Knowing Everything about the Requirements for a Computer-Based, Software-Intensive System

Daniel M. Berry

Computer Science Department
University of Waterloo
200 University Ave. West
Waterloo, Ontario N2L 3G1
Canada
FAX: +1-519-746-5422
dberry@csg.uwaterloo.ca
http://www.cs.technion.ac.il/~dberry

We are all aware of the problems with computer-based, software-intensive systems (CBSISs) not only these days, but also, since we first started to develop CBSISs. They do not do what they are supposed to do; they do not handle surprises well; they are not reliable; and they have poor user interfaces. We call them "brain damaged".

When we are designing a CBSIS, we have to think of everything, everything the CBSIS should do and everything the CBSIS should not do. Thinking of everything means figuring out what anyone using the CBSIS would want it to do under any circumstances and figuring out all possible stimuli, including multiple stimuli, to which the CBSIS must respond. In short, thinking of everything means figuring out all eventualities in all combinations. Like building architects and contract drafters, we CBSIS developers have to think of everything, even things of which no one in his or her right mind would have ever dreamt.

The real cause of any problem with CBSISs is really very simple. A problem happens simply because at no time in the development of the CBSIS to date, did anyone in the team that developed and maintains the CBSIS think of the problem long enough for action to be taken to solve that problem. By "team", I mean the entire set of people whose thoughts and actions affect the development and maintenance of the CBSIS. This group certainly includes all stakeholders, but can include also passers by who happen to interact with other members of the team in ways that affects the CBSIS's development. By "not thinking of the problem long enough", I am referring to the fact that many times, someone thinks of an issue briefly, but due to circumstances, including forgetfulness, sloppiness, and overloading, he or she fails to get the issue into the project's consciousness, often just by failing to write the issue down on a piece of paper. Consequently, the issue evaporates, never to be considered again until the CBSIS fails in a way that brings the issue to the forefront.

In other words, I am assuming that CBSIS developers really desire to avoid all problems and that once they are aware of a problem, they can in fact deal with it. Granted, there are really unsolvable problems, but by and large, with most CBSISs, being aware of a problem is a sufficient condition for its eventual solution.

Therefore, the solution to the problem of CBSIS failures is to make it possible for the team to think of everything. Clearly, we cannot expect any finite group of people in any finite amount of time to think of everything. In the words of Don Gause, no matter how much we succeed to learn about a CBSIS, there will always be that unknown issue that will rear its ugly head when the CBSIS is running; he calls these issues,"nature's last laugh".

Thus, we can expect that techniques that increase human participation in CBSIS development projects will help, while techniques that limit human participation will hinder our attempt to think of everything. We can expect that techniques that get people to think about CBSISs in a variety of new ways will help, while techniques that reduce thinking about CBSISs will hinder. Consequently, I am both pessimistic and optimistic about CBSIS development technology.

I am pessimistic when I see technology being touted as *the* solution to problems or as a way to avoid problems. I get particularly upset at method and tool evangelists who claim that their methods and tools will allow the users of the method to build better CBSISs because the method and tools themselves discover more problems. That is, they are promoting a reliance on the methods and tools and a way to reduce human involvement in the construction of the CBSISs. They claim that methods and tools will do the work for us. I just cannot see how any formal method or mechanized tool can help a designer discover a problem if the designer does not think of it. Indeed, concerns over following the method correctly may make it even harder to think of things not readily apparent.

I am optimistic when I see technology being offered as additional means to help the human developers see the CBSISs being developed under different lights, to apply human thinking and creativity, or to get different people involved in the projects applying the different technologies to the problems. In other words, as I have said elsewhere [1, 2], the biggest benefit of formal methods may be in bringing a group of people skilled in abstraction to work on the CBSIS, providing yet other views of the CBSIS, and attacking the problem with different kinds of thinking.

Therefore, I believe that the most important contribution that the field of requirements engineering is simply making people aware of the paramount importance of discovering and inventing requirements that deal with all issues that might come up. The next most important contributions of the field has to be a set of techniques, be they technical, managerial, social, or even psychological, to get more people into the development teams and to get them thinking more thoughts in more different ways about the CBSISs they are developing.

We have to help software engineers understand that just spending more time studying a problem and understanding its requirements before plunging into design and implementation pays off. We know that an error caught during requirements specification for a CBSIS costs two orders of magnitude less to fix than if it is caught after delivery [3]. We also know that devoting 25% as opposed to 0% of a total CBSIS development budget on the study phase reduces the cost overrun of the development from about 80% to about 5% [4].

Acknowledgments

The author thanks Don Cowan and Maria Nelson for comments on an earlier draft.

References

[1] Berry, D.M., “Formal Methods, the Very Idea, Some Thoughts,” in *1998 Monterey Workshop on Engineering Automation for Computer Based Systems*, Monterey, CA (October 1998).

[2] Berry, D.M., “Formal Methods, the Very Idea, Some Thoughts on Why They Work When They Work,” *Electronic Notes in Theoretical Computer Science* **25**, Elsevier (1999). http://www.elsevier.nl/locate/entcs/volume25.html.

[3] Boehm, B.W., *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ (1981).

[4] Forsberg, K. and Mooz, H., “System Engineering Overview,” in *Software Requirements Engineering*, ed. R.H. Thayer and M. Dorfman, IEEE Computer Society, Washington (1997). Second Edition.