

Teaching software development: a case history

C.S. Johnson

*School of Computing Sciences, University of Technology, Sydney,
P.O. Box 123, NSW 2007, Australia*

Abstract

This paper discusses the teaching of a final year undergraduate software development subject. It describes how the subject was taught and the experience gained over three years of its existence. The subject required the students to work in teams of eight. Their goal was to produce software to meet a given specification to a commercial robust standard. The student teams had no tutorials but reported to the tutor on a weekly basis in a formal management meeting. The key motivational factor to the quality of the work was that project assessment was performed by running the final system, the marking to stop on any system failure.

The most interesting aspect was that students were not given the specification until half way through the semester. They critiqued the previous year's system, so they knew the type of product they were to produce. They spent the first seven weeks preparing JCL, reusable code modules and skeleton module frameworks so that on receiving the specification they could proceed productively. The opportunity for creativity was provided for the students by not including screen layouts in the specification, just the required fields. This meant that each group could produce an "original" system.

There are numerous benefits that other academics can use in similar subjects and this paper summarises the key ideas that are easily applicable. It should be

added that the highest graded system of the second year was adopted by the university and performed with no errors.

1.0 Introduction

This paper describes the teaching of software engineering in the form of a final year subject for students in an undergraduate course. Although this experience was gained from 1986 through 1988 the methodology used is as relevant today as it was then, especially as the underlying theme was "re-use" of code.

The experience gained should be of use for academics that supervise team-based coding subjects. Many of the ideas can also be incorporated into other subjects to improve their effectiveness.

The subject was called Information Systems Implementation and was taught at the Department (now School) of Information Systems in the Faculty of Commerce at the University of New South Wales, Australia. The subject was undertaken in the students' last semester of a three-year Bachelor of Commerce (Information Systems). The subject was compulsory for the Bachelor of Commerce students but could also be undertaken by Computing Science students. There was no thesis or project subject requirement for the degree so to large extent this subject had to incorporate some of the goals of a project-like subject.

2.0 The Problem

Students in the B.Comm (Info. Sys.) degree in the Department of Information Systems completed a three-year degree the main thrust of which was in Information Systems. Thus hardware subjects, even programming, were not taught. The students could pick up these subjects as electives in the Computing Science stream. With no industrial year or training and a dearth of practical experience, the students were somewhat behind their fellow students in other computing degrees in Sydney as far as obtaining employment was concerned. Thus the main goal of the Implementation subject was to give the students, as much practical experience as possible in an interesting and challenging manner, bearing in mind their lack of experience.

Resources available consisted of a University VAX 11 /780 and ANSI COBOL 80. The student numbers ranged from approximately 50 to 100 over the three years. This subject tied together much of what the students had learnt. In previous years the subject had been a basic coding exercise for a fairly common straight-forward system, for example a Fixed Asset system.

3.0 Teaching Objectives

Recognising the background of the students and having had a number of years of commercial experience, I wished to prepare students for the realities of the commercial world. In the process give the students a feeling of pride and accomplishment. My personal objectives were for students to write robust software, to a commercial standard of operation, in a team environment and to produce a total system solution. The term "total system solution" in this context meant that the software produced must be a completed deliverable product, i.e. run from a login, via menus, with users unable to escape to a command level, set up in an appropriate directory structure, production, development, test etc; with JCL for data backup, total system compile etc. The only aspect left out was, of course, system documentation, although the students had to produce a User's Manual, to experience the documentation task. This documentation decision was taken due to workload and time constraints.

To achieve these objectives efficiently and effectively, the underlying principles of code re-use, preparation and interface design were specifically focussed. The published objectives of the subject were :

- 1. Systems implementation involving program design, coding and testing of a practical system in COBOL.*
- 2. Participation in, and control of, project teams both large and small.*
- 3. Systems design and management concepts.*

Two other objectives were important to the success of this subject. The first was to allow the students some creativity. This was achieved by giving the students a specification that did not include any screen designs. The fields on the screen were minutely specified (see Appendix A) but no screen designs were given. Thus each group of students could design a system with a unique appearance. Reports were specified in even vaguer terms. The second aspect was to create a non-confrontational relationship between staff and students. Thus staff were to help students to do the best possible job. Thus the teaching staff acted in the role of a typical commercial project manager. An example of this was that a noticeboard was set up where new information could be relayed to students, general questions answered, etc.

4.0 The System Specifications

A key component was the system described in the specification given to students. Thought was given to providing specifications that were unusual and interesting. To this end a prison information system (SHIPS - Short Bay Information on Prisoners System) and a centralised farm information system

(FARM - Farm Area Resource Management) were developed. A tutorial allocation system (TASS - Tutorial Allocation System) was developed in the middle year in response to the University's inability to provide the author with student numbers before the start of the subject.

The specifications were produced in a transaction-based format using indexed-sequential files. Multiple-keyed access to a file was incorporated. Each system consisted of two or three dependent files to provide integrity considerations. The operations of 'add', 'change', 'delete' and 'enquire' were performed on each file. There were approximately sixteen online applications together with eight reports, giving a system of appropriate depth.

Additionally each system had a real life complexity, i.e. SHIPS had a permissions and an audit file aspect; TASS had separate logins and different capabilities for staff and students; FARM had a three-level hierarchical entity relation, with three types of login access with different permissions. The purpose of these complexities was to provide students with a conceptual complex design task that was integral to and permeated the whole system.

The specifications were approximately forty pages in length and contained unintentional errors made by the author. These were left in (and also in subsequent re-use of the specifications by the author in other subjects) since they provided excellent feedback on the students' understanding of the specifications; i.e. a group's progress could be measured as to when these errors were detected.

5.0 Plan

The distinguishing feature of the subject plan was that the specification was not given to the students until week seven, whence they had six weeks to complete it. The thirteenth week was kept spare, and the students gave a presentation in the final week. The spare week was given as an extension (generally in week 10/11). This was because the author wanted the students to complete the work, the students wanted to complete the work, and usually there were a few days of system down time. Students were notified that system down time was not an excuse, that is, it had to be planned for, but as mentioned earlier, the subject was not meant to be confrontational.

The students were formed into teams of eight, each student had to write two online programs and a report. In the first six weeks the students formed into groups of two to investigate one of four areas in preparation for the second six weeks. They were :

1. *Development of the COBOL Libraries, Directory structure, and JCL.*
2. *Development of the Screen layouts*
3. *Report layouts and the Report Program Skeleton.*
4. *Development of File Input/Output routines and File Descriptions.*

Not only did the students have to give a report to the rest of the group on their topic, they also had to set up code etc. for the group's subsequent use to achieve these tasks. Each student also had to write a simple online application, from which the group would pick the "best" to provide the code skeleton for the actual system, i.e re-use of code. These areas were also provided with challenges; the students were given two logins for eight students, there were text-based windowing utilities available, there was a report writer, and there was an alternative key facility, also, all file accesses had to be tested for success and appropriate action taken, i.e. gracefully close down the system on failure.

In a system of this magnitude students often have a problem visualising what they will produce until just before the end, inevitably too late. To minimise this problem, the students' first task was to critique the previous year's solutions. This was in the form of an evaluation of the user interface. The students had to run at least four of the previous years systems. A drawback to this method, that each year the students sought to improve on what they had seen. The last year's groups implemented impressive complex text-based windowing systems utilising supplied VMS routines.

The subject had eight weeks of lectures, which were mainly for revision and covered the topics of :

- Week 1 1. Project Group Responsibilities and Leadership, 2. Use of Manuals*
- Week 2 1. Project Planning, 2. Program Design*
- Week 3 1. COBOL Skeleton Programs, 2. COBOL Program Standards,*
3. COBOL Version Control
- Week 4 1. Job Control Language, 2. Directory Structures, 3. Screen Design,*
4. Report Layouts
- Week 5 1. Project Control, 2. COBOL Program Degradation, 3. Libraries*
- Week 6 1. Program Testing (guest lecturer)*
- Week 7 1. COBOL Debugging Techniques, 2. Testing Strategies, 3. Walkthroughs*
- Week 8 1. Programming Environments, 2. COBOL/System Utilities*

At any stage a group could ask the lecturer to test their software. In practice, this consisted of two types of tests, the first running a hand across the keyboard; the second was to test some fine feature, e.g. enter 29/2/87. This was only requested once or twice each time the subject was run, as the information was passed between students that testing was crucial.

6.0 Assessment

The system was worth sixty percent and was evaluated by logging in and running the system. If the system crashed, the evaluation was stopped and only marked for what had been tested. To facilitate this, a set of test data was given out one week before the due date. There was to be only this set of data in the system (students then had to clear the files and would often discover problems, e.g. in SHIPS the security file gave permissions to alter itself!). With all student systems having the same data, evaluation was greatly simplified. The systems were assessed on meeting the specifications, consistency of the user interface across the system, ease of use and not crashing.

An assessment of each member in the group, including themselves, was conducted by the students twice during the development of the system, with the second assessment having the ability to influence an individual's marks from plus five to minus twenty of the group mark. Even when applied in extreme cases this caused no problem amongst the students.

Included in the project mark were their user manuals (some innovative students produced these on video tape, quite effectively). A final presentation was required covering *a Management Presentation of approximately 15 minutes on the system and the problems and highlights of its development*. This provided excellent feedback since students generally drew attention to shortcomings in the facilities or information provided by the lecturer.

7.0 Results

The subject ran for three years with numbers increasing from 55 to 77 to 99 as more students from Computing Science chose the subject. All groups produced acceptable systems, the top three producing on average, commercial grade software. Evidence that supports this contention was that the best TASS system was used by the Faculty of Commerce for three years. The first year of operation only required some additional reports to be written (and provided paid casual work for the group). The Manager from the Information Systems Department from Long Bay Correction Centre in NSW favourably evaluated SHIPS (he spent about two hours testing two systems) and made the comment "if we had not just implemented a system I would use one of these". Student feedback from surveys was positive. Many students obtained distinctions (75%+).

The students met all the objectives, both given and hidden, and finished with an apparently excellent understanding of a "total systems solution". As

mentioned above, the quality of the user interface improved significantly each year but this led to one of the drawbacks in the subject.

This drawback was that students pursued this subject to the detriment of their other subjects, with many of them working right through the last weekend (machine response was very poor at this stage, which did not help). As each year set itself higher standards, more work was obviously entailed. Although much of this was done early in the semester, it still added to the total workload. Another area for improvement was in the quality of the code, which could have been better, but marks were not assigned to code, mainly because of the assessment task. The complete code with JCL etc., was handed in and was of the order of two inches thick.

The final event in the subject was a party thrown by the lecturer for the students, where the lecturer provided a case of champagne, in appreciation of the students' effort.

8.0 Conclusions

The following conclusions can be drawn ;

1. Students can write commercial-grade software given the right environment.
2. The critique of previous projects for students undertaking large projects allows them to assess the scope of the solution and understand very early what is required of them. This has been successfully applied in a subsequent subject taught by the author. The only drawback is that standards keep rising.
3. Providing a creative aspect provides ownership and motivation for the student.
4. Students can achieve far more than is often expected, namely the completion of these systems in six/seven weeks.
5. The concepts of HCI, SQA (in particular Configuration Management, walkthroughs, reviews), and code re-use, while common practice today, but were not given this recognition at the time of this project, provided the foundations for the development of quality software applications.



9.0 Recommendations

Although this teaching experience is a number of years old, the lessons learnt can be applied today to the teaching of software engineering and can be applied in different forms across a broad range of subjects in software engineering. Bringing together the disciplines of OOT, HCI and SQA should be the goal of any software engineering course.

This experience demonstrates that with thought and planning by academic staff, students can appreciate and produce robust software within deadlines while at the same time developing their creativity and having fun.



Appendix A

Field Name	Size	Type	Range	Validation	Notes & Function
AGE	2	N	18-99	MUST BE PRESENT	
				(an intentional error)	
BUILD	1	A	L,M,H	MUST BE PRESENT	SPECIFIES Build
					OF PRISONER
					L - LIGHT
					M - MEDIUM
					H - HEAVY
HEIGHT	1	A	S,M,T	MUST BE PRESENT	Specifies Height
					OF PRISONER
					S - SHORT
					M - MEDIUM
					T - TALL
DATE OF RECORD	6	N	TODAYS DATE	NONE	ASSIGNED BY
					SYSTEM, Clears
					DEPART DATE,
					ARRIVAL DATE
CONVICTION	2X30	X	FREE	MUST BE PRESENT	DETAILS OF
			FORMAT		OFFENCE
CONVICTION DATE	6	N	DATE < TODAY	VALID DATE	
SENTENCE YEARS	2	N	10-50	OPTIONAL	ONE OF THESE
					Fields must be
					GREATER than
SENTENCE MONTHS	2	N	10-12	OPTIONAL	ZERO
SENTENCE DAYS	2	N	10-31	OPTIONAL	
INIT.PAROLE YEARS	2	N	10-50	OPTIONAL	ONE OF THESE
					FIELDS must be
					GREATER than
INIT.PAROLE MONTHS	2	N	10-12	OPTIONAL	ZERO, must be
					LESS THAN
					SENTENCE
INIT.PAROLE DAYS	2	N	10-31	OPTIONAL	