Fakulteten för ekonomi, kommunikation och IT

Criticise

Software Engineering

Criticise for Niclas Kihlstadius

Date: 2006-10-20
Name: Henrik Bäck
Course: DAVC19
Instructor: Tim Heyer
Robin Staxhammar

Table of contents

Up-to-date documentation has it's downsides

In his report Niclas writes about always have complete and up-to-date documentation. But is it only a good idea to make all these documents? In this response, Niclas arguments will be put in another perspective.

Reuse of documentation

Niclas writes that reuse of software documentation is good and could be time and money saving. Always when reusing documentation from other projects the risk of errors in the software increases. As described in [5] errors occur in programs due to copy-and-paste of code. As well ass errors in program code can be reproduced by copy-and-paste so can errors in design documents be when these are copy-and-pasted. To copy and paste design documents may be good but can lead to problems.

Newcomers

Niclas also writes that design documentations are needed for newcomers to get an overview of the system. Indeed [1] writes that it is enough with an informative workspace where all project members can access information. In [2] the author says that "A large wallboard located in a public area promoted group interaction around the board, it enabled collaborative problem solving, as well as informing individuals about the local and global progress of the project". This makes it easy for a newcomer to under stand the project but also gives project stakeholders a quick and easy overview for the projects status.

Up-to-date documentation takes time

Niclas states that it is important to keep the documents up-to-date. This, according to him, because otherwise it will be hard to maintain the software. Furthermore he states that it is much easier to understand a higher level of concepts and that software documentation represent that.

The fact is that it is very hard to keep documentation up-to-date at all time. Often is the time and effort put into writing and updating the documentation not motivation enough for members to do a good job. This will result in a bad documentation or a documentation that is not up-to-date at all time. In [3] it is stated that unit tests complemented with descriptive comments can be used as a full documentation in software. This is easier for team members to do and unit tests also help the software to perform correct.[1] also states that the making of software documentation not only displease the writers of it but even the costumer, because the customer only pays for the software itself.

In addition existing software and tests can be used to create software documentation when it is needed. This is very helpful for team members and as shown in [4] there are ways to do this. When writing software documentation one little single type-error can become a big issue and that can make the whole documentation unusable. It is therefore better to use descriptive comments and tests as documentation.

References

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

[2] Back to the future: pen and paper technology supports complex group coordination
Whittaker, Steve (Lotus Development Corp); Schwarz, Heinrich Source: Conference on Human Factors in Computing Systems - Proceedings, v 1, 1995, p 495-502
CODEN: 002163
Conference: Proceedings of the Conference on Human Factors in Computing Systems. Part 1 (of 2), May 7-11 1995, Denver, CO, USA Sponsor: ACM
Publisher: ACM

Abstract: Despite a wealth of electronic group tools for co-ordinating the software development process, instead we find many groups choosing apparently outmoded 'material' tools in critical projects. To understand the limitations of current electronic tools, we studied two groups, contrasting the effectiveness of both kinds of tools. We show that the size, public location and physical qualities of material tools engender certain crucial group processes that current on-line technologies fail to support. A large wallboard located in a public area promoted group interaction around the board, it enabled collaborative problem solving, as well as informing individuals about the local and global progress of the project. Furthermore, the public nature of the wallboard encouraged greater commitment and updating. However, material tools fall short on several other dimensions such as distribution, complex dependency tracking, and versioning. We believe that some of the benefits of material tools should be incorporated into electronic systems and suggest design alternatives that could bring these benefits to electronic systems. (15 refs.)

[3] 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.)

[4] Design documentation retrofitting: an approach for retrieving reusable code
Loy, P.H. (Whiting Sch. of Eng., Johns Hopkins Univ., Baltimore, MD, USA)
Source: Fifth Annual Pacific Northwest Software Quality Conference, 1987, 156-73
Conference: Fifth Annual Pacific Northwest Software Quality Conference, 19-20 Oct. 1987 , Portland, OR, USA
Publisher: Lawrence & Craig, Portland, OR, USA

Abstract: Presents a technique for generating architectural design documentation from existing software. This technique has been proven to be effective in assisting the software professional in making decisions about the reusability of the code. The potential reusability of code is often severely constrained by the quality of existing design documentation. In many cases no graphical design documentation is available; in other cases the documentation is inadequate to assess the relationships between software program modules. To determine whether or not code is reusable, design documentation is needed that graphically depicts the program architecture showing the relationships between program modules in terms of hierarchy, calling sequences, and parameter passing. An approach is presented that enables the software professional to start with existing code and retrofit design documentation consisting of structure charts and interface details. A case study is discussed in detail (10 refs.)

[5] CP-Miner: finding copy-paste and related bugs in large-scale software code
Li, Z. (Dept. of Comput. Sci., Illinois Univ., Urbana, IL, USA); Lu, S.; Myagmar, S.; Zhou, Y. Source: IEEE Transactions on Software Engineering, v 32, n 3, March 2006, 176-92
ISSN: 0098-5589 CODEN: IESEDJ
Publisher: IEEE, USA

Abstract: Recent studies have shown that large software suites contain significant amounts of replicated code. It is assumed that some of this replication is due to copy-and-paste activity and that a significant proportion of bugs in operating systems are due to copy-paste errors. Existing static code analyzers are either not scalable to large software suites or do not perform robustly where replicated code is modified with insertions and deletions. Furthermore, the existing tools do not detect copy-paste related bugs. In this paper, we propose a tool, CP-Miner, that uses data mining techniques to efficiently identify copy-pasted code in large software suites and detects copy-paste bugs. Specifically, it takes less than 20 minutes for CP-Miner to identify 190,000 copy-pasted segments in Linux and 150,000 in FreeBSD. Moreover, CP-Miner has detected many new bugs in popular operating systems, 49 in Linux and 31 in FreeBSD, most of which have since been confirmed by the corresponding developers and have been rectified in the following releases. In addition, we have found some interesting characteristics of copy-paste in operating system code. Specifically, we analyze the distribution of copy-pasted code by size (number lines of code), granularity (basic blocks and functions), and modification within copy-pasted code. We also analyze copy-paste across different modules and various software versions (39 refs.)