CASE TEACHING NOTES
for
"A Walk Through Mike's Code:
A Case Study in Software Technical Reviews"

by
David R. Luginbuhl
Department of Mathematics and Computer Science
Western Carolina University

INTRODUCTION

This case is designed for an upper division software engineering class. It is assumed that students will have read about or discussed the concept of software reviews ahead of time. The case could be handed out ahead of time for an in-class discussion. The context for the case is loosely based on the Federal Aviation Administration's Advanced Automation System [Barlas96], a project to upgrade and expand the software for the nation's air traffic control system.

This is an example of a software technical review (or a "code walkthrough") gone wrong. Mike, the programmer, thinks he (and not his code) is the focus of the review. Annette (the facilitator) is trying (unsuccessfully) to get the review focused on Mike's program, which must interact with other programs as part of a larger system. The other two reviewers are programmers who are assigned to develop other parts of the system, so everyone at the meeting is a stakeholder. Students should be cautioned that this is an extreme example of what could go wrong in a technical review in order to make a point about the importance of teamwork and professionalism.

This scenario challenges the idea of the "lone ranger" programmer. Most software developers work as part of a larger team, and their efforts have an impact on their co-workers. Students see that they will likely be part of a development team, and they need to learn how to behave in such an environment. Mike has given us a clear counterexample, although students might observe that the other players are not entirely innocent.

The case also explores briefly the issue of unit testing. Mike has already subjected his code to some amount of isolated testing. He actually receives some praise for his approach to development of test cases, a non-trivial task for even relatively simple programs.

Objectives

- To introduce software engineering students to the concept of a software review.
- To reinforce the idea that software engineering is about teamwork.
- To introduce concepts related to unit-level software verification and validation.

CLASSROOM MANAGEMENT

Most software engineering texts cover software reviews at some point, and it would be good to have the students read about this before examining this scenario. Other possible resources are given below.

There are a number of variations to teaching this case. Three are given below:

1. Students read the case and answer a number of questions before class to prepare for discussion (see questions in Blocks of Analysis). In-class activity is an instructor-led discussion analyzing the review, pointing out where things went wrong, and how things might have been improved.
2. Students discuss the case in small groups, again with a set of questions that help the students to critically think about the review.
3. Students write (or, even better, act out) alternative scenarios where things go better or where something different goes wrong.

BLOCKS OF ANALYSIS

A list of questions for the students to answer ahead of time can be tailored for a particular class, depending on what points the instructor wishes to emphasize. All of them could be addressed, though perhaps not in one class period. They fall into the following categories (possible responses are included):

Software Development as a Team Activity and as a Part of Systems Development

1. What was the purpose of Mike's code?

   As stated in the narrative, Mike's code detects "proximity to the edge of the tracking radar's range and notifying the system to prepare for handoff." To rephrase, this code lets the system know that another radar should prepare to track the aircraft, since the aircraft is near the edge of the current radar's range.

2. What impact did Mike's code have on the rest of the system?

   If this code doesn't work properly, it's possible that the system wouldn't be able to track the aircraft when it leaves the current radar's airspace.

3. How do the standards contribute to teamwork and systems development?

   Standards contribute to communication by ensuring that everyone is reading from the "same sheet of music." As an agreed-upon set of values, they provide a foundation for personal reliability: one developer can depend on another developer's code, knowing that it was constructed according to standards.

Software Technical Reviews as a Part of Software Verification and Validation

4. What problems did the reviewers have with Mike's code?

   The most serious problem noted was the lack of following standards. Re-use of an existing routine was a possibility, but the existing routine was not included in the code (perhaps Mike had a fuller explanation for not incorporating the existing routine into his code, but the review degenerated and collapsed before he could fully explain).

Also, variable names were not descriptive, another deviation from standards. In both cases, the problem is that the code could be harder to maintain and harder to integrate with other modules. The instructor could use this as an opportunity to emphasize the fact that the cost for software maintenance is generally much higher than code development. For example, Schach (2002), pulling together a number of studies of software development efforts, asserts that "about two-thirds of total software costs were devoted to maintenance."

5. What aspects of the code did Mike receive praise for?

   Mike appears to have thoroughly tested his code at the unit (module) level. He even derived a non-obvious but possibly critical requirement for his code from some emergency requirements. Students can see that a set of requirements sometimes imply an additional requirement not recognized by the customer. Students can also see the importance of testing for boundary conditions, e.g., Mike's consideration of zero altitude.

6. Did Mike have any reason for not using existing code?

   Paul pointed out that the code available for reuse did not handle certain situations needed in Mike's code. This could lead to an interesting discussion on the pros and cons of reuse—was Drake's code insufficient for reuse? Could there be a reason that Drake didn't include the mentioned contingency? Did Mike consider ways to extend Drake's code to handle the emergency situation?

7. What aspects of Mike's product were reviewed?

   The text mentions specifically adherence to standards and unit testing. Implicit in these aspects are issues of maintainability and readability. The ability of other programmers to read and understand a piece of code has implications for integration (programs written by others may rely on this code) and maintenance (future maintainers will need to be able to understand the code).

8. Do you believe that Mike was correctly following standards? Why or why not?

   We get a clue as to why Mike didn't reuse Drake's code, but from his tone early in the review ("I found a pretty cool way..."), it sounds as though he was adopting a "lone programmer" attitude. Students might speculate as to whether Mike would have used Drake's code even if it handled the zero altitude contingency. As for the naming conventions, we don't know the details, but it's clear that he didn't follow the standards there either.

Proper Conduct of Software Technical Reviews

9. How well did the review follow the rules of good software reviews?

   A number of texts have sets of rules or guidelines for good technical reviews. Pressman (2001) has a list of 10. We examine three of those here:

"Review the product, not the producer." Here students may judge that the team failed miserably. In particular, Beverly seemed to focus on Mike, not on his code (note her constant use of "you" when talking about Mike's coding practices).

"Limit debate and rebuttal." Annette attempted to keep everyone on track and was willing to shut down debate when it seemed to her to be unhelpful.

"Conduct meaningful training for all reviewers." We don't actually know if everyone was trained, but students may infer by the participants' conduct that they could have used more training. The instructor could also ask students if it was wise for Mike's first review experience to involve a review of his own code.

10. How could Beverly have better framed her concerns?

    Pressman (2001) suggests framing issues as questions, allowing the producer "to discover his or her own question." On the variable names issue, Beverly might have chosen a variable name from the program and asked how a future maintainer will understand what it represents. Students should appreciate the difficulty in balancing the need for sensitivity and the importance of (constructive) criticism.

11. Did Annette do her job of facilitating well?

    A facilitator must strike a balance between exercising too much control and letting the review go off in an unhelpful direction. Some students might suggest that Annette should have interrupted sooner, but others may see this as too dictatorial. At the very least, it should be pointed out that Annette did try to refocus the review a couple of times. On the other hand, the facilitator is often responsible for ensuring that everyone is prepared for the review ahead of time. This was obviously not done.

12. How should this review have been handled differently?

    Students should recognize that training is a key issue in this scenario. Mike was obviously not ready for a review, and certainly not one involving his own code.

    The exchanges between Beverly and Mike indicate some personality differences that might have been caught earlier. Perhaps Beverly wasn't the best choice as a reviewer in this case. Or she might have been reminded ahead of time of the rules of a technical review. By making things personal, she contributed to the breakdown of this review.

13. What should Annette do now?

    Possible options that students could suggest: counseling both Mike and Beverly; referring both of them to training to learn how to participate in a review; sending a report of the review to management.

    Students may have mixed views on whether or not the participants' behavior should be reported to management. This could lead to an interesting discussion on whether such behavior (Mike's or Beverly's) should be held against an employee in a performance evaluation.

REFERENCES

- Barlas, S., and Burgess, A. (1996). Anatomy of a Runaway: What Grounded the AAS. *IEEE Software*, Vol. 13, No. 1, pp. 104-106.
- Collofello, J.S. (1988). *Introduction to Software Verification and Validation*. Curriculum Module SEI-CM-13-1.1, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.
- Fagan, M.E. (1976). Design and Code Inspections to Reduce Errors in Program Development. *IBM Systems Journal*, Vol. 15, No. 3, pp. 182-211.
- Freedman, D.P., and Weinberg, G. (1990). *Handbook of Walkthroughs, Inspections, and Technical Reviews*. New York: Dorset House.
- IEEE Computer Society (1997). *IEEE Standard for Software Reviews*. IEEE Std 1028-1997. New York: Institute of Electrical and Electronics Engineers.
- Pressman, R.S. (2001). *Software Engineering: A Practitioner's Approach*. New York: McGraw-Hill, pp. 202-209.
- Schach, S.R. (2002). *Object-Oriented and Classical Software Engineering*. New York: McGraw-Hill, pp. 8-13, 139-144.
- Wiegers, K.E. (1996). *Creating a Software Engineering Culture*. New York: Dorset House.
- Yourdon, E. (1989). *Structured Walkthroughs*. Englewood Cliffs, NJ: Prentice Hall.

Acknowledgements: The author would like to acknowledge the helpful comments provided by Dr. Richard Lesniak, University at Buffalo, State University of New York. This case was developed with support from The Pew Charitable Trusts as part of a Case Studies in Science Workshop held at the University at Buffalo, State University of New York, on May 21-25, 2001.

Date Posted: 08/29/02 nas