The Real World Software Process

James M. Hogan   Glenn Smith   Richard Thomas
Centre for Information Technology Innovation
Faculty of Information Technology
Queensland University of Technology
GPO Box 2434, Brisbane, QLD, 4001, AUSTRALIA
{j.hogan,gp.smith,r.thomas@qut.edu.au}

Abstract

The industry-wide demand for rapid development in concert with greater process maturity has seen many software development firms adopt tightly structured iterative processes. While a number of commercial vendors offer suitable process infrastructure and tool support, the cost of licensing, configuration and staff training may be prohibitive for the Small and Medium size Enterprises (SMEs) which dominate the Asia-Pacific software industry. This work addresses these problems through the introduction of the Real World Software Process (RWSP), a freely available, web-based iterative scheme designed specifically for small teams and organisations. RWSP provides a detailed process description, high quality document templates – including code review and inspection guidelines – and the integrated tutorial support necessary for successful usage by inexperienced developers and teams. In particular, it is intended that the process be readily usable by software houses which at present do not follow a formal process, and that the free RWSP process infrastructure should be a vehicle for improving industry standards.

1 Introduction

Recent years have seen a pronounced shift in software engineering practice away from linear waterfall process models toward the iterative approaches pioneered by Boehm [2] and others[1]. It is well-established that the use of a defined, repeatable process is an important factor in improving the likelihood of project success (see for example the data provided by the International Software Benchmarking Standards Group [6]) and this insight has formed the basis of a number of software process improvement frameworks – such as SEI's Software Capability Maturity Model [11]. However, it is not sufficient merely to define such a process and to document it carefully in the firm's software process manual: productivity improvements depend upon its actual deployment as part of software development practice, and its refinement in the light of project experience. However, the overheads inherent in obtaining suitable process infrastructure and facilitating its usage within each team may be troublesome for the Small and Medium size Enterprises (SMEs) which characterise the software industry in the region.

[1]Although it should of course be noted that even the waterfall process in its original statement [10] allowed for continual iterative refinement of the software system.

The dominant iterative process architecture is supported by a number of commercial vendors, notably through the Unified Process framework of Rational [9], a firm which also markets a successful suite of CASE tools integrated to varying degrees with the process. However, such systems introduce a number of barriers to their adoption by SMEs. The initial purchase cost and recurrent licensing expenditure - which may range from several thousand Australian dollars for MicroSoft Word based document template sets, through to tens of thousands of dollars for integrated CASE environments - are readily seen as problematic. Yet, the additional costs of configuring the process for the specific needs of the company – and of training existing and newly appointed staff in its usage – form a less obvious but equally troublesome obstacle in the path of process maturity.

Non-commercial process frameworks such as OPEN[2] provide freely available alternatives of high quality, and appear the obvious solution for firms which cannot afford the commercial products. However, their suitability for the SME environment remains limited by their all-encompassing generality, with the configuration and training costs again likely to prove insurmountable. It would seem that a pre-requisite for rapid process improvement by SMEs is the provision of a process configured specifically

[2]Object-oriented Process Environment and Notation [8].

toward small teams, with supporting infrastructure and integrated tutorial support to allow ready adoption by inexperienced developers.

This work addresses these needs through the introduction of the Real World Software Process (RWSP), a freely available, web-based iterative process designed specifically for small teams and organisations. RWSP embodies the principles of OPEN [8] and the Agile Methodologies movement [1], but is deliberately less generic than these frameworks, being tailored for the SME environment. At the core of the process is the notion of the *tight* spiral [7], providing frequent deliverables and the opportunity for regular client feedback and the consequent refinement of specifications – a characteristic shared with a number of processes which fall under the Agile umbrella.

In addition to providing high quality document templates - including code review and inspection guidelines - the process documentation offers integrated tutorial support, guiding the team through an initial startup phase, and subsequently through multiple process iterations during which functionality is extended incrementally. It is our intention that the process should be readily usable by software houses which at present do not follow a formal process, and that our targeted process infrastructure should be a vehicle for improving industry standards.

This paper is organised as follows. In section 2, we examine the relationship between RWSP and the earlier work of Jeffery, and its links to the Agile movement and OPEN framework. Section 3 considers the process in detail, proceeding from an outline of the process architecture through to detailed examination of the role of each cycle, and discussion of the integrated tutorial support. In section 4 we report on the experience of local software engineers who have trialed the method during its development, and the feedback from industry and academic experts who have reviewed the material. We conclude in section 5 with a discussion of refinements to the material which will be undertaken in the coming months, with particular reference to the issue of tool support.

2 Background

The Queensland University of Technology (QUT) has for more than a decade provided undergraduate software engineering students with substantial process support - particularly through the availability of professional quality document templates (with usage guidelines) and example project documentation. Over time, as students have moved into the profession, these materials have formed the basis – and in some cases the whole – of the software process infrastructure of a number of software houses in the region. In 2000, the authors received a substantial university-based grant to enhance the Faculty of Information Technology's undergraduate Software Engineering programmes, with a particular focus being to integrate industry experience and best-practice within the curriculum. In the light of this history of involvement in the industry, and the recognition that the needs of student teams in many respects mirror those of inexperienced teams within SMEs, it was decided from the outset that the resulting process infrastructure should be formally released to the software development community under a non-restrictive licensing arrangement along the lines of the GNU General Public License [4].

In developing RWSP, the central objectives were to take account of the pressures brought to bear upon modern software houses, specifically the need to produce quality software in a restricted time frame. In particular, we were strongly influenced by the comments of a number of industry technical leads who were willing to answer a number of questions about the core aspects of their development processes, and to offer frank insights into process deficiencies. A recurring theme in these discussions was the difficulty of managing requirements, and of maintaining the relationship with the documented requirements through subsequent phases of the project. A strong linkage of this kind was viewed as essential in order to guard against the twin project nemeses of scope creep and team complacency in the face of slippage.

Similarly, the leads reported substantial difficulty in ensuring that adequate unit and integration testing is performed, and there was a strong preference for automation of unit testing and for integration testing to be performed by an independent team.

After some discussion and a number of refinements, it was decided that the new process should be guided by the following principles:

1. The process should be highly visible but have a minimal cost impact on a project.

2. All artefacts produced must be demonstrably useful to the software project, to the student assessment process and to our goal of ongoing software process improvement.

3. The process should support rapid prototyping and iterative development.

4. The process should embody proven requirements management techniques.

5. The process should use Unified Modelling Language (UML)-based problem and system specification.

6. The process should facilitate rapid translation into industry standard OO languages (especially java & C++).

7. The process should support and encourage independent verification and validation.

8. The process should incorporate programmer level productivity measures such as the Personal Software Process (PSP) [5].

On the basis of these guidelines, and the target audience of SMEs and students, only a demonstrably lightweight process could be considered. Moreover, the industry focus upon requirements management and client feedback – together with the known difficulty of large software projects – encouraged a strong bias toward Agile iterative methods, in which the time between releases is kept extremely short. The 'tight spiral' software engineering course of Jeffery [7] proved a particularly useful model. In this programme, advanced SE students undertake a major project through *five* process iterations within a fifteen week semester, with a working release required at the end of each phase. Jeffery's experience, in addition to demonstrating that such an approach could be successful with inexperienced developers, strongly reinforced the view that short or tight iterations offered the greatest likelihood of success.

While it remains difficult to assign credit for improvements in productivity to particular aspects of a process, there are strong arguments to suggest that the following aspects are significant:

- The ongoing focus upon functionality, with teams devoted to the implementation of a clearly measurable set of a features, rather than the more nebulous concept of a large system 'chunk';
- Regular benchmarking of the software system against requirements and the release schedule, with the (hopefully) inevitable consequence of regular unit and integration testing;
- The ability to organise the requirements into a staged release plan, so that the current software development task remains cognitively manageable;

These considerations have led to the development of a process that has clearly defined phases ranging from the initial set-up of the project, through cyclical development phases to project finalisation. The process architecture, and the role of each phase, are explored in detail in the following section.

3 Process Description

While the Real World Software Process is inherently iterative in nature, during development it was felt that the approach would be made more accessible to the novice if the configuration overhead of the repeated phases was kept to a minimum, with more strongly programmed startup and finalisation phases bracketing the generic cycle.

Thus, the RWSP consists of four distinct regimes, termed here:

- *Phase Zero*, describing the process which takes place prior to the commencement of the formal software project;
- *Phase One*, incorporating requirements gathering, release planning and the initial functionality;
- *Phase N*, the generic, repeatable cycle in which the functionality of the system is incrementally extended and the requirements reviewed; and
- *Finalisation*, addressing issues of delivery and installation.

These regimes are considered in detail in the subsequent sections, with the bulk of the text drawn from the explanatory material integrated with the process. The process may be viewed *in situ* at

`http://www.fit.qut.edu.au/~rwsp`

.

3.1 Startup - Phase Zero

The development process starts with business planning and the role of the new software system within the organisation's strategic plan. A request from one part of the business may initiate the process by establishing the mission and the objectives of the work. These should be underscored by a list of priorities and associated measures to monitor progress and evaluate success. The outcome of this process is a short and succinct project proposal document. It should contain a statement of the broad project mission, together with an initial estimate of costs and a list of other constraints. The proposal should contain:

- A problem definition in terms of business processes and commercial impact;
- The names of the project sponsor, domain experts and key users;
- The broad scope of the project, a cost-benefit justification and the project completion criteria;
- Optionally, cost-benefit, feasibility, risk and critical success factor appendices. These will be produced as required in order to obtain the sponsor's signature;
- Optionally, a throwaway prototype to demonstrate the concept.

In particular, the Problem Statement should outline the main objectives of the project, as well as providing some high-level abstract functional requirements - thus forming the basis for the formal Requirements Specification during discussions with the client.

3.2 Commencement - Phase One

Phase One commences the software development aspect of the project and focuses on discovering initial requirements. A development team must be established, and familiarised with the RWSP. In collaboration with the client, requirements are gathered and written into a Requirements Specification document, after which Release Planning is undertaken to set milestones and determine the likely number of iterations required.

The requirements and design sections of this phase are lengthy and detailed, and the implementation phase may be smaller than in future phases. The idea is to create rapidly a core system that meets the central requirements of the client, so that further requirements can be gathered. The core system is evaluated in consultation with the client, and the results of this evaluation are fed into the requirements for the next phase.

The process support pages give detailed discussion of these issues and links to templates and additional resources. Excerpts from this material are included below.

Phase One

After project approval coming out of Phase Zero, the software development side of the project needs to be initiated. The end product of Phase One is the minimalist system or prototype.

Organise a team of developers

In any software project, it is important to choose a dedicated team. Ideally, the team will have fewer than eight members. Ensure that the entire team is familiar with the RWSP, and is committed to following the process throughout the course of the project. [For a very small project, an individual developer may be sufficient, but use of the process will require that the developer assume a number of roles.]

[**Links:** Team Size; Choosing Suitable People; Team Goals; Team Roles; Team Problems; Managing Team Membership; **Templates:** NEAT sheets – a format to organise a meeting agenda]

Organise document templates

Included with the process are several document templates which can be used to help document and manage all steps of the process. It is important that before you begin the project, you have obtained copies of each template and understand how each one is to be used. Some documents are better used in printed form, others are easier to use on-line.

The resources that the team creates are some of the most valuable resources in a project. The documents that are produced after hours of work need to be carefully managed and stored.

[**Links:** RWSP Documents; Keeping Track of Documents; Guidelines for Document Management]

Requirements Modelling

Expanding on the initial Problem Statement, you should outline the main objectives of the project, as well as providing some high-level functional requirements. In discussions with the client, these requirements will evolve into a formal Requirements Definition. You should involve the client in the development of the acceptance test plan. Minimally, the client should agree with a plan that you write; at best they should write it in collaboration with the team.

[**Links:** Problem Statements; **Templates:** Acceptance Test Plan]

Requirements Engineering

Determine and record the functional and non-functional requirements for the system. Requirements Analysis involves interviewing clients and end users and observing the work environment in an attempt to discover exactly what is required of the project. The Requirements Definition is the record of what the client wants. This step and the previous step may need to be iterated until satisfactory and realistic outcomes are likely.

[**Links:** Requirements Engineering; Requirements Analysis; Requirements Specification; Requirements Document; The Degree of Detail; UML; Use Cases; Creating a Use Case Diagram; **Templates:** Requirements Specification; Use Case Template]

Undertake Release Planning

Once there is some indication of the project objectives, the team needs to plan how it will spend its time. Some things to consider are the number of iterations to be completed, dates of milestones, and resource requirements.

When deciding on the number of iterations, it is important to balance the size of the project with the available time and financial resources. A large or complicated project will require more iterations.

Time and cost estimation is very important. Roughly determine the tasks that need to be completed in order to fulfill the client's requirements. Based on the time taken previously on similar tasks, estimate how long each task will take, then develop an estimated timetable to come to a completion date.

Compare the completion date with the client's desired deadline. If the client's deadline is earlier than the estimated completion date, some changes will have to be made. This may mean adding staff or resources. However, adding too many extra people will not solve the problem, as the communication overhead will take up any time that would be saved by adding additional personnel. If possible, negotiate with the client to set a realistic deadline. This step and the next may need to be iterated until satisfactory and realistic outcomes are likely.

[**Links:** Why do Release Planning?; Writing a Release Plan; Managing Teams; Changing the Plan]

High-level Design

It is strongly recommended that all team members be involved in the high-level design. Experience has shown that if everyone is involved, then team members will be more committed to the design and thus more likely to adhere to it during detailed design and implementation – thus assisting in the system integration step. This is called a "flow on effect". It has also been shown that team members will receive more personal satisfaction in all milestones and outcomes.

Design should not be rushed, as mistakes made in the initial design will usually prove costly to fix once implemented. A good design is any design which produces efficient, maintainable code. For code to be maintained easily, it must be possible to add or remove functionality as required.

The outcomes of this phase would be a high-level diagram (usually a class diagram) and an overview of control and data flow (eg UML interaction diagrams and specifications of public interfaces).

Once the high-level design is completed, the documents are reviewed and particular subsections are allocated to individual team members or small groups of team members for detailed design.

[**Links:** CRC Cards; Principles of Good Design; Cohesion; Coupling; Inheritance; Reviews; **Templates:** Design Specification; Integration Plan]

Detailed Design, Implementation & Unit Testing

The first step of this section involves individuals or small groups working on the Detailed Design of their modules. When the detailed design is nearly completed, the associated unit tests should be produced.

The next step involves the transformation of the detailed design into code and subsequent review of this code for quality and defects.

The final step of this section involves unit testing of the modules written by the developer or small group of developers. The three steps – detailed design, coding and unit testing – are usually cyclical, as the identification of defects during code review or unit testing necessarily results in a revised implementation and may result in a revision to the detailed design.

Once developers have completed their individual units (including unit testing), they should swap code samples and perform a peer review to ensure code quality and consistency within the team.

Upon successful completion of unit tests, developers should integrate their unit code into the main system in sequence, returning to their code to fix any bugs which have emerged from the integration process. [Scheduling of updates to the code base is a critical issue – see the link to Configuration Management below.] As before, this process may require iteration as far back as the detailed design, and it is imperative that unit tests be repeated successfully prior to re-integration. [Guidance on System Integration is given below.]

[**Links:** Purpose of Testing; Good Testing Practices; Reviews; Configuration Management; Frequent Build; **Templates:** A List of Checkpoints for Source Code; Guidelines for Code Review]

System Integration

Apply the integration plan looking for defects in the integrated system. Those who perform these tests need not be responsible for fixing the faults which emerge. When defects are found, they must be categorised, assigned an appropriate priority and traced back to their source.

In some environments the use of Independent Verification and Validation (IV&V) teams can be helpful. These are people not involved in the programming of the system, whose job is to test the current system and report back on any errors found.

[**Links:** Testing; **Templates:** Integration Plan]

Acceptance Testing

Apply the Acceptance Test Plan. If any errors are discovered, go through the steps of fixing and changing until both development team and client are satisfied. Acceptance testing may be performed by the developers, an Independent Verification and Validation (IV&V) team or in conjunction with the client.

Acceptance testing is normally performed in the following order:

- Acceptance testing by the development team (in order to ensure that acceptance is likely);
- Alpha testing performed by the client;
- Beta testing following installation at the client site;

although Alpha and Beta testing may not take place for every release in the schedule.

[**Templates:** Acceptance Test Plan]

Phase Evaluation

Evaluation should involve an assessment of the product produced, and an analysis of the effectiveness of the process used to create the current system. The aim of evaluation is two-fold: to recognise those strategies and techniques which proved effective during the course of the project so that their use may be reinforced or expanded; and to identify areas in the process and product that need to be improved in the next phase.

[**Links:** Purpose of Evaluation; One method: PNI; Other Methods; **Templates:** Phase Evaluation Template; PNI template]

3.3 Cycle - Phase N

In RWSP, Phase N is the lightweight generic process iteration which may be repeated as many times as required by the scope of the project. Phase N is centred around the refinement of requirements and release planning on the basis of testing and client evaluation from the previous phase. Over time, the functionality of the system is incrementally enhanced until all project requirements have been implemented. As before, excerpts from the process support material are provided below, although a number of links and resources have been excised due to their similarity to those listed earlier.

Phase N

Phase N is a label applied to each of the possibly numerous iterations following Phase One and prior to final delivery. The aim of these cycles is to extend the work that was done previously, implementing more features in each successive phase, until the project is complete.

Requirements Engineering

At this stage, the updated requirements document should be based on the combined feedback from the client and team's comments from the previous evaluation. At a minimum, the team should consult with the client at the start of each phase on the features that are to be included in the product. Any new features or changes to existing features need to be added to the Requirements Specification, and incorporated into release planning.

Release Planning

The team will need to decide which of the requirements can be realistically implemented in this phase, and which ones need to be left for future phases. Estimate how long it will take to complete each task for each feature, and how many people are required for each one. Put the tasks in the order in which they must be completed, and set milestones that will be realistic for the number of people available, and the difficulty of the project.

Check the Release Plan to ensure that the project is on schedule and budget. If the project is behind schedule or over budget, negotiate with the client regarding possible changes to the requirements or release plan. This may affect both the Release Plan and the Requirements Specification.

High-level Design

In the design phase, the team should first look at the existing design, and make any necessary changes. These changes may be to fix design flaws found during system testing of the previous iteration or may be to adapt the design so that it can accommodate planned extensions in this iteration.

Subsequently, new features can be designed in light of the existing system. Any existing design documentation needs to be up-dated to reflect the changes and additions made in this iteration. Principles of good design are to be maintained, with design reviews used to help ensure this.

Detailed Design, Implementation & Unit Testing

The aims and tasks of this activity are the same as in Phase One. In Phase N, this activity is applied to both adding new requirements scheduled for this iteration and to the modification of the existing system resulting from fault reports. With refactoring and code reviews being conducted to help ensure the quality of the system is maintained or improved in each iteration.

System Integration

As more features are added, defects may be introduced to the previously-tested system. This is where system testing comes into its own, as it is sometimes difficult to predict the effects of added components on what *was* a quality system. Therefore, it is not sufficient to only test the new part of the system. One useful strategy at this point is to focus on new tests which exercise the new components and their interaces to the main program, while repeating the existing tests that remain appropriate to the system.

Acceptance Testing

The tasks performed in this activity are the same as in Phase One. The only addition for this iteration is that both the previously implemented requirements and the new requirements for this iteration are tested for acceptance by the client.

Phase Evaluation

Evaluation forms the basis of future iterations. The comments from the client and team members directly influence changes made to the requirements and the development process, and therefore the entire program. Even in the final iteration, an evaluation confirms that all requirements have been met, and the client is happy with the finished product.

3.4 Finalisation

The Finalisation phase is used to separate out issues of installation and handover from the generic cycle in order that the latter retain its identity as a lightweight, repeatable iteration. The brief guidance provided in the process support material is reproduced below.

Finalisation

When the project is nearing completion, the team must finalise arrangements for the installation of the system at the client site, and for the scheduled *turn on* and *turn off* dates.

Turn On/Turn Off Dates

As the client's staff will probably be working with existing software, you will need to discuss with the client when to install the new software so that there will be no clashes between the existing software and the new system. Negotiations must result in firm dates for the changeover, and a detailed record of any special arrangements which are the responsibility of the project team.

Installation

Once the client is satisfied that the system passes the agreed acceptance tests, installation of the system can begin. Install the system at the client's place of work and repeat the suite of acceptance tests. If training is needed, arrange it. Some discussions with the client about system maintenance – and the scope and cost of ongoing technical support – are needed, and such arrangements must be clearly defined prior to the end of the installation process.

Project Evaluation

At this stage the project and process is evaluated in an attempt to limit problems in future projects.

3.5 Integrated Tutorial Support

As will have been clear from the treatment of Phase One in section 3.2, the RWSP material provides substantial support for the novice user through links to explanatory material, specialised guides and process templates. These documents range in sophistication from single sentence definitions to extended guidance for particularly troublesome process activities, in which a number of alternative strategies may be explored. In particular, a menu of detailed practical suggestions is given for requirements gathering and analysis, as exemplified by the following excerpt:

> The Requirements Specifications are the semi-formal documents that provide details of all requirements for the system. Techniques you may apply during requirements gathering include:
>
> 1. Think about the problem statement and brainstorm some ideas on what you think the client is going to require. List these ideas down.
> 2. Write a list of questions to ask the client. For example, ”What are you using at the moment?”, ”Why is this no longer suitable?”
> 3. Meet with the client, and ask him/her to explain exactly what he/she wants. Listen carefully and take notes. Try to summarise back to the client what he/she requires to make sure that you understand. Take a tour of the environment in which the system will be used – maybe it has to be capable of operating underwater, or in a dark room – the client might be left with a totally unsuitable product if it does not fit within the working environment.

4. Arrange another time to meet again to discuss the requirements.
5. Research other similar development projects.

Where possible, glossary entries and explanatory text are kept at an elementary level, the intention being to make developers aware of the relationship between industry terminology and concepts with which they may already be familiar, and to provide options which require little additional research or training. However, some activities such as testing and code review may require significant learning, and so the documentation is focused upon the provision of more structured, sequential material to enable the team to become productive in a limited time frame.

Nevertheless, while the process documents are comprehensive, they are not intended as a substitute for a formal course in software engineering, although some hyperlinks between lecture material and the process web site are planned. Ideally, the team members will have had a good grounding in SE process material from their university education, and the process and its web site may then play the role of a coach, ensuring that their development practice remains in accordance with the best technical practice. The usefulness of RWSP in an industry context is among the issues considered in the next section.

4 Industry Review

The authors have been fortunate throughout the development of the process and its associated infrastructure to have received valuable feedback from academic and industry-based colleagues alike. As noted earlier, a number of experienced and talented technical leads helped enormously in shaping the process in its early stages. Subsequently, we have benefited from both their comments and those of a number of academic colleagues, who have examined and reviewed the material at various stages of maturity. More importantly, we have been influenced by a number of trial runs of the material *in practice*, with the process forming the basis for both introductory and advanced software engineering courses within the Faculty.

However, the ultimate test of any new process must lie in its application to a live commercial project, and the experiences of four development groups from the Queensland industry are reported in this section. The groups span four different industry sectors – finance, government, telecommunications and security – and all are small teams, ranging in size from two to five developers. The process maturity level prior to the adoption of RWSP was low, with none of the teams following a formal process prior to its adoption. Process maturity and software engineering experience of the teams may be summarised as follows:

- Three of the teams include at least some team members who had exposure to a formal software engineering process in a university project;
- One team is made up of developers with no formal training and no exposure to processes, aside from reading a small number of articles on the issue;
- In three of the teams all team members had some industry experience, ranging from two years to over twenty years in one case; and
- One team of two had one member with two years experience and the other with no experience.

The experiences of these teams are considered in turn below.

Team One

The first team to trial RWSP completed an intranet project that required thirty-two person months of effort. They followed the structure of the process and used initial versions of some of the templates, but did not have access to the process web-site. Feedback from this team has led to modification of some of the templates and revision of some aspects of each phase. Moreover, it has reinforced the view that the process web site will be beneficial: as developers follow paths other than those specified in the process, a simple and highly-visible process reference point should prove very useful in turning their focus back toward the agreed methodology.

Changes that were made to the process as a result of this team's experience include:

- Use case and requirements specification templates were modified to streamline their usage;
- Measuring and monitoring project progress was more strongly emphasised in Phase Zero;
- Requirements management and traceability was more strongly emphasised in Phase One and Phase N;
- The testing process was completely rewritten for both Phase One and Phase N.

Due to an organisational restructure, this development group no longer exists, and the majority of the development work within this firm is now outsourced, thus limiting the opportunities for further evaluation of RWSP at this site.

The other three teams are currently at different stages of their first project using RWSP, with various difficulties introduced by the culture of the organisation. In each case, the experience of previous processes is limited, and so the report is focused upon the usability and contribution to project success provided by the process.

Team Two

The team with no process exposure has completed phases Zero and One, and one three iterations of Phase N. They are a successful software development house servicing the security vertical market. This project is their first contract to develop a bespoke system for a client. Due to their lack of experience they have sought advice on how to run the project. (The client appears to have well-defined formal processes and this company wishes to appear in a good professional light to their client through the use of a defined process.) At the start of their project they also did not have access to the process web-site. They are using the process infrastructure and are finding it very useful. Once this team was able to access the process web-site they thought it was a useful tool and said "they wished they had access to it at the start of the project". Experience with this team has only led to very small changes in wording to parts of the RWSP description. The limited process experience of the team, and their desire to adopt a process, has meant they have not provided a detailed critique.

Team Three

The third group is a single team within a large company. This company follows no formal software development processes, and has had only limited success with large software projects in the past. At the time of writing, a number of different teams are trialing different processes, with the intention of bringing their development under control. The team trialing RWSP is championed by a business analyst and has three full-time software developers on the project. They are developing a product for an internal client using client-server technology.

Phase Zero was completed by the business analyst on her own and the team has completed Phase One and a single iteration of Phase N. This team has had access to the process web-site from the start of their project. They are using some of the process infrastructure and some of their own document templates. This team is finding the process fairly easy to follow and has not raised any issues to date.

Team Four

The final group is a small team within a medium-sized organisation. This organisation has a formal process that is meant to be used for projects but which is usually ignored. Many of the projects undertaken by this organisation are high-risk in nature, with the potential for large losses with some types of software failure. The information technology manager has committed resources to adopt a software engineering process which interfaces well with management, but will also be *used* by developers. A formal, but lightweight, process like RWSP is thus an ideal candidate.

This project has only recently commenced and the team had completed Phase Zero and started Phase One when a new project manager took responsibility for the team. They are now revisiting Phase Zero with a view to a change in emphasis for the project, and thus no useful feedback has been provided from this team so far. However, the team have made a commitment to provide comments and feedback throughout their usage of the process.

These reports are considered further as part of the discussion in the following section.

5 Discussion and Further Development

On the evidence of early trials in both a university and an industry setting, the Real World Software Process introduced in this paper has proven to be a useful addition to the process infrastructure available to the SE community. Significantly, a large majority of the users to date have found the process easy to follow, and their suggestions have done much to clarify the guidelines and tutorial support. Most of the process design principles listed in section 2 have been incorporated successfully, notably those concerned with visibility, usefulness of artefacts and limitations on their number, and rapid and iterative development. The use of the 'tight spiral' approach has the added benefit of supporting effective requirements management, an effort supplemented by the extensive tutorial material on interaction with the client.

Yet there remain a number of weaknesses in the process as it stands, principally the limited attention to configuration management, and the lack of an appropriate source control system. These issues will be addressed during the second half of 2002 through the introduction of the open source Web CVS system, and the development of usage and installation guides accessible to the target audience and their integration with the process web site[3]

While the process is designed around object oriented development, and provides a significant amount of tutorial support for UML – at least at the level of class diagrams, use cases and interaction diagrams – this is another area in which integrated tool support is desirable. Many students within our programmes use Rational's Rose system under academic licensing arrangements, but this arrangement cannot of course be extended to the wider developer community, and we are presently investigating a number of open source alternatives.

Two additional objectives were considered early in the development of the process, but deferred until subsequent releases:

[3]This work will be supported by a Faculty Teaching and Learning Grant. Material on CVS is available at `www.cvshome.org`.

- It should be designed so that the student's implementation of the process can be easily assessed within a process improvement framework, such as CMM.
- The process should support the development of component-based architectures and the integration of components into a system. It should encourage component re-use;

The first of these requires a greater focus upon the collection of data – notably defect densities and effort records – at both the team and the personal level. Both defect counts and personal productivity are at present addressed through the use of log sheets, but the estimation machinery of the PSP (and its cousin the TSP) is not currently employed, and its usage again may be dependent upon the identification of appropriate tool support. Nevertheless, estimation is a critical issue in the commercial environment, and a perceptible weakness of a number of Agile approaches, so the issue is one which must receive attention in subsequent revisions.

Support for component based architectures is to some extent merely an extension of that for object oriented development, but this is to understate the importance of architectural design in the development of large systems. In common with a number of Agile approaches, RWSP offers only limited support in this area, and it remains an area in which the process needs to be refined. The introduction later this year of the Java Metrics Reporter (JMR) [3] is one positive step in this direction, with the tool offering analysis of the software system at a number of levels of description. Once tool support for UML is integrated with the process, it is intended to exploit the work undertaken as part of the JMR project to provide analysis of designs from the system class diagram. While this approach does not address the issue of design in terms of existing components, it is nevertheless likely to result in a higher level of re-use, and superior system design.

However, regardless of the additional convenience provided by good software tools, successful development of the process and its supporting material will depend upon the extent to which it is disseminated and used within the industry, and on the refinements which emerge as a result of the feedback provided. This paper is an important step in facilitating its adoption.

Acknowledgements

The authors would like to acknowledge the numerous industry developers, academic staff and students who have provided feedback during the development of the process. Particular thanks are due to Michael Forbes (Opcom Pty. Ltd.) and Andrew Reye (Astracon Pty. Ltd.); Chris Ho-Stuart (QUT); David Carrington and Anthony MacDonald (UQ); and Jim Over (SEI). This work was supported by a QUT Teaching and Learning (Large Project) Grant.

References

[1] The Agile Alliance, The agile development manifesto, http://www.agilealliance.org/, 2001. Accessed: 30/06/2002.

[2] B. Boehm, A spiral model of software development and enhancement, *IEEE Computer*, 21(5):61-72, May 1988.

[3] J. Cahill, J. Hogan, and R. Thomas, The java metrics reporter – an extensible tool for OO software analysis, *Asia-Pacific Software Engineering Conference*, 2002.

[4] Free Software Foundation, Inc., GNU general public license, http://www.gnu.org/copyleft/gpl.html, 1991. Accessed: 30/06/2002.

[5] W. Humphrey, *A Discipline for Software Engineering*. Addison Wesley, Boston, MA, 1995.

[6] International Software Benchmarking Standards Group, The benchmark, v.7, http://www.isbsg.org.au/, 2001. Accessed: 30/06/2002.

[7] C. Jeffery, Tight spiral project for communicating software engineering concepts, *Proceedings of the Third Australasian Conference on Computer Science Education*, pages 136-144, 1998.

[8] The OPEN Consortium, Object-oriented Process Environment and Notation, http://www.open.org.au/, 2002. Accessed: 30/06/2002.

[9] Rational Corporation, The rational unified process, http://www.rational.com/products/rup/, 2002. Accessed: 30/06/2002.

[10] W. Royce, Managing the development of large software systems: concepts and techniques, *Proceedings of IEEE WESTCON*, 1970.

[11] The CMMI Product Team, CMMISM for systems engineering/software engineering/integrated product and process development/supplier sourcing, version 1.1, continuous representation (CMMI-SE/SW/IPPD/SS, V1.1, Continuous), *Technical Report CMU/SEI-2002-TR-011*, 2002.