Astronomical Data Analysis Software and Systems VI
ASP Conference Series, Vol. 125, 1997
Gareth Hunt and H. E. Payne, eds.

Experience with a Software Quality Process

Richard A. Shaw and Perry Greenfield

Space Telescope Science Institute, Baltimore, MD 21218

Abstract. We describe the methodology and virtues of software design and source code inspections as vital elements of a quality management process. We also recount the experience of the Science Software Group at ST ScI with these methods, which were formalized and implemented during the past year.

1. Introduction

To hear W. Wayt Gibbs (1994) tell it, the software crisis identified more than 25 years ago is alive and well in the mid-1990s. Citing well known projects with spectacular cost and schedule overruns, Gibbs points out that losses of hundreds of millions or billions of dollars on failed software development projects are distressingly common in industry. The statistics are sobering indeed: something like 25% of all large software development projects are outright failures, and another 50% either do not implement all the promised functionality or are not used. According to Gibbs, "the average software development project overshoots its schedule by half." Another study of large distributed systems showed that over half exceeded their cost estimates, two-thirds exceeded their development schedules, and almost nine out of ten involved substantial redesigns. The problem, argues Gibbs, is that a solid engineering methodology has yet to be fully developed for what is still in large measure the art of computer programming.

Perhaps such grim statistics do not hold for software development efforts in astronomy. Yet many participants in the ADASS conferences have probably been involved in some software project that, for whatever reason, could have turned out a little better: perhaps the project took longer than anticipated, or the planned functionality was never fully achieved, or the result was plagued with bugs.

The importance of computers and software to astronomy has grown dramatically during the past few decades. Yet the specialized needs of astronomers are still, in large measure, addressed with custom-built software systems and applications. Generally, this software must be built and maintained with rather modest (by industry standards) resources. So it is important to identify the best methods of software engineering in order to minimize the risks associated with software development, while increasing productivity and product quality.

Our experience in the Science Software Group at ST ScI is that the cost of correcting bugs in a large software system that is distributed to the community is rarely less than one-half staff day, and can consume several days in some cases. This surprisingly large cost includes the effort to reproduce a problem, isolate the bug, devise and test a fix, install the fix in the configured system, document the changes (in the source and user documentation), and advertise the fix and likely user impact in the release notes and on-line news postings. Clearly, any process that identifies defects before code is released to the community is worth investigating.

2. Quality and the Software Development Process

There is compelling evidence that a studied, methodical emphasis on software quality is the best means to contain development costs and minimize schedule delays, particularly for larger projects. As a result, a number of software companies have, over the last two decades, instituted technical review processes. The benefits of a technical review process flow from a few key aspects of the software development process. These include the observation that all software developers are blind to certain kinds of defects in their own code, and that other developers—with different blind spots—can discover them with only modest effort. Secondly, the sooner software defects are discovered, the cheaper and simpler they are to fix. There is no single, correct method for assuring defect-free software, but it does take a serious commitment on the part of management for a software quality process to succeed.

Software development can be broken down into several distinct activities, including: problem definition, requirements analysis, high-level design, detailed design, construction, integration, unit testing, system testing, maintenance, and enhancements. Although our focus in this paper is on the design and construction phases, an effective software quality effort is characterized by a focus on identifying and correcting defects at all stages of development. A successful software quality effort must also be adaptable to local circumstances and therefore have a feedback mechanism. Most importantly, though, the process must produce results that are quantifiable, and the effects of local adaptations must also be quantifiable. Without that, it will never really be known whether or how well the process is working, and whether it is worth the effort.

3. The Review Process

According to various studies (see McConnell 1993 and references therein) the single most effective means to identify and correct code defects is the review process. This paper focuses on design and code reviews, although formal reviews can be applied to any stage of software development. The most formal variety of reviews—inspections—typically catch about 60% of the defects present in software (Jones 1986). While these rates are generally much better than that achieved by simply testing the end product, the kinds of errors found with each method are often quite different. In this sense, design and code reviews complement, rather than replace, testing as a tool for assuring quality software. A significant difference between inspections and testing, though, is that defects are identified and corrected in one step. Testing simply identifies a defect—it reveals nothing about how to fix it. Formal design and code inspections provide an effective means to catch defects early, where they are easiest and cheapest to fix.

Reviews can actually take a number of forms, including management reviews, walkthroughs, code reading, and inspections. The first of these is usually a poor means of discovering problems, though it may serve other useful purposes. Code readings or walkthroughs are more effective, and may be more appropriate for very small programming groups, but these methods are usually less effective at uncovering defects than formal inspections (McConnell 1993). Design and code inspections, being the most formal kind of review, involve preparation by the participants, a meeting, and a formal inspection report. There are three distinct roles in the inspection process:

The Moderator handles the review logistics and ensures that the review materials and reviewers are prepared before allowing the inspection to proceed. She keeps the reviews focused on detecting (rather than solving) problems, and records the defects found by the reviewers. The moderator limits the review meetings to 1.5–2 hours, and enforces proper review etiquette. She prepares an inspection report listing all defects, plus various statistics (e.g., time spent by the reviewers in preparing for the meeting, the number and type of defects found, etc.). It takes some training and experience to be a good moderator.

The Author notifies the moderator that a review should be scheduled, and distributes paper copies of the design or code to be reviewed. The author answers the reviewers' questions during the review meeting, and afterward documents how each identified defect was corrected or addressed. It is the author's responsibility to partition larger projects into segments that can each be reviewed within the time constraints for each review.

The Reviewers (usually two) must be technically competent and have no managerial role over the author. They review the design or code in advance, though it is best to limit preparation to roughly 2 hours per review meeting. (If they need more time, it is usually a sign that the review product should have been partitioned into smaller parts.) During the meeting, the reviewers should stick to technical issues and practice proper review etiquette. Most programmers can be successful reviewers with little additional training.

A significant point is that management must *not* participate directly in the reviews. For the process to succeed, the authors must not perceive the review as an open forum on their abilities or performance. The purpose of the reviews is to improve product quality, and the presence of management at the review undermines that purpose. Managers should demonstrate commitment to the process by not letting schedule pressures shortcut or otherwise cripple the review policy. The point is to distribute responsibility for the product to the review team, not just the author, and to reward the quality of the review and the product as corrected by the review. Management should, however, correct problems with the review process, such as poor or irresponsible behavior on the part of the participants. Management should see that the recommendations of the review panel are addressed, and should override the panel only under extraordinary circumstances.

It is not possible in a short paper to cover all the issues relating to software inspections. See McConnell (1993), Freedman & Weinberg (1990), and Humphrey (1989) for more detailed descriptions of the process, and the means by which its effectiveness can be measured. The use of software inspections is not a "silver bullet" that will solve all problems. It is only one tool of many intended to increase software quality and productivity, but it is one of the most effective.

4. Experience in the STSDAS Group

Design reviews and source code inspections were formalized and implemented in the Science Software Group at ST ScI during the past year. While objective, quantitative data on quality control processes are generally difficult to come by, we have endeavored to keep detailed records of the review results, as well as of the process itself, in order to assess their efficacy. To date, six applications and one system-level utility have been reviewed. Reviews of modest-sized applications have taken approximately one half to one week of staff time to complete.

This quality management process has been very successful at detecting and eliminating software defects early in the development process, and has unquestionably resulted in higher quality software products. Information on defect rates, defect categories, etc. is still a bit sketchy. The process itself has also been generally well received by the programming staff. As a result, a number of sociological benefits are being realized, including a transfer of coding expertise (usually) from the more experienced to the less experienced programmers, a broadening of knowledge within the group about the software system as a whole, and a greater sense of teamwork and shared responsibility for the software products. It is fair to say that the group is still gaining experience with this process, and its value to the organization is still being evaluated. But as of this writing, our interest and commitment to this process is quite strong.

References

Freedman, D. P. & Weinberg, G. M. 1990, Handbook of Walkthroughs, Inspections, and Technical Reviews, 3rd Ed. (New York, NY: Dorset House Publishing)

Gibbs, W. W. 1994, Scientific American, 271, 86

Humphrey, W. S. 1989, Managing the Software Process, Chapter 10 (Reading, MA: Addison-Wesley)

McConnell, S. 1993, Code Complete (Redmond, WA: Microsoft Press)

Jones, C. 1986, Programming Productivity (New York: McGraw-Hill)