

# Tight Spirals and Industry Clients: The Modern SE Education Experience

James M. Hogan, Glenn Smith, Richard Thomas

School of Software Engineering and Data Communications  
Queensland University Technology  
GPO Box 2434, Brisbane 4001, Queensland.

{j.hogan, gp.smith, r.thomas}@qut.edu.au

## Abstract

Modern software engineering education is driven by an expectation that industry best practice and state of the art software technologies should not wait until graduation, but should appear early in the undergraduate curriculum. Moreover, students in our programmes have a strong, and well-justified belief that industrial strength project work is well-regarded by potential employers, and have had little difficulty expressing their preference for more relevant courses.

In response, we have significantly enhanced the industry linkages and professional focus of our SE programme through a series of initiatives over a four year period. These initiatives have included: the introduction of a more modern development process employing multiple iterations; a novel approach to the first software engineering unit in which the students must extend an existing (phase 1) software system; the involvement of a live industry partner in the advanced software engineering unit, with regular and candid feedback from the company staff; and the provision of detailed support materials for version control and automated unit testing as part of our process infrastructure. This paper details our experiences during this period, the rationale for our process and course structure, and discusses the feedback on our approaches from students and local employers.

*Keywords:* Software Engineering Education, Iterative Development Processes, Problem Based Learning.

## 1 Introduction

Software engineering projects provided to students have long suffered from their reliance upon the fictitious client, the bogus business case and some remarkable variations in scope and domain. Such problems are exacerbated by the strictly finite supply of project ideas in the mind of the university teacher. The principal difficulty with these projects is to provide students with sufficient motivation that they become enthusiastic about its completion. At its heart, this problem is most readily solved through the introduction of a live industry client

with real, rather than simulated concerns, and problem domains which reflect the priorities of the industry.

Yet the commitment to a live client cannot be undertaken lightly, and may involve unacceptable risks on both sides if novice developers are involved. Rather, it is necessary to develop explicitly the range of skills that support the professional team over a series of stages, the industry client experience being the culmination of the education we provide, and the beginning of the transition to the workplace. Yet, if student interest is to be maintained, even our preparatory work must have an industrial edge, providing a number of challenges for the teaching staff.

In this paper, we examine the changes that have taken place in our curriculum as a result of this focus, and our experiences in making the ideas work in practice. We begin, in section 3, with an historical review of our software engineering programmes, leading naturally to discussion of the driving forces for these changes (section 4). In section 5 we examine the most important prerequisite for an industry-oriented curriculum — the development of an industry standard process and related infrastructure that may be made available to students. The specialisation of this process to teaching novice and advanced SE students is considered in section 6, and our experiences are related in more detail in section 7. We conclude by summarising our initiatives, and some possible extensions to the programme.

## 2 Historical Perspective

Software engineering and process material has been taught on a significant scale at our university since 1986 (Thomas, Semeczko, Morarji, and Mohay 1995), with the team project a core feature of our approach. As is well documented in the literature, the use of a team project in a software engineering unit helps students to understand clearly the purpose and importance of a software engineering process (Tomayko 1987, Ford 1994, Bagert et al 1999).

The Faculty's software engineering curriculum has undergone four major reviews:

- The initial version of the unit made use of a team project but provided little process guidance to students, aside from the theory presented in lectures.
- The first review focused on introducing a disciplined process structure based on international standards, reflecting a standard waterfall approach to development.

- The second review focused on the teaching strategies used. The single SE unit was split into two units, Software Engineering Principles (SEP) and Advanced Programming Laboratory (APL). SEP focused on theory and its practical work was not project based. APL was assessed entirely by project, with short theory lectures to support the process.
- The third review re-introduced a project into SEP. It was found that students did not appreciate the process theory if they did not apply it simultaneously in practice. APL was maintained so that the project could be as small as possible in SEP and also to provide the benefit of giving students a chance to learn from their mistakes in their first project. At this review the SE process material was given a minor update and a set of supporting material was introduced.
- In the latest review, a new process was developed for use in software engineering related units at our university. Industry-based projects and aspects of Problem Based Learning (PBL) have been introduced into APL. APL typically has enrolments of one hundred or more students who work in teams of five to seven. The project makes up 100% of the assessment for the unit which lasts for the entire semester, making it a significant size for a student project.

### 3 Motivation for Change

Much of the motivation for this revision of our software engineering educational framework has been considered in the introduction. In practice, however, the goal of greater industry relevance and increased student motivation can manifest itself in a variety of forms.

Central to our approach - as it provides some indication of baseline quality to potential industry partners while allowing greater student involvement in the development process - is the provision of a self-contained modern software engineering process and associated infrastructure. The Real World Software Process (RWSP) embodies key aspects of the Agile methodologies (Hogan, Thomas, and Smith 2002), and the combination of flexibility and industry relevance has led to markedly increased interest from the students.

.Industry-based projects have students working with a real client who has a significant interest in the results of the project. Developing software that is of benefit to a real client is in itself a strong motivator (Tomayko 1987), as is the use of leading-edge commercial tools in the projects. (And, the possibility of future employment with the client cannot be downplayed as an additional factor.)

However, it was quickly recognised that a revision to the SE process material may not of itself be sufficient to support an externally driven project, due to the ongoing need for team members to seek out information on new

technologies and to communicate this within the team, and the need to manage successfully the team environment itself. For this reason, traditional delivery mechanisms were supplemented or replaced altogether by approaches founded upon problem based learning.

These issues are considered further in the subsequent sections.

#### 3.1 Limitations and responses

The SE process previously employed in our programme allowed some incremental development, but was nonetheless seen by students to be more heavy-weight and traditional compared to RWSP. The earlier process placed more emphasis on up-front planning and some students tended to defeat the incremental nature of the process and would follow a more linear path through development. RWSP is similar to other Agile methods in that it makes the incremental development model much more visible, and markedly more difficult to subvert successfully.

In the previous versions of both SEP and APL students would work in small teams developing fictitious projects from scratch. The intention was that APL would allow students to learn from their mistakes in SEP and apply the process more rigorously on a larger project. However, student reports suggested that they did not see the benefit of doing a second project; indeed, some complained that “they did not learn anything new”, leading to the view that the APL capstone was insufficiently distinguished from the project novitiate of SEP. To some degree, this may again be the result of the limited set of ideas which emerge from the academic project creator.

The new versions of SEP and APL now provide significant differences in their projects. SEP still uses a small project created by the teaching staff, but the selection is consciously more topical, with recent projects involving web-based voting; on-line business information systems - providing analysis of exchange rates and securities prices; genomic sequence analysis and plagiarism detection. The approach is particularly novel in that students start the project at the end of the first phase of development. Thus, each team is provided with a core working system and supporting documentation, which must be extended across two full iterations of the development process. As described above, APL uses a larger industry-based project that students start from scratch. These differences mean that students now see benefits of a second project, and are markedly more professional as they approach the more challenging project in the second unit.

#### 3.2 Industry trends

The term “software engineering” was coined flippantly at a NATO conference in the 1960s (Naur and Randell 1969). Since that time, substantial efforts have been made in both industry and academia to try to bring an engineering-like discipline to bear upon software development. More formal software development methodologies began appearing in the 1970s, with

processes and process improvement emerging as a significant focus in the early 1980s. In essence, this period saw the creation of a sustainable discipline of software engineering.

Many processes have been published over the intervening twenty years, and aside from the recent disputation over process weight-loss, among the most significant trends is the shift from rigid prescription of process steps toward the provision of more general process frameworks that are meant to be adapted to project needs.

In this light, the current focus on Agile methodologies is best seen as a series of guiding principles to be adapted to the team size and project scope. Nevertheless, there are some characteristics common to most successful applications of these approaches. Agile methodologies are championed as ideal for business oriented software: where the majority of teams are small, requirements are unclear, and the underlying business processes may rapidly change.

In light of the nature of much of the software development done in the region, and the likelihood of a strong match between student project teams and those characteristic of the agile movement, it was decided that our in-house process should embrace the principles of light-weight software engineering processes while building upon the strengths of our previous process and staff expertise.

### 3.3 Benefits

It was quickly realised that the heavy-weight nature and significant up-front planning of the previous software engineering process would not support interaction with an industry client in an academic setting. Students would require more frequent feedback to complete a commercial project within the time frame of a university semester, and the RWSP was designed to suit these requirements. It was decided that students would complete multiple cycles of the process, thus allowing for more regular feedback.

The use of PBL was seen as a more suitable delivery mechanism for the domain knowledge and technological background required for the industry project. The completion of multiple cycles of RWSP also provides for student reflection on the software process and related deliverables. This is a critical component of the PBL approach, and supports the professional goal of continuous software process improvement.

The use of an industry client also introduced radical change in the underlying technologies used for each project. Thus students may not have sufficient knowledge of technologies they are required to use. The use of PBL encourages students to analyse the requirements for the problem at hand, and to further analyse the knowledge available within the team. The team must then identify and address deficiencies in knowledge. The teams are encouraged to distribute the task of discovering information, then to share the information within the team, leading to stronger learning experiences.

The use of PBL also more accurately reflects how similar projects would be undertaken within industry. Even though there are concessions made for the academic environment in which the project is set, the problem solving process is the same. This gives students better preparation for entering the industry environment.

### 3.4 Comparison with model curricula

In the past decade software engineering has moved from being an unusual programme offered by only a few universities to a mainstream component of information technology education (Ford 1991, Ford 1994, Bagert et al, 1999). Carrying over from computer science education has been the idea of a capstone project near the end of the educational programme. In a software engineering student project, the emphasis is typically on the process, rather than the product as is common in other discipline areas. All current model curricula for software engineering education include a capstone project as part of the structure (Chang et al 2001, Bagert et al 1999). In the present context, SEP is used to teach software engineering theory and APL provides the capstone project. Both units have a strong emphasis on the process that students follow.

The Joint IEEE-CS / ACM Task Force on the “Model Curricula for Computing” released a model curriculum for computer science in late 2001 (CC2001) (Chang et al 2001), and the corresponding model curriculum for a software engineering programme has recently been released<sup>1</sup>. Within the computer science model curricula, pure software engineering topics make up over ten percent of the core content. Software engineering is split into eight core topics:

- Software Design
- Using APIs
- Software Tools and Environments
- Software Processes
- Software Requirements and Specifications
- Software Validation
- Software Evolution
- Software Project Management

and four elective topics:

- Component-based Computing
- Formal Methods
- Software Reliability
- Specialised Systems Development

The revised software engineering major covers all eight of the core topics at least to the level suggested within the model curriculum, and covers all the elective topics as well. Many of the core topics are encountered for the first time prior to SEP, and the majority are extended to the relevant depth with this unit.

---

<sup>1</sup> See <http://sites.computer.org/ccse/>

Prior to starting APL students will have covered all the core topics at least to the suggested depth, as well as at least some of the electives.

This approach means that our students gain the benefits of undertaking a project while learning much of the core software engineering theory (SEP), while reinforcing this theory through the second iteration of the SEP project, and through the dual iterations of the more significant capstone project of APL.

A similar mapping is observed in the case of the Software Engineering Institute (SEI) “Guidelines for Software Engineering education” (Bagert et al 1999). These guidelines specify nine software engineering modules that contain the recommended content for a degree programme with a strong software engineering emphasis.

This comparison with the model curricula shows that our delineation of project work fits the expectations of modern software engineering education, while the dual project approach ensures that the use of a software engineering process is reinforced to a greater extent than these models suggest. Thus, students are more aware of the importance of a process model upon graduation from the programme, without significant loss of content with respect to the model curricula. This is achieved through the intensive nature of the degree programmes and through the use of PBL in key units. PBL puts the responsibility for learning on the students, with academic staff providing the appropriate environment to support the learning needed. This has meant that a unit like APL can concentrate on the process, while still providing a learning environment where students acquire additional material, such as component-based computing.

## 4 Process Description

The Real World Software Process is a lightweight software development process that is ideal for small-scale application development. There are four key phases that make up RWSP:

- **Phase Zero** — describing the process which takes place prior to the commencement of the formal software project. The key feature of this phase is that a project is proposed within the context of a business or organisational need. The project must have clear goals and well-defined success criteria.
- **Phase One** — incorporating requirements gathering, release planning and the initial functionality. This phase starts the software development part of the project. The outcomes of this phase are an initial model of the system and a working core product.
- **Phase N** — the generic, repeatable cycle in which the functionality of the system is incrementally extended and the requirements reviewed. Each increment of this phase reviews the result of the previous phase and plans and implements the next stage of the system.

- **Finalisation** — addressing issues of delivery and installation. This phase deals with the issues related to formal hand-over or turn-on, once acceptance tests have been passed.

Maintenance is viewed as an independent project that starts over at phase zero. These phases above are described in detail in (Hogan, Thomas, and Smith 2002). The process may also be viewed at:

<http://www.fit.qut.edu.au/~rwsp/>

## 5 Applying RWSP

While the RWSP may support a range of small-team project activities, there are nevertheless some important differences in emphasis in the manner in which it is applied in the introductory and advanced software engineering units. These are considered in turn in the following sections.

### 5.1 Software Engineering Principles

In Software Engineering Principles, students commence work at the end of phase one of the RWSP. Notwithstanding the planning effort undertaken by the academic coordinator, and the considerable attention to verisimilitude, the project is ultimately fictitious and so there is no real phase zero. A minimal version of the system is completed and unit tested, but provided to students prior to system test. Their principal task is to extend the system through two iterations of phase N. Students are provided with a complete set of documents for *phase zero*: the user requirements specification and release plan; and *phase one* of the project: the system requirements specification, the acceptance test plan and the design specification. These documents are used both as examples for the students and also as the starting point for their extensions to the system.

The first activity undertaken by the student teams is to extract the system source tree from their CVS repository, and to execute the phase 1 acceptance test plan. Unit tests for each of the key non-GUI classes are executed as part of the system build, and phase 1 system testing is undertaken for the first time by the student team. While the product is completed with considerable care – unit testing is extensive and automated – there is an expectation that some defects will be uncovered during this activity, thereby exposing the team to the twin benefits of familiarity with the quality assurance and maintenance processes and familiarity with the system itself.

Once students have had two weeks to become familiar with the system, the supplied material, and their team structure, they start their first extension of the system. They are provided with a selection of requirements within the release plan, and each team must work through phase N of the RWSP to capture the details of the new requirements, design the extension, and to implement and test the revised system. They have seven weeks to complete their first iteration of phase N.

Towards the end of this iteration students negotiate the requirements for another extension to the system. They then perform another cycle of phase N to implement the new extension. This final iteration is to be done in a four week period.

This approach to the project in SEP has four main benefits.

- Students are provided with an educational framework in which they can work to reinforce the concepts presented in the theory component of the unit.
- Students are exposed to a modern incremental software engineering process.
- The phase one artefacts provide students with a set of examples of the material that they need to produce in the project.
- By taking over the project at the end of phase one, students are exposed to some of the issues related to maintenance.

These initial benefits of the RWSP approach have been supplemented in recent semesters with a focus upon modern open source development tools – notably the build tool ant, the version control system CVS, and the java testing harness junit. Our process infrastructure has been extended to incorporate detailed tutorial guides for each of the latter two tools, and extensive demos are given during the early stages of the unit.

These activities equip novice developers with the skill set required for their work in a more demanding environment, such as the industry-based project of APL.

## 5.2 Advanced Programming Laboratory

In the Advanced Programming Laboratory unit, RWSP is applied in a standard fashion. Prior to the start of the semester, the academic coordinator liaises with the industry client to set up the project and effectively completes phase zero. The students start the project at phase one and complete two cycles (each a single increment of phase N). Finalisation is not formally undertaken; the students' completed products are simply demonstrated and delivered to the client. Only minor modification to the requirements documentation process is required in phase two. In this setting, the students discuss requirements with tutors; however it is assumed the client accepts whatever requirements are specified. This omits the negotiation and signoff process with the client, as this would be too time consuming, and is not possible when scheduling the project within a university semester.

The introduction of a problem based learning approach for the delivery of domain and technology-based information, prompted a shift from the use of traditional lectures to the use of industry experts as resources. Where the project required the use of technologies with which the students were unfamiliar, an industry expert was brought in to provide a seminar on the required technology. The student teams needed to prepare suitable

questions to make the best use of this resource. Providing students with access to appropriate domain knowledge and support for understanding new technologies is essential if the project is to be completed. However the students also needed to be encouraged to accept the shift in responsibility for acquiring information, and an explicit PBL strategy – building in part upon their experiences in earlier units – is an effective mechanism for ensuring that this transfer may take place.

Teams require some degree of ongoing access to the industry client, while respecting the need to limit the frequency of individual contact. Much of the necessary transfer may take place through client seminars, in which a client representative presents an outline of the system requirements, with more explicit detail emerging through open student questioning. To support this process, the students are provided with a requirements document prior to this seminar, allowing the teams to complete some preliminary investigation to identify aspects requiring further clarification. Subsequent access is filtered by the teaching staff, to ensure that only the most pressing requests for clarification are presented to the client staff.

Each team has an allocated tutor who acts on behalf of the client, and meets with the student teams weekly. To maintain some level of consistency of information across multiple tutors, all meetings are held in a single tutorial with the lecturer also being available. The lecturer assists tutors to answer difficult client oriented questions. The lecturer can then pass any unresolved questions onto the client. This hierarchy is required to support the large number of students in this unit without overburdening the client.

It was also shown to be critical to prototype the project using the laboratory facilities that the students will use. An important aspect of this prototyping is to use the same administration privileges as the students. The academic setting creates a difficult environment for development as while a standard industry project team would have access to administrator privileges on the development system, this is not always the case for students. Similar issues arise through the need to protect the work of student teams from their competitors. These restrictions must be considered and the project must be tested under these conditions.

## 6 Experience

This series of initiatives has resulted in three notable improvements in student software engineering project work: the prototype-first release to students in the initial stages of their first project; the involvement of an industry partner in the capstone project, and the design and implementation of a new lightweight development process with substantial tutorial support.

Providing students with a completed phase one on their first project has improved student confidence and reduced project start-up stress. Focus groups and project post-mortems have been conducted with successive groups of

students who have followed this project approach in SEP. The vast majority of students have appreciated being provided with the artefacts from the completed phase one of the project and then working with these artefacts. The students' comments have indicated that the benefit of this approach out-weighs the problems of having to understand artefacts that have been generated by someone else.

The selection of a suitable industry partner to provide a challenging and motivating project is critical to the success of this initiative. This has required that the academic coordinating staff be highly motivated to support the project and put in the preparation for the project prior to the start of each semester. The industry client must also be prepared to accept the outcome of a failed project. So far, each semester has produced successful projects from most teams, however it would be unwise to suggest that these are all industry strength solutions. In the projects so far the client – successively a major provider of financial services industry systems (four semesters) and the state government department of transport (3 semesters) - has been interested in the delivered student projects as a proof of concept for a new system. This has worked well in ensuring an appropriate balance between client interest and project risk – especially given the range of J2EE technologies which have underpinned these projects.

There has been a significant rise in student motivation, readily apparent in the quality of work produced since the introduction of industry projects. The most obvious improvement is in the reduction in the number of poor systems produced; even the weaker teams appear to be able to maintain sufficient motivation to get through the project successfully.

Based upon student reports it appears that key factor in increasing student motivation is the “real world” nature of the projects, the contact with industry and access to industry professionals, and professionally managed exposure to current technologies. Student and client feedback have both been very positive for the use of industry based projects, although some students have reported difficulties in managing the challenge of new technologies.

Yet the ongoing entrepreneurial challenge for the academic coordinator is to ensure that new ‘business’ is available in time for each new semester, and this has proven a demanding exercise in spite of the good will of our partner companies.

Introduction of RWSP has facilitated the previous two improvements, by providing an iterative process that supports the development models needed for each initiative. Using a light-weight, iterative process has also been beneficial in “selling” the process to students, by being able to leverage the industry interest in agile methods. The fact that some local commercial organisations use RWSP has also helped generate interest and motivation with students. Finally, the tutorial guidance provided by the process web-site and the

additional infrastructure provided have greatly assisted students in following the process.

## 7 Conclusion

This paper has reported on the software engineering project work initiative at our university. The dual project approach employed has proven to be a useful way to reinforce process ideas with students. It provides a way for students to practice on a project while they are learning the material, and then to “solidify” their knowledge in a capstone project. This provides students with the benefit of learning about software engineering principles and processes earlier in their programme, while still maintaining the benefit of a capstone project.

The experience section of this paper has highlighted the role of a new lightweight software engineering process in supporting the dual project approach. The provision of project artefacts has made the first software engineering project less stressful and more manageable for students. The introduction of an industry client has improved motivation in the capstone project. Together these two project units provide students with a solid grounding in software engineering processes and their application in development projects.

## 8 References

- Bagert, D. et al. (1999): Guidelines for software engineering education version 1.0, *Technical Report CMU/SEI-99-TR-032*.
- Chang, C. et al. (2001): *Computing Curricula*, <http://www.computer.org/education/cc2001/final/index.htm>.
- Ford, G. (1991): 1991 SEI report on graduate software engineering education, *Technical Report CMU/SEI-91-TR-002*.
- Ford, G. (1994): A progress report on undergraduate software engineering education, *Technical Report CMU/SEI-94-TR-11*.
- Hogan, J., G. Smith and R. Thomas. (2002): The Real World Software Process, *Proceedings of the Sixth Asia-Pacific Software Engineering Conference*.
- Naur, P. and B. Randell. (1969): *The 1968/69 NATO Software Engineering Reports*. NATO.
- Thomas, R., G. Semeczko, H. Morarji and G. Mohay. (1995): Core software engineering subjects: a case study ('86 - '94), *Proceedings of the Software Education Conference '94*, pages 24-31.