The Importance Of Complete And Up-To-Date Design Documentation in Software Development <Release Candidate 1>

By Niclas Kihlstadius, University of Karlstad

Design documentation in software projects should always be created and kept complete and up-to-date, in order to achieve higher quality and lower costs. The following text argues why this is the case, covering both initial design and software maintenance.

Why not simply use code to communicate design ideas? Project members in general can more easily communicate using the design documentation, since the concepts used is more easily understood both by hard-core programmers and perhaps-not-as-technically skilled project stakeholders. This follows from the fact that it consists of several more or less high-level "views" of the system, omitting all the low-level implementation details. As a result, the key components of the system can easily be identified, and the tasks of planning and implementing them can be assigned to people. In other words, planning gets simpler. Indeed, [1] discusses some "advantages of explicitly designing and documenting a software archictecture", like the use of abstract system representations for communication with project stakeholders, and for work planning, and the fact that project managers can use them to assign tasks, simply because they do not contain a lot of details.

Another consequence of design documentation being more abstract is the fact that it's not as strongly tied to the implementation. Good design documentation might very well be reused in other contexts. E.g. consider design patterns. Reuse, as allways, saves time and therefore hopefully money as well. [1] also writes, "it may be possible to develop product-line architectures where the same architecture is used across a range of related systems".

Design documentation should also be kept complete and up-to-date. To realize this, consider the fact that development teams are dynamic. People leave and new arrive. Newcomers need an overview of the system, and a place to look for design information. It would be hard for them to work efficienly if the design wasn't documented at all, and very confusing if it didn't match the current system. As stated by [2], "One of the major advantages of this approach to documentation is the amelioration of the Mythical Man Month effect [...] When new programmers join the project they do not have to depend completely on the old staff for their information."

Also worth noting is the importance of proper design documentation during software maintenance. Software itself can very well survive its original team, and therefore the successors need something to conduct to gain knowledge of the system. Simply reading source code is not very efficient on its own. This is justified by [3]:

> The most difficult problem in modifying software systems is understanding the intent of the original programmers. Although tools that support program comprehension on source code level are of grat help, adequate documentation is the most obvious and efficient way to support this comprehension process. Software documentation is a necessity to enable maintenance, and increasingly attention is being paid to it in practice.

Obvioulsy, all changes done by the maintenance team must be documented in the same manner. The availability of earlier documentation should be of great use when doing this. E.g. a developer who needs to create a new design diagram, could simply look at the other similar diagrams in order to understand how to do this properly. In fact, according to [4], "Results show that, for complex tasks and past a certain learning curve, the availability of UML documentation may result in significant improvements in the functional correctness of changes as well as the quality of their design."

As seen, there is a need for proper design documentation, but it must be produced with care. If not correctly maintained, it will confuse the developers, which ultimately affects the quality of the software. This is supported by [5], who states:

> Empirical data shows that software documentation products and processes are key components of software quality [Card87, Lien81, Romb87]. These studies show that poor quality, out of date, or missing documentation is a major cause of errors in software development and maintenance.

To conclude, software engineering projects can benefit a lot from complete and current design documentation. Do not consider it as simply a pile of papers describing the system. Instead, see it as a medium for clear communication between project members, as a source for instant understanding and productivity, and, more importantly, as a guide for producing high quality software.

~ 450 words

References

[1] Ian Sommerville, "Software Engineering", 7th edition, Addison Wesley, 2004

[2] Parnas, D.L., Clements, P.C., "A rational design process: how and why to fake it", Proceedings of the International Joint Conference on Theory and Practice of Software Development (TAPSOFT) , 25-29 March 1985

[3] Johannes Samestinger, "The role of documentation in programmer training from conventional documentation to literate programming, hypertext and object-oriented documentation"

[4] Arisholm, E., Briand, L.C., Hove, S.E., Labiche, Y., "The impact of UML documentation on software maintenance: an experimental evaluation", IEEE Transactions on Software Engineering, June 2006

[5] Cook, C.R., Visconti, M., "New and improved documentation process model", Fourteenth Annual Pacific Northwest Software Quality Conference, 1996

[2] Parnas, D.L., Clements, P.C., "A rational design process: how and why to fake it", Proceedings of the International Joint Conference on Theory and Practice of Software Development (TAPSOFT) , 25-29 March 1985

Abstract: Software Engineers have been searching for the ideal software development process: a process in which programs are derived from specifications in the same way that lemmas and theorems are derived from axioms in published proofs. After explaining why one can never achieve it, this paper describes such a process. The process is described in terms of a sequence of documents that should be produced on the way to producing the software. The authors show that such documents can serve several purposes. They provide a basis for preliminary design review, serve as reference material during the coding, and guide the maintenance programmer in his work. They discuss how these documents can be constructed using the same principles that should guide the software design. The resulting documentation is worth much more than the `afterthought' documentation that is usually produced. If one takes the care to keep all of the documents up-to-date, one can create the appearance of a fully rational design process (15 refs.)

[3] Johannes Samestinger, "The role of documentation in programmer training from conventional documentation to literate programming, hypertext and object-oriented documentation"

Abstract: High-quality software documentation reduces the maintenance burden and improves productivity by enhancing reusability. Well documented software systems are also needed for students to learn from designs and implementations of experienced engineers.

Documentation is neglected in software education to a great extent. Neither documentation skills are taught, nor well documented systems are used for learning purposes. The availability of programming tools already plays a major role for choosing a programming language for software education. Usually and unfortunately, neither the availability of documentation tools nor the availability of documentation support in programming tools have an impact on this choice.

Furthermore, conventional documentation does not seem to be very suitable and attractive for learning and teaching. This paper outlines how concepts like literate programming, hypertext and object-oriented documentation can be combined to improve software documentation quality and accessibility. This can result in a more effective education of our future software engineers.

[4] Arisholm, E., Briand, L.C., Hove, S.E., Labiche, Y., "The impact of UML documentation on software maintenance: an experimental evaluation", IEEE Transactions on Software Engineering, June 2006

Abstract: The Unified Modeling Language (UML) is becoming the de facto standard for software analysis and design modeling. However, there is still significant resistance to model-driven development in many software organizations because it is perceived to be expensive and not necessarily cost-effective. Hence, it is important to investigate the benefits obtained from modeling. As a first step in this direction, this paper reports on controlled experiments, spanning two locations, that investigate the impact of UML documentation on software maintenance. Results show that, for complex tasks and past a certain learning curve, the availability of UML documentation may result in significant improvements in the functional correctness of changes as well as the quality of their design. However, there does not seem to be any saving of time. For simpler tasks, the time needed to update the UML documentation may be substantial compared with the potential benefits, thus motivating the need for UML tools with better support for software maintenance (36 refs.)

[5] Cook, C.R., Visconti, M., "New and improved documentation process model", Fourteenth Annual Pacific Northwest Software Quality Conference, 1996

Abstract: Inaccurate, missing, incomplete, or out of data documentation is a major contributor to poor software quality. A four-level software system documentation process maturity model and assessment procedure developed by the authors was described in 1993. This paper briefly reports the results of 26 assessments of software projects at 7 different organizations. These assessments have yielded promising results plus informative comments and suggestions about the assessment process. The major part of this paper describes improvements to the assessment process based on these comments and suggestions. It also includes the results of 25 software projects at 7 different organizations using the new process (8 refs.)