Software Quality Journal **Vol. 10 No. 3, 2002**

Editorial: Why Good Software Goes Bad

In preparing several courses for my department's new on-line master degree program, I have been using a courseware software system that supports the delivery of academic courses over the worldwide web. This system was adopted by my university and is used by faculty in a wide-range of departments: courses in the humanities, physics, veterinary medicine, engineering, and, of course, computer science.

The system users' computer backgrounds range from naïve to very savvy. Faculty and support staff members use an email list to post problems and solutions to using the system. Users' problems can be very basic, from how to use basic browser and editing features, to more sophisticated problems related to the interface between browsers and the system. Some of the problems are due to the awkwardness of the system interface; other problems are caused by the system design. Users can solve many of the problems by "workarounds", a series of often-convoluted actions.

Here are some of the problems that I have experienced. Developing quizzes on the system is very tedious; it involves a sequence of actions on browser forms for each question. (Third parties supply some quiz development support tools, but the one I tried also has glitches.) There is no way to move a portion of one course to another one --- I cannot take a lesson developed for one course and move it to a similar one. As far as I know, there is no workaround. As a result, instructors must often do redundant work when preparing two similar, but different courses, for example an on-line and an on-campus section of the same course. Another problem is that the system does not support co-instructors. It gets confused when a course has more than one instructor. From my perspective most of these problems are caused by design faults; a better design could eliminate or at least lessen the problems.

In spite of the problems, the system provides the required primary functionality fairly well. Instructors can prepare and post lecture notes, on-line quizzes and exams, audio and video, and other materials. Grades can be updated automatically and posted with security.

The system was selected by a university committee, which evaluated several candidate courseware systems. I believe that the committee would still select this system over the others, even with the knowledge of our several years of sometimes frustrating experience with it.

In one sense, this is very good software; it performs its major functionality well --- probably better then its competition. In another sense, it is bad software; using it can be very frustrating, awkward, and much more time consuming than necessary. Really solving the problems in this system may require a major redesign effort.

This is good software in that it fills a great need, but it has numerous design flaws. How did it reach this state? Probably in the same manner that other heavily used software systems gain glitches. The software starts as an excellent solution to a problem in a small application domain. However, its success leads to an expanded user base, with an

expanding set of requirements. And the original developers could not have predicted many of these requirements.

From what I have heard, the courseware system grew out of one professor's "home brewed" software for on-line support of his own courses. From there, it grew to supporting other courses in his department. The system was well liked by its small group of users, and it became the basis for a startup. After it reached a wider audience, and was adopted by major universities, the system began to be used in ways that the original users, a few professors in one department, could not have predicted. The system also had to adapt to many new versions of new browsers, and web servers.

As the user base grew from use by a dozen or so instructors to perhaps thousands of users in widely varying environments, the system requirements became much more difficult to meet. Novice users needed to have a very easy to navigate user interface, and they needed to be able to use the system without knowing much about the underlying mechanisms involved. Still the basic design of the system depended on an understanding of hyper-links, and the system included both internal and external links that instructors will tend to use. Experienced users needed to have shortcuts, and a way to avoid using the sometimes-tedious user interface. As it is now, the system does not satisfy either the novice or experienced user. As judging from the courseware mailing-list postings, both groups are often frustrated.

We have also observed a tremendous growth in the required on-campus resources needed to support the system. This system is supported by a centralized server, which runs the courseware software. Instructors and students are clients who access the system through browsers. Each year, the system seems to be replaced with more powerful hardware to maintain adequate response time. Professors upload documents, images, video, and audio; students upload solutions to assignments. These uploaded items can be quite large; thus, system storage requirements have grown significantly. At least two full-time staff members support the system and provide service to the faculty users. It seems that we are back in the mainframe business!

Could there be alternative designs that would make the system easier to use and manage? Absolutely. A skilled group of software engineers could develop a set of requirements that encompasses all the stakeholders' needs. However, the current set of stakeholders and their requirements were not known during early development, and the underlying software architecture was designed on the basis of a much smaller set of stakeholders with simpler needs.

Could the system designers determine a comprehensive set of requirements early in the system development? Probably not. This system grew because of its early popularity; I doubt the small group of early developers could have predicted future users' needs.

Could the system designers have developed a much more flexible system, one that could be more easily adapted to meet future needs? Possibly. However, like in most "bootstrapped" companies, the initial developers did not have the resources to produce a highly flexible system. They could not devote much energy into making the system scale up to a larger user community that would exist in a possible future. They had to satisfy immediate needs for a few customers quickly.

The goal of much of the new software development technology is to increase the adaptability of software. The promise of object technology, design patterns, and agile processes (e.g., *extreme programming*) is to make software easier to adapt and reuse. So,

perhaps if the developers made good use of these techniques the system would have grown to better satisfy the growing demands of users.

However, to use these new technologies to make a system easier to adapt, you must know what future changes will be made. For example, each design pattern will make certain changes easier, and other changes more difficult. The Visitor pattern applied to a compiler makes it easier to add compiler phases, like optimization, and harder to modify the syntax of the language that is being compiled. So, we need to know future requirements to make optimal use of technologies such as design patterns. Unfortunately, even witches and wizards find that "divination is one of the most imprecise branches of magic"[1].

In the end, I see two the key reasons that often-used software is usually "clunky" --- hard to use, with many glitches. Early developers could not predict future user demands, and if they could they often did not have the resources to build the most flexible, scaleable design.

Remember this, just about the only software that never has problems is usually called "shelfware".

Please share your thoughts on these and other software quality issues. Send your comments to me at bieman@cs.colostate.edu.

James Bieman
Fort Collins, Colorado
U.S.A

[1] Professor McGonagall to Harry Potter in *Harry Potter and the Prisoner of Azkaban*, Scholastic Press, 1999, p. 109.