to describe the software architecture of a system. We found that it works well for communicating the static structure of the architecture: the elements of the architecture, their relations, and the variability of a structure. These static properties are much more readily described with it than the dynamic properties. We could easily describe a particular sequence of activities, but not a general sequence. In addition, the ability to show peer-to-peer communication is missing from UML (13 refs.)