

Parachuting Software Engineering Practices into the Hostile Environment of 4th Year Final Term

M. R. Smith,

*Department of Electrical and Computer Engineering,
University of Calgary, Calgary, Alberta, Canada T2N 1N4
Email: smith @ enl.ucalgary.ca*

Abstract

Two years ago an elective 4th year microprocessor course taught to a small class of 35 was switched to become a compulsory 3rd year course taught to 150. Student grades became bimodal, either D+/C- or A-/A, and the instructor's teaching evaluation plummeted. A rework of the program the following year made no significant difference. To overcome these problem, elements from Humphrey's Personal Software Process were parachuted into the undergraduate program. Modified versions of the earned value analysis procedure and review sheets were developed, together with a defect analysis approach directed towards improving the manner in which laboratories were handled.

Introduction

Ever since the Computer Engineering Minor program was introduced in 1981, both courses in the microprocessor stream at the University of Calgary's Electrical and Computer Engineering program have been taught in the fourth year to a small class. The lectures were tailored so that the students were prepared for the six practical laboratories in the courses. The classes were enjoyed by both students and instructor with high marks obtained by the students and reasonable teaching evaluations for the instructor.

Two years ago, one of the courses was switched from a 4th year elective to a compulsory course. The class size jumped from 35 to a mixture of 159 students from 3rd year, 4th year and those returning from industrial internship. The change did not go well. Practical laboratories, the main focus of the course, were frequently left incomplete. Students marks became bimodal (D+/C- or A-/A), and both instructor evaluation and personal satisfaction in teaching plummeted. The following year, with significant course changes made, went no better.

At the same time as the undergraduate curriculum changes were introduced, a new M. Sc. with a specialization in Software Engineering was funded. One element of this new graduate program was the introduction of courses formatted around Humphrey's Personal Software Process (PSP). The response of students to the practical elements of PSP suggested that the introduction of these concepts would be useful in the undergraduate microprocessor courses.

Given our current environment relating to undergraduate software engineering practices, existing course load and accreditation requirements, it was only possible to introduce certain practical PSP elements. This paper discusses the reasoning behind the elements selected -- earned value analysis, code review and the introduction of defect analysis in the laboratories.

The new software practices have been dropped (parachuted) onto a small group of students who are without significant formal software engineering training. Although these students have been exposed to the concepts of "good programming practices", the majority see continued hacking as giving them a theoretical, rather than practical, disadvantage. Place these students in the final term of 4th year just months from graduation, and you have a fairly unreceptive audience expected to be passively, if not actively, hostile as suggested in the paper title!

Modified Earned Value Analysis

Students in a general compulsory course have different expectations to those in an elective course. Many will be taking the course because it is required of them rather than wanted. Many are expecting to do poorly with less access to the instructor for help as compulsory courses are typically of a larger size.

Further complicating the issue is the instructor's own personality, teaching style and expectations. For example, I teach my microprocessor course with an unusual emphasis on interaction with C/C++ software in addition to the normal hardware interfacing. A dramatic shift in student attitude is needed to switch from an introductory C/C++ course to the more design oriented microprocessor laboratories. Further, I don't inflate my marks, I quiz heavily and I actually expect the students not to do well during the first part of the course. In this situation many of the students don't properly recognize their likelihood of success until late in the term.

In previous years, compensation for this student attitude has been made by indicating an expected final average class mark of at least B-, despite low early quiz marks. Unfortunately, such announcements are quickly forgotten. Repeating them after the students have received the anticipated low marks is perceived as an instructor's excuse

after scoring a home goal. In addition, course evaluation on the instructor typically occurs before the students appreciate that they will be graduating with high marks in this course.

Humphrey [1] provides a practical mechanism, earned value analysis, for tracking successful completion of a project. A variant, earned grade analysis, was developed to overcome the students' concern over marks. Many of the concepts translated directly over between the two applications. PSP tasks become the course components, and

the task effort can be identified with the component weighting in the final course mark. In the PSP, the project manager and developer are the same. In the variant, the manager is the professor providing deadlines for quizzes *etc.* The developer becomes the student who receives a percentage of the final course mark on completion of each component. These elements of Humphrey's earned value analysis can be recognized in the Earned Grade Analysis spreadsheet shown in Figure 1.

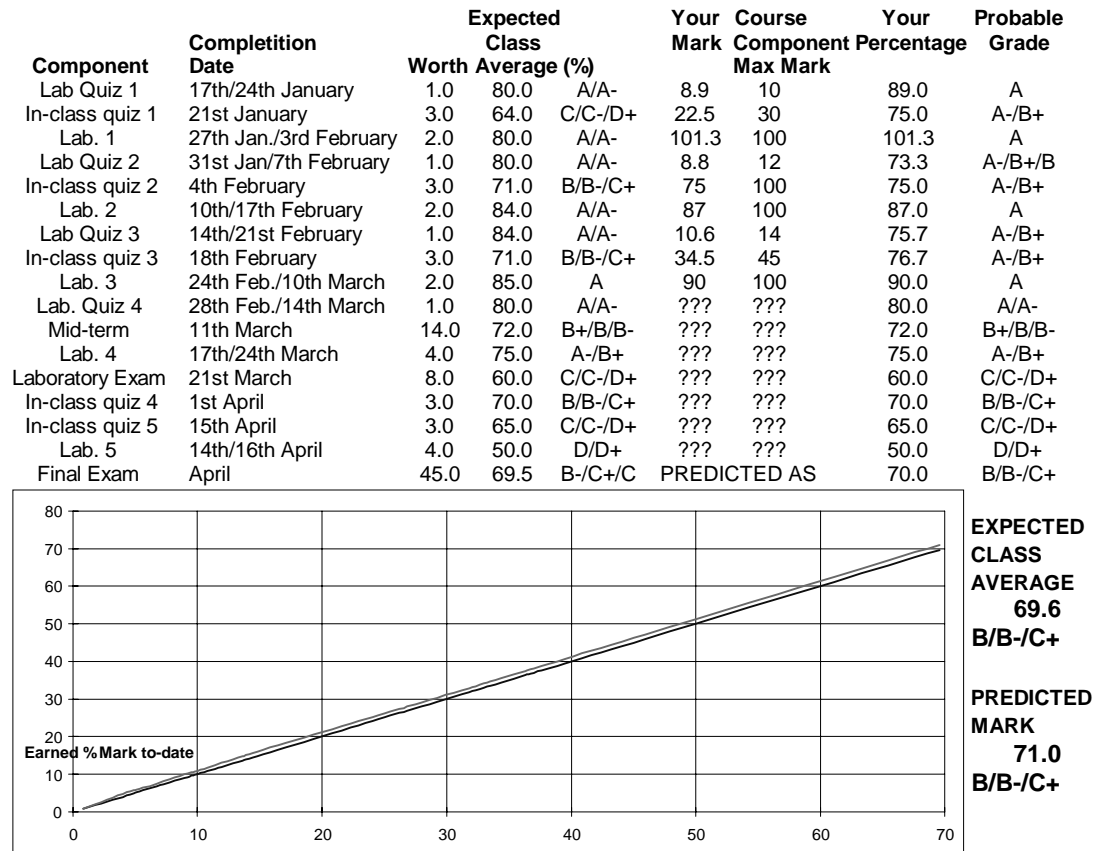


Figure 1: -- Screen printout from an Excel spreadsheet used to track earned grades during a term.

Humphrey tracks whether a project is on time by comparing actual and planned time of completion for each component. No reward is given for undeveloped components. Adopting an equivalent approach to track a student's marks through a 3 month course would not provide the required progress reassurance. The total marks awarded for completed components in the early part of the course would appear to the student as an earned D grade at best. To overcome this discouraging aspect the student is initially awarded the expected class average for each course component. As students enter their own scores in the cells marked ??? on the completion of a component, predictions

of the final exam and course marks are made based on known marks and expected class averages.

There are also relationships between the course mark analysis and size estimation techniques in PSP. Poor size estimation, with a large error, occurs when a project is considered as a single entity. Estimating the project as a series of smaller tasks leads to a lower overall performance error, even if the individual component errors remain high. The first concept corresponds to a student thinking that the poor mark in the latest quiz directly translates into a flunking final grade. By contrast, instructors recognize that the final mark is little affected by any individual component.

An earned value analysis carried on during the final exam is also useful. Students are pre-warned that they are expected to mark, in their minds, each exam question completed. They record their predicted mark on the exam's front cover. If their prediction proves correct within 10%, they are rewarded a bonus mark to compensate for the "time spent on the analysis". This approach had students coming out of the final exam happier as they had a better idea of their own performance based on the whole exam rather than a remembered poor mark on one question. It also provided an unexpected bonus for the instructor. For some questions it became clear that many students expected high marks but actually received low marks. This clearly indicates a course component understood in concept but without sufficient comprehension to actually apply the material. Such an indication immediately provides concrete direction for course changes.

The advantage of introducing formal software engineering terminology during Laboratory Development.

An incremental technique to the development of hardware and software during undergraduate laboratories should be encouraged as an incremental approach minimizes risk. Unfortunately, many students see incremental development as offering no practical advantage over their normal hacking approach of design during coding and testing of the complete project. Humphrey points out that it is necessary for students to show by their own personal experience that taking the wrong approach is to their own detriment. However, given the other material to be covered in a non-software engineering course, the time needed to gain this necessary experience is not available.

The first step to proactively shorten the process is to show the students that the various prototypes of an incremental laboratory approach form a SPIRAL life cycle. This compares to the WATERFALL life cycle of the standard hacking approach. With some basic software engineering terms introduced, the next stage is to bring in an industrial speaker willing to discuss the details of software disasters, the gorier the better. This speaker can be asked to stress the trend of local industry towards introducing software engineering practices. This hits home early in a final term 4th year course as the students in the process of attending interviews. They realize that knowing "something about software engineering" may provide the needed edge in getting a job.

Students are more likely shift their own software development process when they realize that the time they personally spend to fix a "defect discovered in a phase" is significantly less than the time to fix a "defect discovered

later". Although students without formal software engineering training may be familiar with the phases associated with a project's development, they have difficulty in distinguishing between them without practice. In addition, laboratory prototypes are so small in scope that any defect analysis associated with individual phases is meaningless. The concept of "defect discovered within a phase" was therefore replaced with the more relevant "defects discovered within a prototype".

In addition, a change of terminology was also introduced. Students were said to "make mistakes" rather than "discover defects". Mistakes were classified using the more emotive terms from the Motorola University software engineering course books [2]. Mistakes made and discovered during a particular prototype were classified as "Errors", suggesting a mild problem that is easily fixed. Mistakes made during one prototype and not discovered till later were "Defects", hinting at a more serious difficulty.

In the course of a laboratory, students quickly recognize that the time consumed with fixing errors is significantly less than that required to fix defects, even when detailed metrics are not gathered. With the basic software engineering concept in place, it is straight forward to show the relationship between problems with the student's process and those of industry. This is all that is needed to convince students that having a process that introduces errors rather than defects is to their advantage. This approach may not be exactly what Humphrey advocates. However it is a significant step in the correct direction of making the students comfortable with the possible advantages of practicing formal software engineering.

Application of the ARMIE review sheet

The next problem was to develop a technique that took advantage of the students recognizing that it was better to have an error than a defect. Parachuting process management techniques into the middle of a non-software engineering course does not provide a great deal of opportunity to develop a personal approach. However, as an instructor you are already familiar with what will cause problems during exams and laboratories from your own personal experience. Using this information, and perhaps with the help of a senior student who has already taken the course, you can develop an ARMIE review sheet. ARMIE is an acronym for "Avoiding Removal of Marks In Exams".

The ARMIE sheet was used with an approach that follows the intent of Humphrey's PSP. During the first laboratories, students mark the ARMIE sheet according to whether a mistake was recognized during their prototype in which it was introduced or later. Combined with mistakes from the in-class quizzes, it was hoped that the students will recognize that they are personally making certain classes of

errors or defects more frequently than others. This is a context-relevant, accelerated variant of the PSP mid-term report.

As the term progresses, the students are encouraged to use their own ARMIE sheet during the laboratories as part of a review process to recognize errors rather than leave behind the more time consuming defects. To ensure that the students made a deliberate effort to use the ARMIE sheet, their use in quizzes requires that the updated sheet be handed to the instructor the day before the exam.

G. Q. M. -- Goal, Question and Metric

Humphrey [1] makes the point that process changes serve little purpose without a goal in mind. Obvious goals are to provide a better experience for both students and instructor together with a significantly improved teaching evaluation. Questions and metrics are associated with whether the parachuting of PSP principles into an undergraduate course will serve any useful purpose in achieving the desired goals. Unfortunately, given the time constraints on submitting this paper, the metrics really needed to evaluate success are not yet available. Specifically, our administration enforces a slow pace by insisting that course comments are not made released until all possible student appeals have been settled. When this paper is presented at FIE '97, it will be possible to offer more than just the following simple, qualitative observations.

Students clearly understood the differences between "defects" and "errors". By the end of term, they were using the concepts to make jokes. My mistakes were verbally classified according to whether they had been made in this or an earlier lecture. One salesman was also made very aware of the difference when problems arose with his product demonstration. However a practical question on the final exam that dealt with the recognition and yield analysis of defects and errors in a series of simplified laboratory prototypes was answered poorly. This was a question that the students identified as one where they felt they had performed well. Clearly more practice on the use of these concepts is required.

The use of the ARMIE review sheets to identify basic errors worked well for the first four laboratories. By the final laboratories, the basic syntax errors were no longer perceived by the students as causing a problem and the sheets were no longer used. However, I observed that the same mistakes were still being made.

The ARMIE review sheet was also intended to allow the students to review their quiz answers for errors made. Since the sheets identified the student's own mistakes, rather than identified solutions, they are not a cheat sheet. With the exception of the final exam, the sheets were not used as

anticipated. Attempting all question on one of my exams is not necessary to receive an A grade. However, the students tend to hurry through all questions to maximize their marks, leaving little time to check for mistakes using the ARMIE review sheet. However, students report using the ARMIE sheet to review possible mistakes prior to the quiz, still a valid SE usage.

I have kept track of how many ARMIE sheets were voluntarily handed in prior to each quiz in an attempt to see how many students considered them useful. The rate dropped off considerably towards the end of term. However, when I introduced a new format for the ARMIE sheet for use in the final exam, well over 50% of the class took the time to move the numbers from the old sheet to the new. I also found that the laboratories went more smoothly this term with the students keeping track of their errors. However whether this is due to smaller numbers (16 per lab rather than 48), students in a second course in the stream or the ARMIE sheet itself remains to be determined.

I made each student hand in an updated earned grade analysis just prior to my own evaluation so that they would realize the success that they were having. Whether this strategy was successful will be determined when the evaluation comments are returned. It was clear from comments that some students were keeping close track of their grades with the EGA sheet. At the end of the term, they were able to identify just how well they had to do in the final exam to achieve the letter grade they felt they deserved. In addition, it was easier for me as an instructor to provide counseling for the student when they came in with a poor quiz and an updated EGA printout.

Conclusion

Although the final analysis is not completed, it would appear that parachuting several software engineering practices from Humphrey's Personal Software Process into the middle of a final year microprocessor course solved many problems. Students concerns from previous courses revolved around marks and laboratories. A modification of Humphrey's Earned Value Analysis process solved difficulties associated with perceived progress during the course and during exams. Introduction of software engineering terminology associated with "defects in and out of phase" into a SPIRAL lifetime model for laboratories convinced the students of the advantages of actually following an incremental development approach rather than just given it lip-service. This led to the successful introduction of a code review sheet for use during quizzes and laboratories to cut down commonly repeated errors.

The techniques have been very successful with a small class in an 4th year elective microprocessor course. By the time the conference presentation is made in November 1997,

a compulsory, introductory microprocessor course with a large student population will be well under way. It should be clear by then whether the solutions work with the younger, less experienced class.

When the final conference proceedings version of this paper was submitted, administration had not forwarded the necessary information. Details will be made available at the following web site

www.enel.ucalgary.ca/People/Smith/ENEL515/fie97.htm

References

- [1] W. Humphrey, "A Discipline for Software Engineering", Addison Wesley, Don Mills, Ontario, 1995.
- [2] Software Engineering Management Certificate program offered by the Faculty of Continuing Education, University of Calgary in conjunction with Motorola University, 1996.