A Great Legacy A Great Opportunity

Steve Adolph - WSA Consulting Inc
Wesley Coelho - “SmallEcommCo”

SmallEcommCo is an independent software product company whose founders developed the core product. The product was a hit and the company experienced rapid growth which began overwhelming their development capabilities. In addition, it was difficult at best to leverage new and emerging technologies with the legacy architecture and technologies that formed the basis of their product. However, it was well known that leading edge technology was important to SmallEcommCo’s ability to remain ahead of their competitors.

After much deliberation, SmallEcommCo’s principles made the bold decision to not only revamp their core product but also their software development process. This white paper is the story of how SmallEcommCo revamped its software development process to support their growth. OpenUP was adopted as part of this revamping and this white paper’s focus is on how OpenUP was introduced into SmallEcommCo and the lessons learned during this process.

SmallEcommCo

SmallEcommCo develops and markets a high performance e-commerce system built on a foundation of open source components (e.g. Velocity, Spring, and JBoss Rules). The system is designed to be flexible so customers can modify it to suit their needs at a reasonable price.

The legacy version of their e-commerce offering is successful and attracted a number of high-profile Fortune 100 customers. In the last year, staff grew from around 20 employees to around 60. This growth was driven by customer demands for new features and new systems. However, the legacy architecture limited SmallEcommCo’s growth potential and they knew greater efficiencies could be gained by creating a new platform on new technologies that could be easily extended to accommodate new features.

Furthermore, technology was moving ahead rapidly. What only a few years previous had been bleeding edge technology was becoming mainstream or even legacy. The existing system was built on Hibernate, JSP, and Struts. For the new system to remain on the leading edge and support the new innovative features customers were demanding, it would be built with a new architecture using Hibernate[1], Spring, JBoss rules engine, Lucene search, and Ajax technologies.

[1] Hibernate has since been replaced by OpenJPA

Eating the Elephant

A number of approaches were offered for legacy replacement but they all could be distilled down to two options, eat the elephant piece by piece, or swallow it whole. After much debate it was decided to take the all or nothing option, swallow the elephant whole and re-write the system from scratch rather than evolve the previous system. The radically different nature of the desired architecture simply made it infeasible to evolve the existing system to the new design.

Legacy replacement projects are high risk projects. According to Adolph people underestimate the amount of risk associated with a legacy replacement [1] because it is tempting to assume the legacy mitigates many of the risks associated with a software development project. Unfortunately this is only true when the legacy replacement closely mimics the behavior of the existing system. In fact, the risks may be higher than building a new system because of what Fredrick Brookes referred to as the "second system effect" While many may insist the legacy will mimic the existing system, this is rarely the case because the forces that drive the need for legacy replacement ensure the legacy will be different. Otherwise, why engage in a legacy replacement in the first place?

The decision to create a new system from scratch also added an additional project management risk because removed one important degree of freedom from us, we could not trade features for time. If we consumed all our time and resource and did not deliver all and more functionality than the existing system we would have failed no matter how well we implemented the features, or how beautiful and clean our technical architecture was. No one gets excited about wonderful new systems that deliver fewer features than their predecessor.

Replacing the Methodology

There was consensus among SmallEcommCo's developers and management that their current software development practices were not up to the challenge of managing this project and mitigating the substantial risks. The existing development practices followed a waterfall model where an analyst created use cases and screen shots for a new feature which were then given to a developer. The developer implemented the behavior described by the use case and a test engineer then verified the behavior. While these practices had worked adequately in early projects, recent projects clearly demonstrated the inadequacies of this home grown methodology. At this point management decided to seek outside help in managing this project.

OpenUP

SmallEcommCo engaged WSA Consulting to help them manage this project. WSA Consulting Inc often introduced lightweight versions of the Rational Unified Process to its clients during an engagement. WSA believes many of the practices encouraged by RUP such as iterative development, requirements management, risk management and architecture are necessary practices for coping with the risks in uncertain and fluid development environments. However, many small organizations find RUP's comprehensiveness daunting. Most small firms do not have a person with the necessary knowledge or confidence to tailor an appropriate RUP process, let alone a specialized software process engineering group.

At this time, an early pre-release version of OpenUP was available and in WSA's opinion there was a good fit between what OpenUP offered and SmallEcommCo's needs. After discussions with the principles at SmallEcommCo, it was agreed to adopt OpenUP to manage the risk associated with the legacy replacement.

What is OpenUP

OpenUP is a small team open source variant of the venerable Unified Process and is available online at http://www.eclipse.org/epf. OpenUP is an iterative process that is minimal, complete, and extensible. It values collaboration and stakeholder value over unnecessary deliverables and formality. OpenUP combines elements of Agile and Unified methods to create a hybrid process that leverages proven best practices of both process perspectives that include:

- Time-boxed iterations that deliver incremental stable functionality to the stakeholders
- Frequent iteration assessments that adjust the course of product development
- Tight collaboration between stakeholders and team members
- Contextual requirements that reduce ambiguity and improve collaboration (use cases)
- Classic best practices for designing object-oriented systems
- Robust test cases

Unified processes have helped teams of many different sizes and industries to become successful in solving their stakeholder's problems. OpenUP is a lightweight realization of the essential elements of a unified process. OpenUP describes the following:

- Four phases (which contain iterations) that provide different perspectives during the development of the system
- An early focus on architecture so a stable framework can be validated before the bulk of the system is implemented
- Rigorous examination of mitigation of project risks that reduce the possibility of unforeseen problems and increase project predictability
- Use cases to characterize functional requirements and drive testing and development

Agile processes and techniques that have a track record of improving a team's success are incorporated into OpenUP. These include:

- A Scrum-based and collaborative approach to project management
- An agile perspective of estimation and planning

- Self organizing work assignments
- An evolutionary approach to architecture and design through different levels of refactoring
- Test Driven Development

Merging these different perspectives creates a process that's lightweight, robust, and whose structure and components have a proven history of making development teams successful in providing value to their stakeholders.

This process defines roles in order to clearly call out what skills are necessary to perform certain tasks. At the same time, the process is intended to be applied to self-organizing teams. If a team member has the skills necessary to be an analyst, a tester, and a developer, then it's appropriate for that individual to play all those roles. Roles are not meant to inhibit team members from performing any work that needs to be accomplished.

Why OpenUP

Even as a minimal process, OpenUP seems quite voluminous and therefore daunting. Exploration of the process reveals delivery processes, task and step descriptions, roles, work products, guidelines, concepts, principles, over 500 pages of content in all and growing. All this in what is claimed an agile process. However much of this content is not prescriptive, but rather provided as guidance, as reference, information available to you. The essence of OpenUP is it is a collaborative and iterative software development process that focuses on architecture and requirements management. While most agile methodologies are expressed as a few simple principles, they are supported by volumes of books, articles and websites. Agility does not mean simple.

Even if OpenUP is agile, why use OpenUP then? Why not use some other agile process? We ourselves have stated in public forums the success of the project resulted from the domain experience, leadership, and collaboration fostered by healthy team social dynamics. While the use of OpenUP was not a decisive factor in our project success, it was a powerful enabler that provided the common language and way of thinking that coordinated the efforts of our team so they did not work or talk to cross purposes. Any iterative agile process that facilitated rapid feedback would have sufficed. However, OpenUP had four key advantages:

1. Its focus on architecture, requirements management, and risk management. The process encouraged developers to think about what feedback they needed and what they needed to worry about. While these issues may be second nature for experienced software developers, they may not be the issues less experienced developers would know to concern themselves with.
2. OpenUP is more structured than other agile software development methodologies. While there are those who associated structure with coercion, an appropriate amount of structure enables a team to be effective[2]. Neil Harrison refers to this as a "Liberating Form." – structure that liberates creativity[3]. OpenUP provides more guidance and structure on a wider range of issues than most other agile methodologies such as XP or Scrum.

3. None of the team members nor any members of the management had strong objections to the introduction of OpenUP. It is unfortunate that some early experiences, ignorance and fear have lead to many misconceptions regarding other agile methodologies such as XP.

4. It was similar to methodologies WSA was most familiar with and therefore able to most effectively and confidently apply the methodology at SmallEcommCo. There was already a great deal of risk associated with the project and introducing a methodology no one was familiar with would have only added unnecessary risk to the project without any real prospective benefit.

Introducing OpenUP to SmallEcommCo

No great decision making ceremony surrounded the introduction of OpenUP at SmallEcommCo. WSA simply suggested the project leads visit the EPF website (www.eclipse.org/epf) and download the latest copy of OpenUP. Most of the staff at SmallEcommCo were already familiar with UP variants including RUP, but were not using them. Like most software developers, they were eager to learn something new.

Opening up the SmallEcommCo Culture

An important and often overlooked consideration during the introduction of an agile methodology is the relationship mindset among project participants. Unlike more linear chain like waterfall process that usually limits interaction between project participants (e.g. an analyst hands a specification off to a developer who hands code off to a tester), OpenUP is an iterative process and requires people from all disciplines to continuously interact.

For a process like OpenUP to work we needed to create a culture that could support an iterative and collaborative work style. One cannot mandate a social environment that supports effective collaboration. Rather we have to lead by example and select people for our team who are open to working collaboratively.

SmallEcommCo being a small and reasonably fast moving company already had a reasonably good work culture, but most developers still worked fairly independently of one another. We made significant efforts to strengthen the collaborative environment by creating opportunities for the entire team, analysts, project managers, architects, developers, and testers to meet and train together. Simple opportunities to bring people together were the project kick-off meeting, iteration planning and assessments. Leadership molds group dynamics and team performance [4] and we made an effort to recruit a dynamic and energetic project manager from SECO's staff for our project.

Tailoring OpenUP

Minimizing the Minimal

We did not use OpenUP "out of the box." Despite the assertion that OpenUP is minimal and complete, we tailored OpenUP for SmallEcommCo. Much of our tailoring reflected SmallEcommCo's informal development culture and replaced formal documents by assimilating many of SmallEcommCo's informal development practices. Software developers know how to develop software and have an internalized methodology that is the sum of their academic knowledge and industrial experience which they draw on when creating software. Developers know how to analyze the architecture, demonstrate the architecture or design the solution. However, they may not all do it the same way, or may have performed these tasks in different environments. Developers may have differing levels of experience.

We tailored the process to take advantage of developer's knowledge and re-align the disparities. We used OpenUP as a guide or moderator for the developer's existing knowledge. The process of tailoring OpenUP was one of assimilating SmallEcommCo's existing software development practices and aligning developer knowledge of software development.

It is also important to point out here that project kick-off was not the only time we tailored the process. Process tailoring is an ongoing activity of constantly revising the software development process. At the end of each iteration we held a retrospective and modified the process based on the lessons learned. There were practices that early on seemed like a good idea and even worked when the team was relatively new and unfamiliar with the task at hand. Later as team members became more experience with the task set out for them, some of these practices became unnecessary. For example, during the early iterations, when a developer was assigned a use case for implementation, they were first to realize the use case by sketching a simple sequence diagram. Initially this was useful for maintaining architectural coherency, but as the team grew more familiar with the architecture and common architectural mechanisms it became unnecessary and therefore the practice was consensually dropped.

This idea of constantly adapting the process may raise cautionary flags among project managers. How can you manage a project if the software development methodology is constantly evolving? What changes are practices the software developers use to coordinate themselves and not governance or quality assurance practices. For example, when we discovered that some of the ways we were performing reviews were consuming more time than the benefit we received, we did not remove reviews from the process, but sought out more effective ways to conduct reviews.

Furthermore, we did not change OpenUP's project governance practices. We did not change major milestones such as the Lifecycle Objectives or the Lifecycle Architecture, or what we considered to be the fundamental and strategic structure of OpenUP. Rather we modified and adapted the day to day collaboration and work practices of the development team in response to lessons learned.

None of these changes were formally written as part of a localized OpenUP. Usually they were communicated verbally or by e-mail. While most team members were told of OpenUP and how to obtain it, few bothered. Most of them relied on the ongoing training and consulting, the wiki and osmotic learning by doing. An important lesson is no one learns or becomes comfortably fluent with a software process by simply reading it. They have to experience it and the only way to experience it effectively is apprentice with someone who has already worked with it

The important point for anyone considering adopting OpenUP is do not be afraid of minimizing the minimal process. If a specific practice will not work or does not make sense in your context, then don't do it simply because it is part of OpenUP. However, ensure the issues addressed by the practice are addressed to your satisfaction in your environmental context. This is the real meaning of minimal, the issues that must be addressed in any small software project.

Work Products, Documents, Whiteboards, and Wikis

OpenUP prescribes a significant set of work products, far less than the hundreds of artifacts prescribed by RUP. However, the list of work products suggested by OpenUP may still make many novices shy away from OpenUP thinking, "Why do I need all these documents for a supposedly agile process?"

In our opinion, each of the work products prescribed by OpenUP must be addressed, but not necessarily produced and not necessarily produced using the sample template provided with OpenUP. A number of the work products are classified as specifications. In OpenUP a specification defines exactly what something is or what it must do. Examples of OpenUP specifications are the Use Cases, the Supporting Requirements Specifications, Architectural Notebook, and the Test Cases. For many of us who grew up on traditional projects, this means documents and the inclusion of a sample template in OpenUP encourages this point of view. However, the work product templates included with OpenUP must be treated as a gift and a guideline. For if you have nothing else, at least the template will help you understand what issues need to be addressed. Its format and even its use in our view must be addressed within the context of the project.

Albert Einstein's saying "Make everything as simple as possible, but not simpler" applies to documentation. Many people have the mistaken view that agile methodologies are documentation-less. To paraphrase Einstein, make documents as simple as possible, but not simpler." Documentation was intentionally kept to a minimum, but there were still a number of work products we felt were necessary:

- Vision Document
- Project Plan
- Iteration plans
- Software Requirements Specification (SRS) – use cases and supplementary requirements
- Developer's Handbook
- Development Guide
- Deployment Guide
- User Manual

Most of these documents are end user documents and not prescribed by OpenUP. However we draw your attention to them because it has been our experience on many projects that teams forget to create these documents.

The vision document described the goals of the project as a single primary goal that was further refined into specific issues to be addressed by the project. The vision statement is described in more detail later. We believe the process of creating this document was more important than the document itself because creating it required us to clearly define the project's purpose.

The SRS took the form of a set of wiki pages that defined the requirements for each feature. The requirements were specified using use cases and screen mockups created with Visio diagrams. The SRS also contained test cases for the features.

The developer's handbook was an internal document with contributions on topics such as style conventions, commit checklists, security best practices, and information on how to set up a development environment.

The Developer's Guide serves both internal developers as well as clients who are customizing the product. Main sections of this document include Programming with SmallEcommCo, Architecture Reference, and Technical Feature Design. Developers created new entries in the Technical Feature Design section as new features were created. Initially, the contributions consisted of hand-drawn sequence diagrams and class diagrams that were sketched before implementation and then scanned.

However, it soon became clear that the designs typically did not remain stable throughout development and it was necessary to update them after implementation. The scanned paper-based diagrams stored on the wiki made it tedious to update the sketches and we felt that the time required outweighed the benefits of the diagrams.

This situation led us to modify the developer guide documentation process. Instead of sketching designs before implementation, we wrote text-only descriptions of the designs immediately *after* implementation. This made it much easier to update the documentation and reduced the need to update because it was only the final implementation that was documented. The text-only descriptions were intentionally kept short and followed a format that roughly approximated the information provided by the sequence and class diagrams.

The SECO5 team was a small co-located team who were intimately familiar with legacy system. We used the OpenUP document templates as "checklists" to ensure we had not let any obvious issues fall through the cracks. While SmallEcommCo did capture specifications in a formal document, they made extensive use of a project wiki for sharing information. Many small teams share information using Wikis. The SECO5 Wiki was organized into the following sections:

- Customer Documentation – Consisted of the user manual, deployment guide, and developer guide.
- Requirements – The main document in this section was the Software Requirements Specification.

- Programmer's Handbook – An internal collection of patterns and guidelines for creating software.
- Quality Assurance – Documentation on the QA process as well as test cases and reports.
- Project Management – This section contained several documents including the Vision Document, Software Development Plan, iteration plans and retrospectives.

We also took advantage of the greatest software engineering tool developed to date, the digital camera. Much of our Wiki content was generated from photographs taken of white board sketches. The architectural notebook started as a digital photo of a sketch made on a white board and then evolved into a developer's handbook mostly as a catalog of patterns and guidelines for creating software.

Process Training

Several training sessions were provided by WSA to introduce OpenUP's fundamental concepts and terminology. All members of the software development team including developers, testers, and analysts attended the training. The advantage of whole-team participation during training was that quick agreements could be made by everyone for how an OpenUP practice would be specifically applied at SmallEcommCo.

Training is more than just a way to instill facts and knowledge, it is also a way to begin solidifying the team, to start giving them some shared experiences and shared vocabulary[5]. This is something many organizations don't take advantage of, sending individual team members for individual skills training rather than training the team together.

Throughout the project, continuous training was maintained as a standard practice. Developers attended a topical software engineering conference as well as an off-site database performance course. In addition, team members took part in internal presentations as well as several Webinars on various topics.

Communicating Methodology Changes to the Team

We did not use the EPF composer to modify OpenUP. Changes to the process and development practices were communicated either through e-mail or verbally during project meetings. Some of these agreements were added to the project Wiki.

Formal Meetings

"Meeting" is a term that is laden with considerable baggage. Most of us have been subjected to so many badly run and pointless meetings that we view them as either useless dog and pony shows or pointless irrelevant discussion groups and a waste of our valuable time. But a meeting is also an opportunity for social interaction when people come together, talk and share information and seek resolution to issues. Sawyer [6] noted the correlation between team performance and opportunities for team social interaction. A well run, focused meeting is a valuable tool for building the collaborative environment.

OpenUP does not prescribe specific team meetings but many of the tasks require such a meeting. At SmallEcommCo we held the following regular meetings.

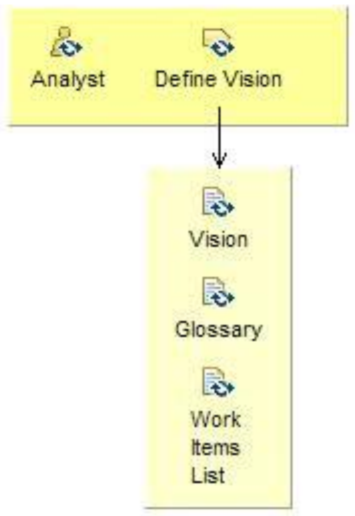
Iteration Planning	Before the beginning of an iteration, the project manager and product manager create the first draft of the iteration plan.
Iteration Kickoff and Retrospective.	At the beginning of an iteration, a kickoff meeting is held that begins with a retrospective of the previous iteration. After the retrospective, the iteration plan is presented , reviewed, and revised by the project team.
Iteration Assessment	At the end of the iteration, the iteration is assessed and changes are recommended.
Daily Scrum	- Daily stand up meeting
Design Meetings	- Regularly scheduled weekly meeting for tackling architectural issues.

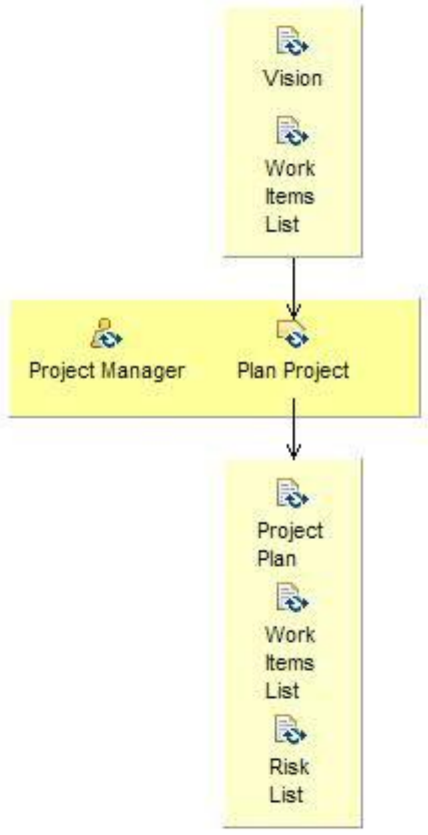
Some may ask why design meetings must be scheduled, why couldn't the team members come together when necessary? Such ad hoc design meetings did occur, but it was often difficult to get the right people together who were required to make a decision. In a small company people often have responsibilities outside of the development group. For example, the analyst also provided sales support. Therefore a regularly scheduled meeting meant we had a reasonable chance of getting the principle's together into a meeting where we cold commit to a decision.

Project Management Discipline

This section describes the major components of the actual process used during the SECO5 project. This process is comprised of a combination of concepts adapted from OpenUP as well as our own best practices. We describe our initial process as well as any lessons learned and resulting adjustments to the process.

Activity: Project Initiation





While project planning is ongoing activity in OpenUP, it all must start somewhere. When the project is initiated, the project manager defines a vision and creates a project plan. Within the *Initiate Project* activity there are two tasks, *define the vision* and *plan the project*.

Task: Define the Vision

OpenUP provides a template for a vision document that describes who the stakeholders are, system features, the constraints on the system and the system boundaries. Our initial vision statement at this point was one simple sentence:

"To make it much easier for our current and future customers to create and maintain sophisticated online stores."

We also defined several high-level goals based on this vision:

1) Enforce uniform development standards.
2) Use rule engine technology to give users more control over business rules.
3) Provide flexible mechanisms for external systems to remotely exchange information with the system.
4) Make it easy to deploy and manage SmallEcommCo in a clustered environment.
5) Unit test extensively to make it easier to verify that future enhancements and modifications are correct.

Further guidance was provided by WSA in the form of mantras that reminded everyone of the project's scope and the time pressure we were under:

1) Time is not our friend.
2) Spend a dollar to save an hour.

In our opinion, a small co-located team does not require an extensive vision statement and unless a company is operating under a coercive quality standard. A small co-located team only needs something to align their respective visions of what the project is about, something that can be reduced to a mantra. A formal vision statement is more for the consumption of external stakeholders, especially stakeholders who may be distant from the project and/or have corporate governance requirements they must satisfy. This was not the case at SmallEcommCo so a simple vision statement was adequate. In a similar environment to SECO the perceived need for an extensive vision statement should raise some significant red flags.

Task: Plan the Project

OpenUP describes this task as "A collaborative task that outlines an initial agreement on how the project will deliver the product vision." The output of this task is a project plan, a risk list and work items for the work items list.

One of the most important project planning steps is estimating the size of the project. How much work do we have to do? How long will it take? How much will it cost? These are the questions management wants answered before they will open up the corporate wallet and fund the project. To answer these questions and ensure management was aware of the uncertainty in our estimates, we produced a range of estimates and labeled the end points "low" and "high." This was used to explain the range of possibilities to the management committee. We provided management with several scenarios based on different sets of assumptions and demonstrated what cost drivers could push the project from a low estimate to an estimate that was three times higher.

One mistake many practitioners commit is trying to make early schedules appear more precise than is warranted by the available data. We nicknamed our estimating method SWAG for Scientific Wild Ass Guess because that is exactly what the estimates were: our best guess. At this early time in the project we only had two sentences for the vision statement and an empty work item list. This is very little information for creating a project plan. Fortunately we were able to fill in the blanks by exploiting expert knowledge. This was a legacy replacement project and the people involved were experts in both the legacy and the problem domain. We made our estimates using the legacy system as an analogy.

Using this sketchy data and drawing on our own experience, we then built an equally sketchy project plan showing the length of each phase (inception, elaboration, construction, transition) and their expected milestone dates (Life Cycle Objective, Life Cycle Architecture, Initial Operational Capability). No formal project plan document was created, rather, the project plan was presented as a power point presentation. A strong recommendation we have when presenting such a plan is to draw it free hand and scan it. The hand drawn wavy lines will dramatically demonstrate the uncertainty associated with the plan. We knew immediately that our plan was terribly rough, but as Eisenhower stated “plans are useless but planning is everything.” We knew that the data gathered from our first iterations would help us refine the project plan.

What our plan did not describe is what each of the 9 initially planned iterations (we actually ran 12) had to achieve or what features were assigned to those iterations. A common mistake made when doing iterative planning is assigning features in advance to every iteration. The point of iterative development is to assign features to the next iteration, and at the end of current iteration review and re-plan. However, there is nothing wrong with looking ahead and proposing features for future iterations as long as you are willing to re-organize the plan in response to feedback. As the project progressed we started making intelligent guesses about which iteration a task may be assigned to.

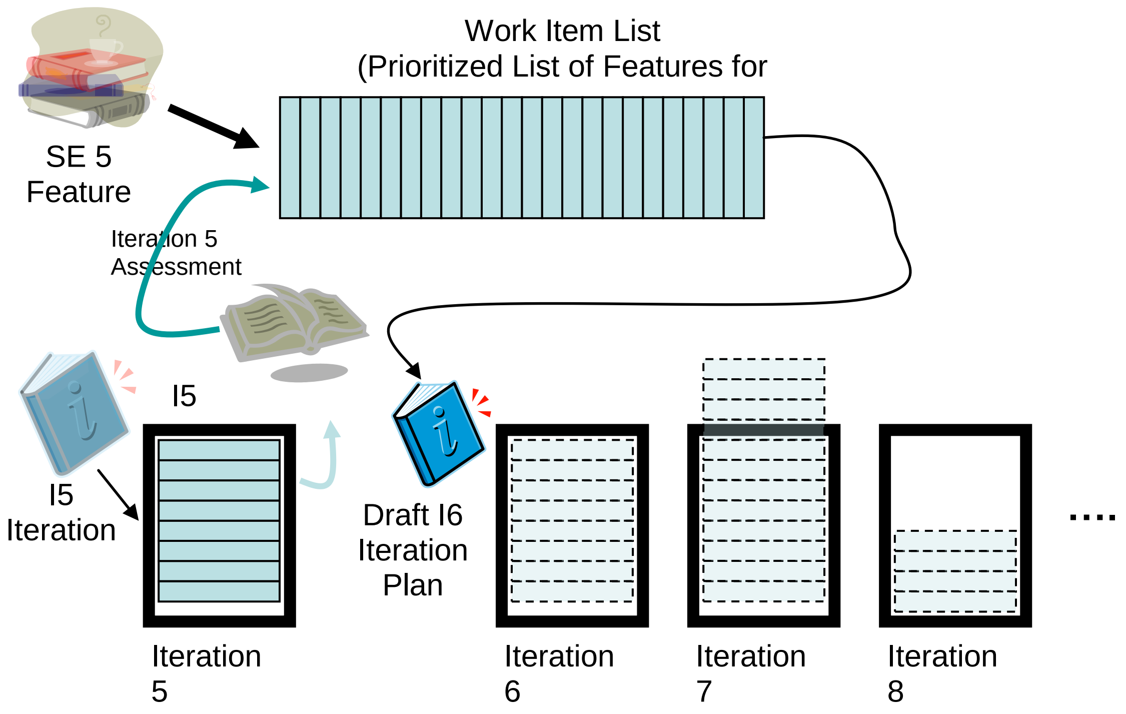
How long is an Iteration?

An important project parameter is choosing the iteration length. A general principle is a larger team requires more time to start up and wind down an iteration. Some people prefer short iterations such as one week, others prefer longer iterations of four weeks or more. We arbitrarily chose three weeks. Why three weeks? Why not 2 weeks? Why not 4 weeks? Like so many decisions, this one is very much visceral, three weeks “felt right.” This was a new methodology for SmallEcommCo, so there would be significant overhead as people became familiar with the methodology which suggests a longer iteration. But we wanted rapid feedback and to keep the team focused on their objectives which suggests a shorter iteration. There was also a political dimension to this choice. Most members of the project team would not be comfortable with a shorter iteration. One must remember that not all (in fact very few) project parameters are driven by technical consideration.

The Work Item List

The work item list constitutes a backlog of all scheduled work to be done within the project. The purpose of the work item list is to collect all requests for work that will potentially be taken on within the project, so work can be prioritized, effort estimated and progress tracked. Each work item may contain references to information relevant to carry out the work described within the work item.

We represented the work item list by using half-sheets of paper taped to a wall. Vertical sections of the wall represented iterations and the backlog items were roughly allocated to an iteration by placing them under the corresponding iteration label.



This low-tech planning approach provided excellent visibility of approximately when a specific feature was to be implemented as well as which features were at risk of being dropped from the project. This approach also made it easy to move features between iterations and quickly see the impact on the project.

Risk Management

If you don’t actively attack the risks the risks will actively attack you. Risk management is integral to OpenUP and a step in project planning the project is creating a risk list. We chose to package the risk list as part of the iteration plan because this approach would provide higher visibility to the risks. At each iteration kick-off, the project manager reviewed each risk with the team. Because this list was reviewed and reordered at each kickoff, we had the talents of the entire team contributing to mitigation strategies and monitoring the risks as they became less of an issue and eventually dropped off the list. Of course, new risks were also identified and added to the list.

Sample Risk List

Top Risks:

1. Various dojo-related issues need to be resolved before product can ship – namely, excessive load time for dojo library - BLOCKER.

   Strategy: Dojo 0.4.0 will be released sometime in October, and Vendor v will keep us updated once the release date nears. Developer W will continue tuning in I10, as per B from Vendor v feedback.

2. The Preview Release will need to run on Derby http://db.apache.org/derby/ as Hibernate does not support HSQL, which is an in-memory database that we currently haven't tested thoroughly on.

   Strategy: It is supported by Hibernate and Developer D has done a quick smoke test on his PC so it looks promising, but we need to do a lot more testing on this DB before it ships. QA will be focusing time on testing this in I10.

3. The Dev Guide and User Manual documentation is a substantial amount of work that still needs to be completed.

   Strategy: In I9, a plan for completion will be put together for both documents so that it is managed over the next two iterations. Developer D will be going to the developers and handing out portions of work to be completed for the dev guide.

4. There is a lack of automated feature testing and regression testing in QA on all platforms.

   Strategy: We need to stabilize the SF UI and get THIRD PARTY data before we can start to do this automation.

5. There has not been testing or performance testing on a clustered environment yet for all platforms.

   Strategy: Developer D will focus on testing a cluster on all 4 application servers and then documenting his findings during I10 (it has only been done on TomCat so far, and configured on JBoss). .

6. Performance for the architecture we are building may not be good enough, for both the SF and CM.

   Strategy: We need to keep a close eye on performance testing. This is our main goal for Iteration 6 and will continue throughout the rest of the iterations. We need to ensure that performance for the SF is at least equal to that of 4.1 by the time we ship the product.

7. AJAX Screen Feature / THIRD PARTY may be an impact on the SECO 5.0 team starting in I7 (Developer D. and Developer E. Time may be greatly impacted).

Strategy: Need to determine the impacts on the team and make sure to incorporate this into iteration planning. A design session will be planned with Developer D. in determining how to make this demoable for analysts, customers, etc. for Sales.

11. Missing information in terms of business rules and workflow for items like Promotions, Checkout, Shipping, and Taxes for the Store Front for the QA team to test

    Strategy: This will be worked on in I9 by Developer D as he is finishing up the SF integration and updating the SF use cases.

The format of this risk list is simpler than that of the template provided with OpenUP. We did not see the need for a more precise classification of the risks (e.g. by type) because we did not believe this added any useful information. The close social structure of the team would help individuals have a common understanding of the nature of the risk. We also chose to rely on the expert judgment of the team members to prioritize the risk list rather than explicitly calculate impact and probability. Risks were revisited at the iteration kickoff and iteration assessment meetings.

A common mistake many small teams make is either they ignore the risks, or assume that creating the risk list that somehow the risks will magically disappear. Risks must be dealt with and this is why presentation of the risks were front and center of all iteration assessment and iteration planning meetings.

Project Kickoff

Like "Meeting," "ceremony" and "ritual" are terms laden with unwanted baggage. Many people often use ceremony as a derogatory term to imply a wasteful activity. However, some ceremony is necessary because effective ceremonies and ritual bring people together and provide a shared experience that inspires harmony in a team. Opportunity for the team and the stake holders to interact is an important driver for success. Sawyer and Guinan [6] demonstrate that social interaction was a predictor for variances in the effectiveness of software development teams.

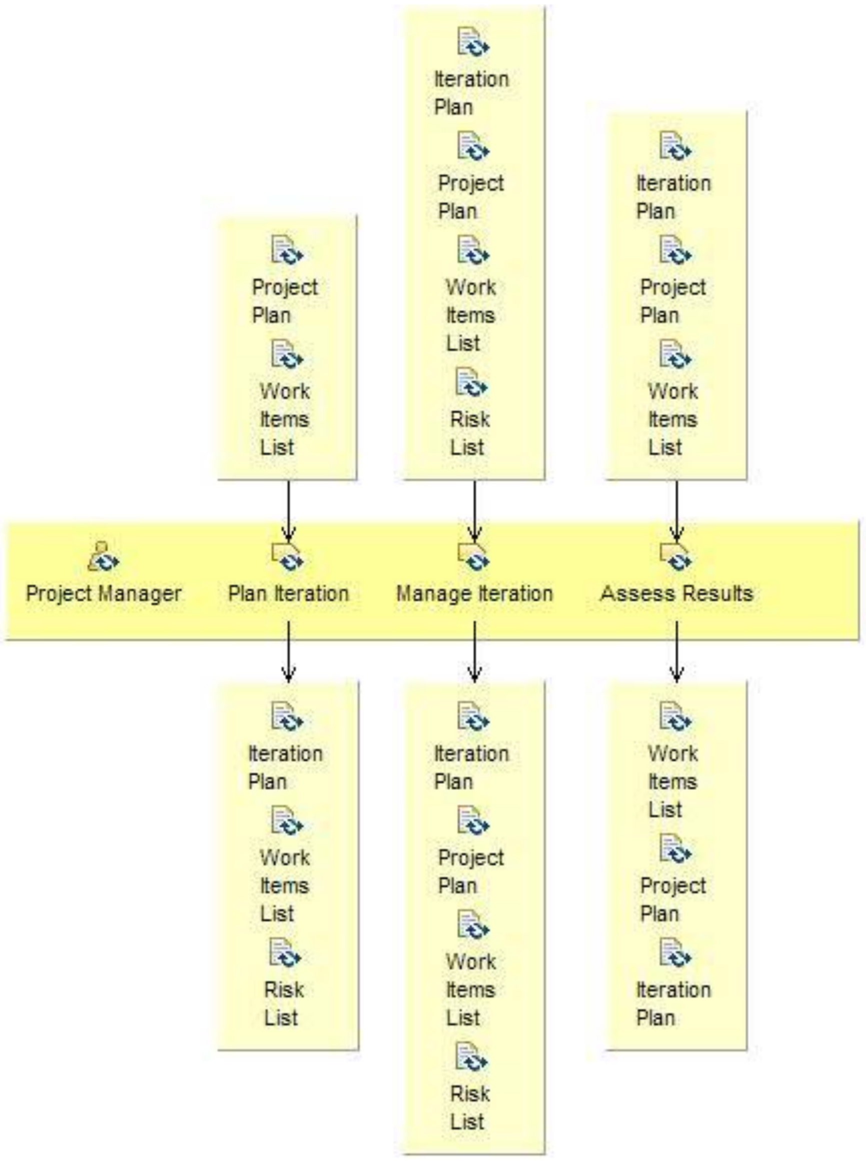
A project kickoff meeting is not something explicitly prescribed as part of the OpenUP project initiation process. But for any project with more than two people we strongly believe a kickoff is essential to bring the entire project team together and explain vision for the project. This experience begins laying the foundation for turning a group of like minded individuals into an effective team. Collaboration is an OpenUP core principle because we believe collaboration is essential for a team to work effectively. Collaboration only emerges in a healthy team environment and we believe bringing people together to share the project vision is one practice which encourages a healthy team environment.

We were fortunate the project was small enough that the entire project team could sit together in a large conference room. This included all disciplines: project management, developers, testers, analysts and stakeholders. During this meeting we introduced the project vision and the first draft of the project plan. We also provided a brief overview of the project methodology and why we chose this methodology.

The Cookie Theory of Meetings

Something that is often overlooked in meetings is the provision of food for meetings. Nothing brings people together faster than sharing food. All our iteration kickoff meetings were catered. It does not have to be expensive, some decent pastries and for the more the health conscious a bowl of fruit will suffice (strawberries were especially popular at SmallEcommCo). To paraphrase the credit card ad campaign: “Team goodwill – priceless”

Activity: Plan and Manage Iteration



OpenUP describes this activity as "Initiate the iteration and allow team members to sign up for development tasks. Monitor and communicate project status to external stakeholders. Identify and handle exceptions and problems." The project manager has primary responsibility for this activity and has three tasks, plan the iteration, manage the iteration, and assess the iteration.

Task: Plan Iteration

Iteration planning is the fundamental process that drove the project. This activity occurred in advance of a given iteration and the plan was presented at the iteration kickoff meeting. The end result of iteration planning was, of course, to produce an iteration plan.

The iteration plan represents the work a team can accomplish in one iteration. It is not a target to be met, such that when the team fails to meet the lofty targets they are derided and demoralized. Rather it is a realistic statement of work that given the talents of the team and the resources they have available to them, what can they realistically accomplish during the iteration. There is nothing wrong with wishful thinking and setting targets so long as we acknowledge that this is what they are. But for effective management of a project, the plan must reflect reality.

Setting Iteration Objectives

The iteration planning exercise begins with the objectives for the iteration. The objectives are high-level statements describing what is to be completed during an iteration. This essentially summarizes the highest priority tasks or features that have been selected from the project backlog for implementation during the next iteration. The iteration objectives became part of the iteration plan presented to the team during the iteration kickoff.

The following is a list of objectives from one of the later iterations in the project.

- Complete priority bug fixes (majors and above) for a 5.0.1 release to the public by Friday November 24th
    - Builds will be provided, as necessary, for QA
    - An Installer Release build will be completed every Friday for testing on Derby
- Complete implementation of Featured Products in the Commerce Manager for Customer C requirements and testing
- Complete integration of SmallEcommCo feature-x items, and subsequent documentation for a public release and testing
- Respond to support requests, and update User Manual, Developer Guide, and Deployment Guide, as necessary to reduce expected product support requests
- Flush out the Developer Guide with more content and tutorials
- Cluster testing on various platforms to be completed
- Gather estimation figures for upgrading to dojo 0.4.0 and 5.1 use cases
- QA to test Featured Products, Feature x, and complete feature testing on WebLogic, Websphere, and JBoss (with smoke testing only on Tomcat)

Estimation

Work was estimated by developers in the days leading up to the iteration in which the work was scheduled. It was the developer who was assigned the work that would estimate how long it would take. If the work was subsequently assigned to another developer, that developer would then re-estimate it. This allowed developers to adjust times based on their experience and familiarity with the task. Furthermore, this provided a mechanism for developers to refine their estimation capabilities because reports were generated that compared estimated and actual times for each major task in an iteration.

Estimating techniques were not mandated, but developers often broke their tasks down into subtasks in order to more accurately determine how long it would take. Having a process in place gave the developers a reality check on their estimates because process made them aware of all the tasks required to "develop the solution" that creating a work product required more than simply creating code. Developer estimates included items such as:

- Detailed design
- Implementation
  - Domain object model
  - Database model
  - Service layer
  - User interface
- JUnit testing
- Performance testing and tuning
- Buddy Reviews
- Documentation

Task Selection and Assignment

The work items under the next iteration on the wall provided the high-level tasks to be scheduled into the next iteration being planned. The next step was to prepare a lower-level assignment of specific sub-tasks to individual developers. The sub-tasks initially were assigned to developers based on developer preference and expertise in a given area. The next step was to ask the developer to estimate the amount of time required for the assigned tasks.

We used hours as the measure of development effort required to complete a task as well as a measure of how much work could be done in an iteration. While it is popular to use abstract points to measure the size of an effort, our personal observation is that, left on their own, a many people simply work out some simple multiplier to convert points to hours.

A three week iteration has 15 work days. From these 15 days we automatically held back 10% (2 days) for the usual overhead of administrative issues, bad days, sick leave, server crashes, and all the other innumerable petty circumstances that make straightforward tasks difficult. This is not slack time, buffer or even giving permission to the team to "sand bag" their estimates. It is an acknowledgement of the inevitable small problems that will be encountered. This unfortunate fact of life can either be acknowledged and explicitly managed, or you can ignore it and watch it pop up in unpredictable places.

If staff were on vacation then, their days were removed from the pool. This would give us development capacity for the iterations (staff*days). Tasks were broken down into half day granularity. Tasks longer than 4 days were decomposed into smaller tasks. Tasks were assigned to the iteration from the work item list in priority order until the iteration was filled. This gave us an initial iteration plan. This activity was usually performed by the project manager, lead analyst, and architect. This may seem contrary to agile principles of self organization, but it enabled the staff to not be interfered with until the planning meeting in which they could re-estimate the task and alter the assignments.

One way to view this is the project manager engaged in a pre-planning exercise, and along with the architect and analyst acted in the role of an onsite customer selecting the highest priority features for this iteration. The early assignment of tasks was based on conversations the project manager had had with developers. When the iteration plan was presented to the team, the team could “push back” or suggest revisions to the plan. There were many instances of a team member suggesting a lower priority activity be added to the current iteration because it would make work easier in the long term. Developers were then asked to make final estimates on the work items assigned to the current iteration. This is more in the agile spirit of self organization.

Once the developer estimates were available and developers were satisfied with their assignment, a final adjustment could be made to the iteration plan so that the estimated development time matched the available development time.

The Iteration Plan Document

The SECO5.0 Iteration Plan document followed the template distributed with OpenUP with several additions:

- We included a copy of the risk list in the iteration plan.
- We showed how resources were calculated and who was going on vacation and when during the iteration (and also statutory holidays).
- Buddy Assignments – who would work with whom during the iteration.

- Implement SF – Payment Options
    - PayPal – X days
    - Authorize.Net – X days
    - Verisign – X days
- Implement CM – SKU X days
    - View Product SKU X days
        - service layer + junit X days
        - UI X days
    - View SKU X days
        - service layer + junit X days
        - UI X days
    - Create SKU Wizard X days
        - service layer + junit X days
        - UI with wizard X days
- Implement Rules engine basic traceability
  X days

- Create AppliedOrderRule Domain Object (0.4)
    - Domain Object, DB table, Unit tests, Hibernate
- Update CheckoutService and test cases (X)
- Update Shopping Cart domain object + test cases (X)

- Update the order object, hibernate and test cases (X)
- Add a description to each rule and write the descriptions (X)
- Update rule code for each rule to pass in the rule UID (X)
- Update each rule action (in delegate) to retrieve the rule and pass to cart + test cases (X)

- *Documentation for dev guide* *X day*
    - *Rules* *X days*
    - *Other (from Dave)* *X days*
- *Merge multi-language support of Country Names files from Justin* *X day*

- Implement CM Use Cases X-x days
    - CM – Create Category x-x days

- Implement CM Use Cases: X day
    - CM – Asset Manager X days
        - Asset manager dialog display and layout (X days)
        - Asset manager and upload logic (X days)
        - Asset tree widget (X days)
        - Local file selection box assuming use of an OS dialog (X days)
        - Implement asset controller (X days)
        - Modify CM and SF to use asset manager to load all images (X days)
    - CM – Edit Product Image X days

The iteration plan was a comprehensive document which the project manager went through with the entire team during the iteration kick-off. The purpose of a comprehensive document was to ensure everyone knew what was going on, who was doing what, the constraints, and reminded of the project risks.

From an agile perspective, it could be argued such a planning document is superfluous or even an indication of a less than agile process. That in an effective agile team the knowledge represented by the iteration planning document should be communicated tacitly and by using parking lot diagrams. In retrospect, the use of parking lot diagrams for monitoring tasks would have worked well. However, this misses the point, we used the iteration planning document as a tool to ensure all members of the team knew what was expected of them, and of their colleagues during the iteration. We will not argue this was the only way to accomplish this task. However, we believe it is necessary that everyone understands the iteration.

Task: Manage Iteration

This is an ongoing task where the project manager is responsible for assessing project status and identifying and managing any issues that arise. These issues include exceptions, problems, risks, and opportunities. The project manager is responsible for communicating project status and managing stakeholders' expectations.

An important tool for performing this task is the daily Scrum meeting. The daily scrum at SmallEcommCo was held each morning in the main area where developers worked. Team members stood in a circle and took turns explaining what they contributed the previous day, what they were to work on today, and any items that were blocking them. It was often the case that team members couldn't resist problem solving during the scrum and it became everyone's responsibility to ask people to take issues “offline” when problem solving went on for more than a minute or two.

Task: Assess Iteration

Iteration Assessment Meeting

Performing retrospectives is one of the most useful tasks in the development process. Retrospectives were held after each iteration, just prior to the iteration kickoff. Given the relatively small size of the team, it was possible to go around the room collecting feedback on what worked, what didn't work, and what could be done to improve productivity. This allowed us to constantly tweak the process, discarding practices that did not add value, adding new practices, and discover barriers to making rapid progress.

This process also started to build trust because people saw their ideas were valued and they had ownership of the software development process. They also began to see that it was safe to express their dissatisfaction with the way things were progressing or not progressing and seek solutions to problems they were experiencing.

Internal Release and Customer Feedback

Not only did we assess the development process but we also presented the evolving product to management and other stakeholders for their assessment. The three week release cycle forced convergence and developers worked together to ensure that the product was stable and ready to demo at the end of each iteration. This process bought the development team a great deal of credibility with the C-level management because at the end of each iteration there was proof the system was growing and evolving.

Requirements Discipline

On the surface, the requirements for this project should be fairly stable and well known. This was a legacy replacement project and our mandate was not to alter the functionality of the system. However, business realities dictated that it was necessary not only to replace the existing system but also to dramatically improve upon it. Furthermore, it was counter productive to be using new tools like Ajax to re-implement features in a JSP/Struts style. Therefore, it was often the case that new requirements were defined for features being developed.

Task: Identify and Refine Requirements

Requirements were captured in a just in time manner with use cases and screen shots of Visio mockups. Early during the system planning we entered cards for features into the work item list. When it came close to time for implementing that feature, an analyst would create a use case or portion of the use case required. The collection of use cases and screen shots became the Software Requirements Specification (SRS). This SRS was maintained on the wiki so that it could be easily updated over time. However, team members often found it too tedious to adjust the Visio mockups and update the screen shots.

Brand (RQF-CM-BRAND-UC1)

PRECONDITIONS:

1. CM User with Role to use Admin section is logged into the Commerce Manager.

MAIN SUCCESS SCENARIO:

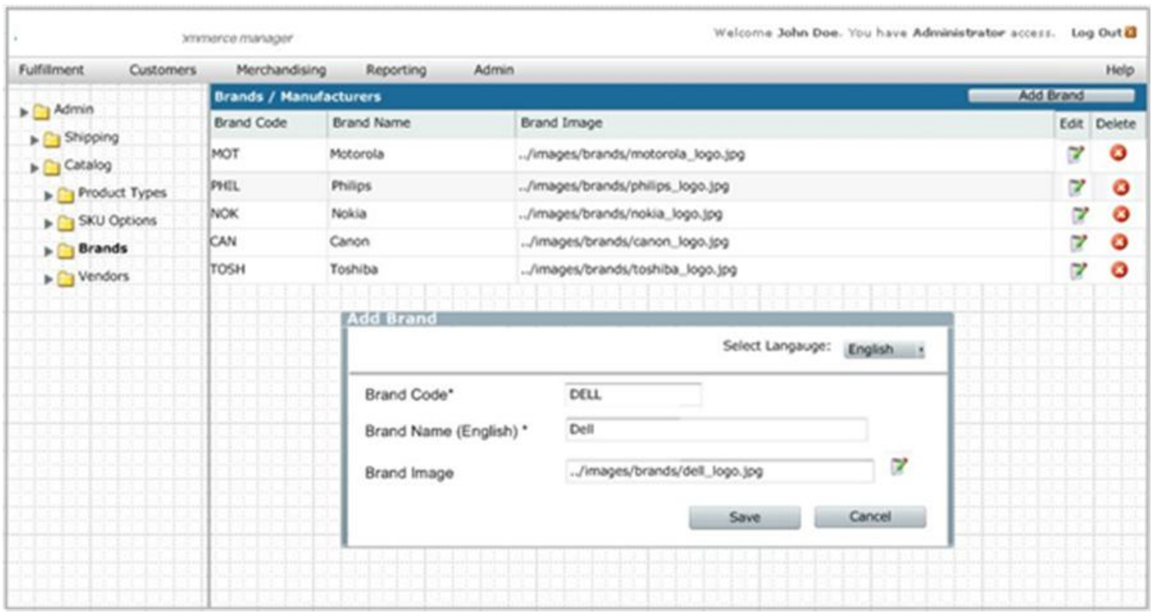
1. CM USER: Selects Admin - Catalog - Brands section of CM
2. SYSTEM: Displays list of all configured BrandsBrand (RQF-CM-BRAND-S2)
3. CM USER: Selects 'Add Brand'
4. SYSTEM: Opens popup "Add Brand" window (in modal mode)
5. CM USER: Enter's required information.
6. CM USER: Clicks Save
7. SYSTEM: Saves new Brand to the database
8. SYSTEM: Refreshes list of Brands Brand (RQF-CM-BRAND-S2) and closes popup window.

ALTERNATIVES:
2a. SYSTEM: No Brands are configured. System displays messgage "No Brands"
3a. CM USER: Selects "Delete Icon"
3a1. SYSTEM: Deletes the Brand from the database
3a2. SYSTEM: Refreshes list of Brands Brand (RQF-CM-BRAND-S2)
3b. CM USER: Selects "Edit Icon"
3b1. SYSTEM: Opens popup "Edit Brand" window (in modal mode)
3b2. CM USER: Changes name, and / or image
3b3. CM USER: Clicks Save
3b4. SYSTEM: Updates Brand to the database
3b5. SYSTEM: Refreshes list of Brands Brand (RQF-CM-BRAND-S2) and closes popup window.

SUPPLEMENTARY SPECS:
1. If a Brand is in use (i.e. mapped to a Product) then the delete icon is greyed out / disabled.
2. Brand Code is unique. Only one Brand with the same code can exist.
3. Only the "Brand Name" is langauge dependant. Only the default language is mandatory.
4. When editing an attribute the following CANNOT be changed/edited:

- Brand Code

NOTES:
Brand is also known as Manufacturer. This should be documented in the CM user guide Glossary.



Brand Code	Brand Name	Brand Image	Edit	Delete
MOT	Motorola	../images/brands/motorola_logo.jpg		
PHIL	Philips	../images/brands/philips_logo.jpg		
NOK	Nokia	../images/brands/nokia_logo.jpg		
CAN	Canon	../images/brands/canon_logo.jpg		
TOSH	Toshiba	../images/brands/toshiba_logo.jpg		

Select Langauge: English

Brand Code*	DELL
Brand Name (English) *	Dell
Brand Image	../images/brands/dell_logo.jpg

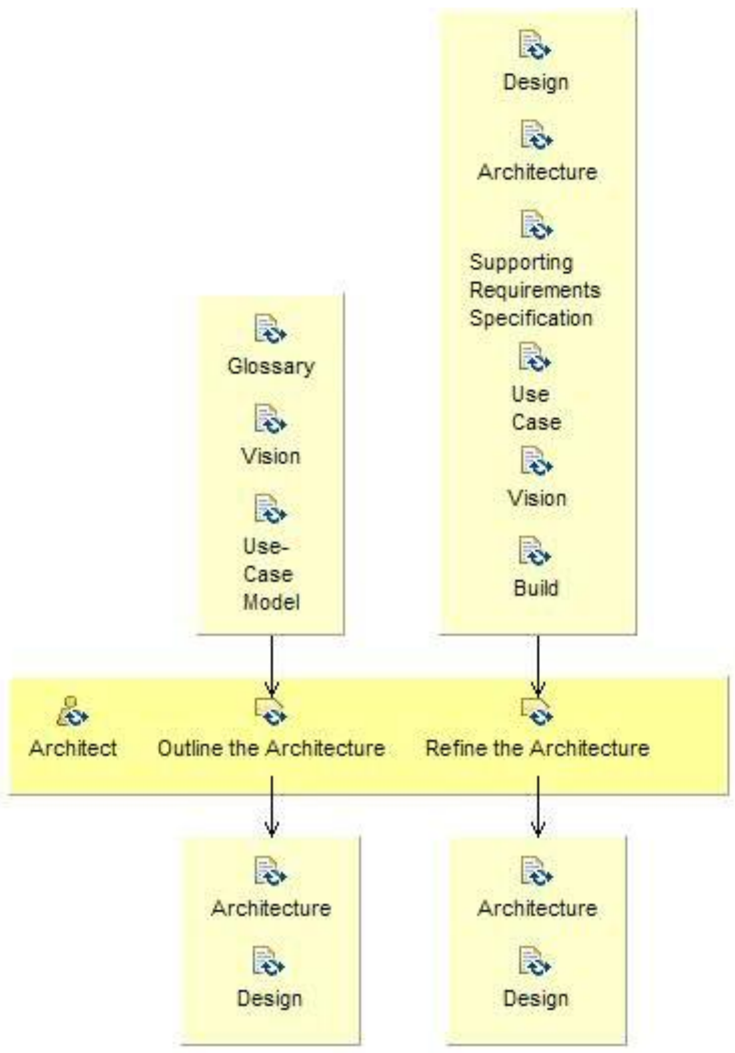
Requirements elicitation mostly consisted of experienced developers acting in the role of analyst creating use cases for existing system behavior. However, new work often came in the form of new screen layouts. New technologies such as Ajax and Lucene gave SECO5.0 better interactivity and visual appearance than its predecessor. While we had a mantra if it’s not in the legacy then it’s not in SECO5.0, it seemed a sin not to take advantage of the new technology.

Architecture Discipline

Software architecture is a concept that is easy to understand, and that most engineers intuitively feel, especially with a little experience, but it is hard to define precisely. In particular, it is difficult to draw a sharp line between design and architecture – architecture is one aspect of design that concentrates on some specific features.

There are some who suggest an explicit architecture activity is unnecessary because architecture emerges through continuous and aggressive refactoring. Our personal experiences do not support this point of view. In the SmallEcommCo case the architecture for on particular UI component was somewhat hastily assembled mid-way through the project and did not receive sufficient review. This resulted in a subsystem that offered comparatively low software qualities such as performance, scalability, and maintainability. The end result was that it was necessary to re-architect this component.

This experience led us to adjust the development process to provide more emphasis on architecture on an ongoing basis. A small architecture committee was appointed to take responsibility for the product architecture. This committee met regularly to review architectural issues, develop standard implementation patterns, and determine if there was any major refactoring that should be brought to project management's attention for scheduling into an upcoming iteration.



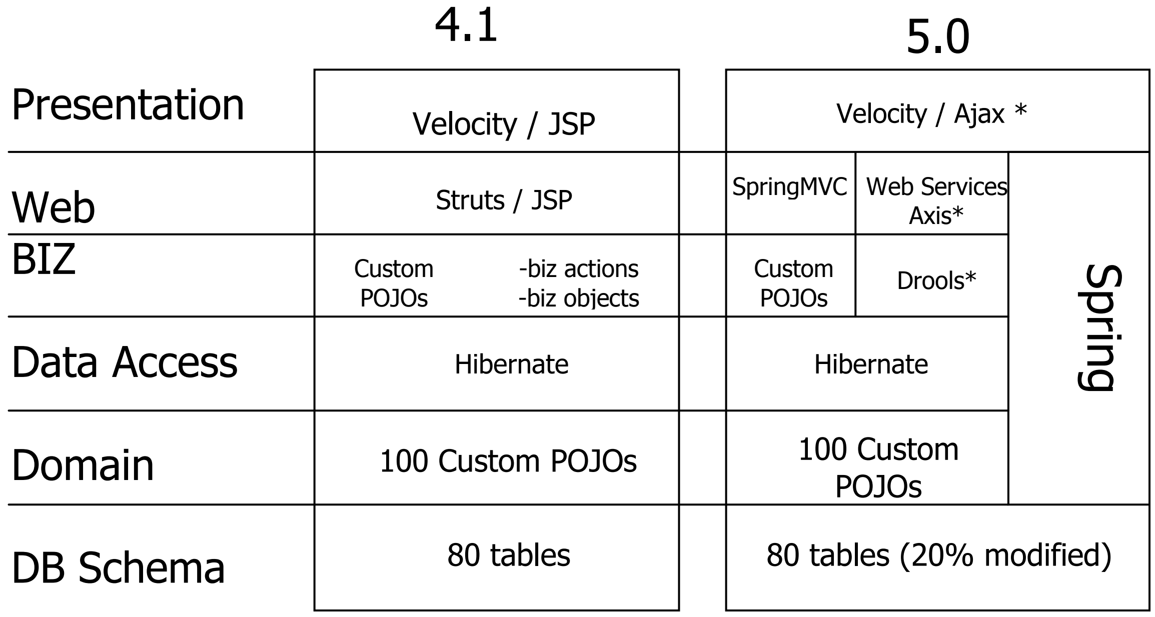
Task: Outline the Architecture

This task focuses on identifying the architectural goals for an iteration that will guide development and testing. It relies on gathering experience gained in similar systems or problem domains to constrain and focus the architecture so that effort is not wasted in re-inventing architecture.

A key goal of the project was to migrate to an architecture that would easily support development of a feature-rich application. The architecture for the majority of the new system was actually developed during an R&D project that began over 6? months before the actual project kickoff. By the time the project officially began, a well-defined architecture was in place along with documentation of the component structure and design conventions

The initial architecture for the system was a sketch on a white board demonstrating the difference between the legacy architecture and the desired new architecture. This diagram was photographed and placed into the wiki becoming the first page in our architectural notebook.

The diagram may seem trivial, but its importance cannot be overstated because it set a clear architectural vision for the entire team. This simple diagram identified layer’s and policies for the implementation of those layers. The interface between these layers and patterns of interaction between the layers became part of the evolving developer’s handbook (our implementation of the OpenUP note book).



	4.1	5.0		
Presentation	Velocity / JSP	Velocity / Ajax *		
Web	Struts / JSP	SpringMVC	Web Services Axis*	Spring
BIZ	Custom POJOs -biz actions -biz objects	Custom POJOs	Drools*	
Data Access	Hibernate	Hibernate		
Domain	100 Custom POJOs	100 Custom POJOs		
DB Schema	80 tables	80 tables (20% modified)		

.

Task: Refine the Architecture

This task builds on the work performed during Task: Outline the Architecture. The objective is to make the architectural decisions necessary to support the objectives for the current iteration of the project. The decisions taken as part of this task are concrete and unambiguous. They are captured in the Architecture Notebook and communicated across the team.

There was no explicit “refine the architecture” task that appeared in an iteration plan. Refining the architecture was part of the ongoing development task for each developer. Developers were assigned a feature and worked with an architect to create this feature. If a developer were creating a new architectural mechanism then they were responsible for updating the Architecture Reference – our equivalent of the Architectural Notebook.

This said there were a number of scheduled design sessions where senior designer collaborated to resolve upcoming architectural issues such as portal integration, security and authentication strategy, web service architecture, and object caching design.These were often where technology implementation decisions were made such as the use of Dojo.

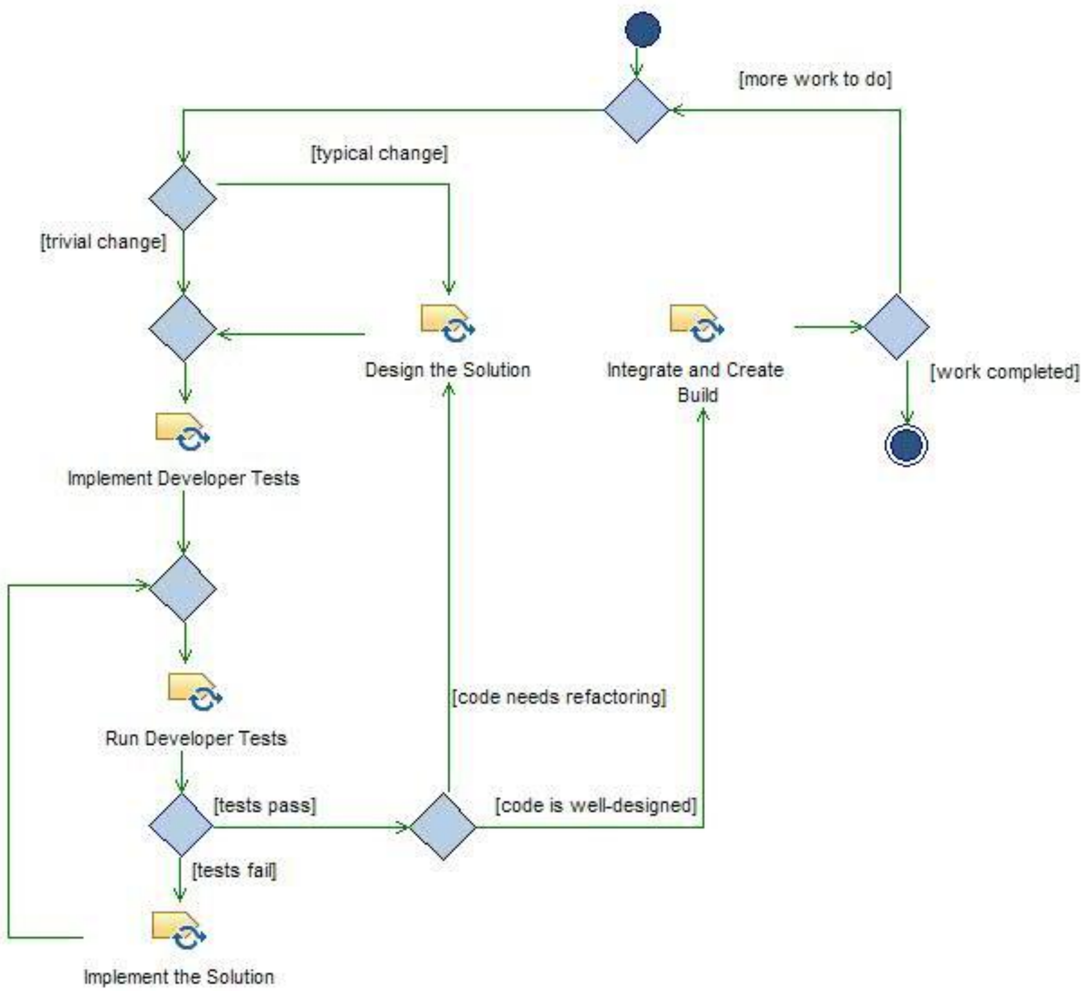
Development Discipline

The purpose of this discipline is to:

- To transform the requirements into a design of the system-to-be.
- To adapt the design to match the implementation environment.
- Incrementally build the system.
- Verify that the technical units used to build the system work as specified.

With each iteration, the tasks in this discipline will evolve an ever more capable and ever more stable Build.

Task: Developing the Solution



It was assumed that when a development task was assigned to a developer, they were undertaking the following tasks:

- Refine the Architecture
- Design the Solution
- Implement the Solution
- Implement Developer Tests
- Run Developer Tests

A development task was fairly encompassing, the developer may have to work with an architect to make architectural refinements while designing the solution. The developer worked with a buddy, coded, reviewed, and created unit tests. Most of the developers had experience with the legacy system and did not need much further clarification.

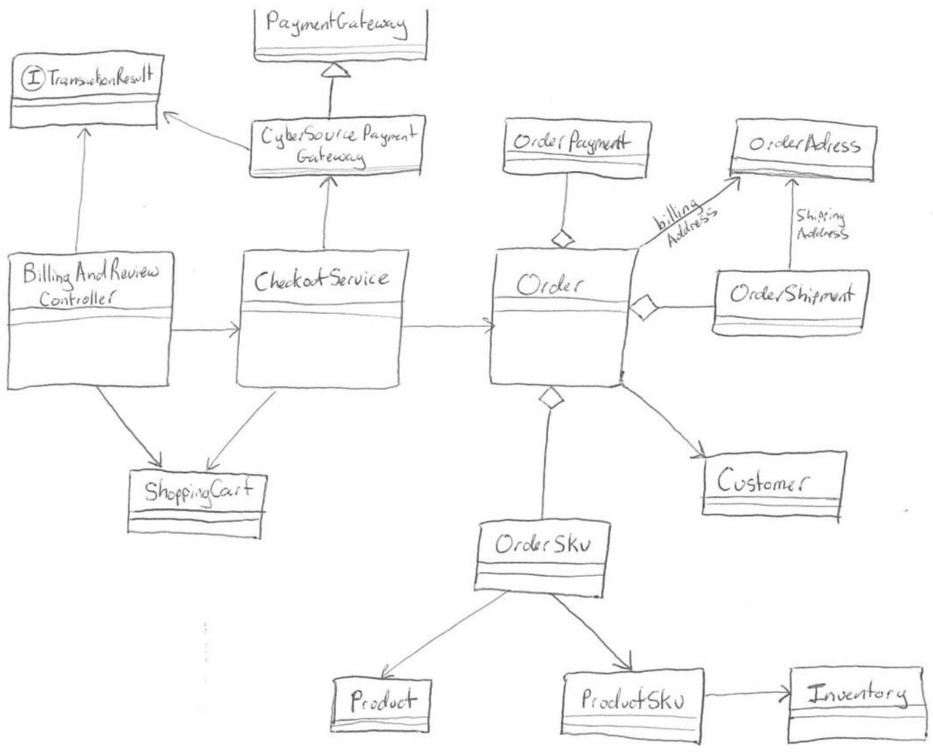
The development task is where most of SmallEcommCo's existing software development processes were assimilated. There were existing coding standards, version control standards, etc.

Design

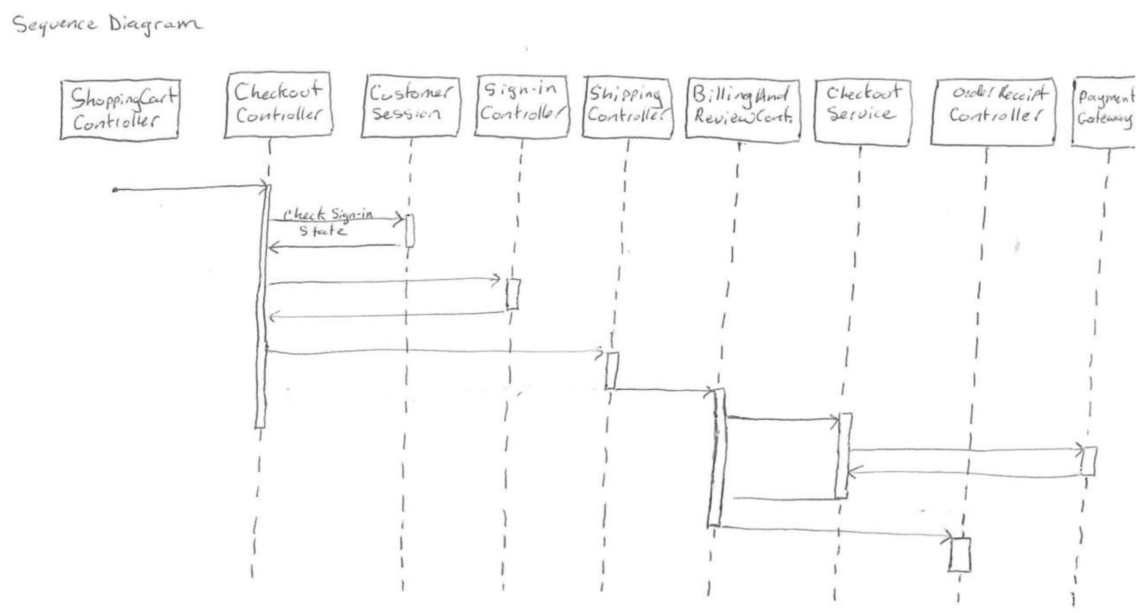
Feature designs often reflected architectural patterns that were set out during the R&D project that preceded the kickoff of SECO5 development. When new designs were needed that were not specified by the architecture, developers would typically sketch their design ideas on paper or a whiteboard and have them peer reviewed before beginning implementation. These designs were most often expressed using boxes and arrows roughly approximating class diagram notation. Often, several iterations of these sketches were produced before arriving at the final design.

Initially, the paper design sketches were scanned and uploaded to the wiki. However, we found that the design often required updating as development progressed and it became too tedious to update the wiki. In later iterations, developer-level documentation was written for each feature only after the feature was completed. The initial design sketches were discarded as they were invariably out of date. The developer-level documentation was intentionally kept minimal and typically consisted of a brief introduction and what was essentially text representations of class diagrams and sequence diagrams.

Sample diagrams



Checkout (RQF-SF-BC-UC5)
Class Diagram



Coding Standards

One of the goals of the project was to develop a codebase that is easy for customers to modify. One way to contribute to this goal is to maintain consistent coding standards across the system.

SmallEcommCo already had existing coding standards and these formed the basis of a developer's handbook document. This document was then extended to contain additional standards and guidelines that were specific to the new technology and architecture of the system being developed.

Although some coding standards are best captured in a handbook, other coding practices can be automatically checked and enforced at the source code level. We primarily used the Checkstyle and PMD tools for this purpose. Checkstyle focuses on coding style conventions such as consistent use of white space, parenthesis placement, and Javadoc comments. PMD checks for potentially problematic code such as `if` statements that are too deeply nested or methods that are too long.

These tools use pre-configured sets of “rules” by default. These rules offer a great starting point, but we found it necessary to customize them to better suit our internal development conventions and prevent them from becoming an annoyance. For example, it was necessary to remove all rules that were capable of producing false positives. These tools are not useful if you must ignore their feedback because it may not be correct. We also removed several rules that we simply couldn’t live with because they were too restrictive.

By integrating these tools with the build process (via Ant tasks and Cruise Control) we were successful in enforcing consistent coding standards throughout the system. If any violations were detected by the build, the offending developer would need to correct the problem. By using IDE integration provided by the tools, violations of coding standards could be reported immediately to the developer as they worked, thus avoiding the need for the build server to catch the problem.

Buddy Reviews

All developers worked with a buddy who had secondary responsibility for the task. We described this as pair programming lite. The buddy teams could work together in any manner they found effective. There were only two obligations on the buddy:

- They had to become familiar with the solution. This was to lower the so called “truck number” on our project.
- They had to approve of the solution. This ensured that there was always a second set of eyes looking at a solution and avoiding the programmer in the back room syndrome.

Code Reviews

We attempted to hold weekly code reviews as well as reviewing code as a step in developing new features. However, developers are busy people and the productivity gains resulting from code reviews are not realized immediately. This makes it tempting to postpone code reviews and we believe that more time should have been devoted to them throughout the project.

Although we could have performed more code reviews, a significant amount of code was reviewed. Developers found that this review provided several benefits:

- Development expertise was transferred to less experienced developers
- Knowledge of specific areas of the codebase was communicated so that others could confidently work in those areas
- Developers became more aware of emerging coding conventions
- The system was developed in a more consistent way
- Ways to improve code quality inevitably arose from the reviews

Unit Testing

Developer testing is different from other forms of testing in that it is based on the expected behavior of code units rather than being directly based on the system requirements.

It is best to do this at a small scale, much smaller than the complete code base to be authored by a developer over the course of an iteration. This can be done for one operation, one field added to a user interface, one stored procedure, etc. As the code base is incrementally built, new tests will be authored and existing tests might be revisited to test additional behavior.

Other points to consider from SmallEcomm's test discipline

- We used a coverage tool to measure and monitor coverage. For a while we were reporting the coverage stat at every iteration post-mortem. However, good line coverage doesn't mean your code is actually well tested and you need to be careful that coverage isn't the end in itself.
- JUnit turned out to be a great part of our process. These tests were run automatically by the build server. Developers would be notified if any of them failed. The customer support projects love them because it helps them detect bugs.
- The UI layers were not well tested because it was difficult to automate UI testing. We focused more on the lower layers of the architecture that were easy to unit test.
- Test Driven Development may not appeal to everyone, but it insures the tests are written. You need to write the tests when you write the code otherwise it will be hard to get them done. Also, you typically find bugs right away when unit testing new code.
- The unit tests saved dozens of round Jira task round trips from QA to Dev and back… With the tests, you know you've broken something before you even check in, so the rest of the team isn't disrupted.

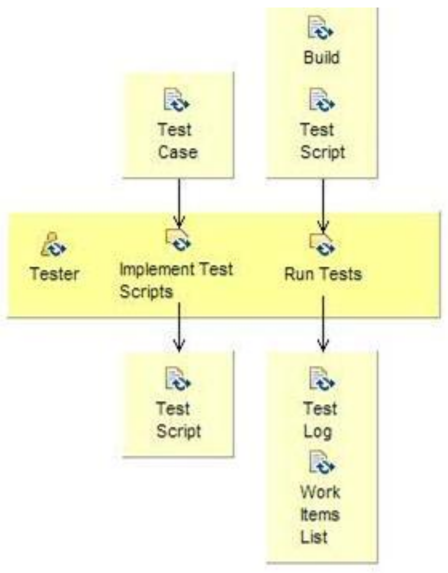
In short, say whatever you want about agile software development, but automated unit testing works.

Test Discipline

The purpose of the OpenUP test discipline is to:

- Provide early and frequent feedback that the system satisfies the requirements.
- Objectively measure progress in small increments.
- Identify issues with the solution.
- Provide assurance that changes to the system do not introduce new defects.
- Improve velocity by facilitating the discovery of issues with requirements, designs, and implementations as early as possible.

- The Test discipline is iterative and incremental. It applies the strategy of "test early and test often" in order to retire risks as early in the system's lifecycle as possible.



We were fortunate that QA was involved early in the project, and while separately managed from the development team, actively participated in the iteration planning and retrospectives. We made every effort to ensure there were no artificial barriers to communications between QA and development. The contributions made by our QA staff during the retrospectives were instrumental to improving our development process.

Task: Implement Test Script

When specifications for a feature were completed, a QA engineer would be assigned to write test cases. When the iteration in which the feature was completed ended, the QA engineer would execute the test cases. During testing, bug reports were filed and the developer implementing the feature would implement the fixes at the start of the next iteration. Bugs were fixed at the beginning of each iteration to prevent them from snowplowing indefinitely into future iterations.

Unfortunately, we found that this QA approach was problematic due to the length of time between feature completion and fixes of all bugs related to the feature. With this process, a feature could be completed at the end of iteration 1 and then passed to QA for testing. QA would then test the feature and create bug reports during iteration 2. At the beginning of iteration 3, the developer would fix the bugs. The bug fixes could then be verified during the remaining time in iteration 3. This scenario shows that it may take up to two iterations (six weeks) to verify that a feature is correct.

This relatively lengthy QA process results in two problems. First, inefficiencies arise because a developer needs to fix bugs in code that they have not worked on recently. Second, the QA process becomes several iterations behind the development process. This is problematic when the project needs to be released externally at an iteration boundary and QA needs to catch up.

To address these problems, we found it necessary to tighten the loop between development and QA. QA needs to have the ability to build and test the latest version of the software as soon as features are completed, regardless of iteration boundaries. As bugs are found, the developer should fix the problems despite the interruption to development of the next feature.

Another way to reduce the test-fix-verify cycle is simply to have developers perform the first round of careful QA on their feature, perhaps even executing test cases prepared by QA. Despite the time this takes, much re-familiarization and communication overhead will be saved if a bug is avoided in this way.

Software Performance Management

One activity we added to the process is software performance management. Good performance does not just happen, and hoping the system will be fast enough is not a course of action. We continuously tested and improve performance as the system evolved. Performance was a concern for us because the usual cost for architectural flexibility is run time performance andany of our risks were related to performance management, for example, we were using the JBoss Rules engine for business rules rather than custom code. We assumed that this would result in a loss of run time performance.

SECO 5.0 is intended to target larger customers, it was important to optimize the system's performance. One developer became the performance engineer during an early iteration and retained this role throughout the project. Here are some of the activities performed by the performance engineer:

- Configure the product in a realistic deployment scenario
- Generate or otherwise obtain large data sets
- Test the product's performance using large data sets
- Locate performance bottlenecks using profiling tools
- Implement code changes as required to improve performance
- Capture the required performance changes as guidelines to be followed by the rest of the team

One way to improve performance management would be to integrate performance tests with the build system. This would provide an early warning system when any feature is added to the system that significantly impacts performance. However, this requires significant additional deployment and testing infrastructure and we have not yet achieved the goal of automated performance testing with each build.

Configuration and Change Management Discipline

Configuration and Change management is about controlling changes to artifacts, ensuring a synchronized evolution of the set of Work Products composing a software system.

Task: Integrate and Create Build

We used automated build tools (Cruise Control) to build the system whenever a change was committed to the revision control system. The build tool would then notify the development team via email if any problems were encountered while building, checking coding standards, or running unit tests. This helped maintain the system in a consistent state that could be retrieved at any time for testing, demos, or further development.

However, we found that it was not always easy to maintain a successful build. Since the build server checked not only for successful compilation but also unit test execution and coding standards, developers found it relatively easy to “break the build.” Furthermore, the failure report generated by the build system was often not able to clearly identify which developer had made a change that broke the build. Since the developer who broke the build was unaware, the build would remain broken for longer than was necessary.

This situation required the introduction of a rotating “Build Police” role. This person’s job was to investigate build failures and ask the offending developer to correct the problem. When we introduced a $1 fine for causing a build failure, the build suddenly became surprisingly stable.

Task: Request Change

Our project mandate did not leave much room for feature changes and at the beginning of the project we naively assumed SECO5 would be the functional equivalent of its ancestor LEGACY. At a functional level this was true, but the use of Ajax and other new web technology meant SECO5 could implement existing features more effectively than LEGACY.

We did not have an explicit change control process. Rather, during the iteration planning process, task cards were prepared and estimated. A task card may recommend a change in the in the way a feature was implemented. This ad hoc process was effective for small team; however, there were several instances of work being undone to take advantage of new found knowledge. It is doubtful that a more explicit change control process would have controlled these changes.

Lessons Learned

In the end, we were able to successfully complete the project on time and on budget although several low-value features were dropped in order to ship the product on time. Throughout the process, we gained some valuable insight into what worked well, and which processes required further iteration. We have commented on these items throughout this case study and they are further summarized here.

Don’t do it alone. – Either engage help or hire someone who has done it before. You simply will otherwise waste too much time second guessing yourself asking the question “is this the way we’re suppose to do it” You are too likely to try to bend your organization to an academically correct perception of the methodology rather than using the methodology to enable your organization.

Break the chains - Do not assume an organization use to working in a linear fashion with chain processes will effortlessly adopt a more collaborative work style. If the culture is not use to working in a concurrent collaborative style, then these traits must be cultivated or the introduction of OpenUP (or any agile process) will not be successful.

Find the leaders - A project always requires someone who is a leader and can engage all team members, but even more so in a light weight or agile project. In SECO5 a dynamic project manager was able to actively create the social climate required for team members to actively engage with each other to share implicit information and feel safe when offering their critiques.

You know best – the creators of OpenUP had a lot of good ideas but are not working in your environment. Do not be afraid to adapt and evolve the process to satisfy your needs. Non of the OpenUP committers will come to your site and slap you[2]. However, make sure when you discard or modify a work product or task you understand the intent behind that work product or task, and that whatever alternative you have effectively addresses the issues.

Adapt and Evolve. Don’t just evolve the software, evolve the process. OpenUP/Basic was created by well meaning practitioners who drew on their experiences to create what they believe is an effective small project software development process. Their experience is not your experience and you must be prepared and willing to adapt the process to your needs and evolve the process as your needs change. Do not panic if you are not evolving the process using the EPF Composer. While there is a risk over a larger organization of different interpretations of the methodology, frequent group interaction should keep the methodology current.

Don’t adapt to the methodology, adapt the methodology to you – remember the methodology is supposed to be an enabler, not a coercive straight jacket.

Do it because it works – don’t adopt practices simply because they are written up in the methodology. Put practices on probation, if they are not working, or people do not believe the weight of the practice is worth the benefit, then either determine why you are not getting the expected benefit, modify the practice, or drop it all together. This is why you do assessments and retrospectives, to improve the process.

Do it together – methodology is not the property of an elite software process engineering group. It belongs to everyone in the process, it must be seen as accessible and adaptable by all members. There must be opportunities to not only let team members express their

[2] With perhaps the exception of Per ☺

concerns and ideas regarding the methodology, but opportunities to actively encourage that expression. Not only were the regular end of iteration retrospectives were a powerful opportunity to discover what didn't work, solicit new ideas, but they were an opportunity to impress upon the staff they owned the process.

Let Software Process Engineers play in their sandbox. - Finally, don't let OpenUP's references to activities, tasks, disciplines, delivery processes and all other forms of software development activities throw you off. These concepts are of interest to those doing software process engineering with the EPF Composer. While it is true what we were doing is technically software process engineering, we did not use the EPF Composer to capture our changes.

In addition to lessons we learned about using OpenUP there were other process lessons that we should share

- Ensure that the documented vision matches the real goals of the project's leaders.
- Maintain a tight loop between feature development, testing, bug fixing, and bug verification.
- Use retrospectives to continuously tweak your development process.
- Use automated tools to enforce your development standards but be careful that they do not become an annoyance.
- Continuous integration building is very helpful but may require active policing to reduce build failures.
- Code reviews are essential but difficult to schedule. Make them a high priority because they will save development time in the long run.
- Design your documentation process so that necessary documents are easy to maintain.
- Performance testing and tuning is a full time activity. Ideally, performance testing should be part of the automated integration build process.
- Build architectural review into your process.
- Monitor risks and the progress of their mitigation strategies.
- Solve the highest risk problems first.

The Final Word

We successfully deployed a tailored version of OpenUP at SmallEcommCo and the use of OpenUP was one factor in the successful creation of SECO5. Adapting OpenUP to work within SmallEcommCo's corporate culture and continuously evolving the process in response to lessons learned contributed to that success. However, the real key to our success was the hard work of the project manager and other key players did to create a culture that could work in a highly collaborative environment. While OpenUP provided the structure to channel and focus the team's energy, it certainly could not create it.

Epilogue

After the launch of SECO5 the project team was rapidly scaled up to develop new features for SE6. With a larger team, it was necessary to make changes to the process so that it would scale to accommodate a larger team.

As the team grew, the morning scrum and iteration planning became unmanageable. This made it necessary to split the team into three smaller sub-teams. These sub-teams were partitioned by logical feature groups to maximize their ability to operate independently. Each sub-team was lead by a team lead and functioned much in the same way as the original SECO5 team.

Each team held morning scrums and individual iteration kickoffs as before. However, in the new structure, team leads held additional scrums three times each week with the QA lead, project manager and product manager. At these scrums, team leads provided aggregate information about their team's status and any blocker issues. Similarly, the key points from iteration kickoffs and retrospectives were aggregated and presented at an iteration kickoff held by team leads and the project and product managers.

With the larger team, it was also necessary to change the way iteration and project planning was conducted. Previously, it was possible to post work items on a wall, grouped by target implementation iteration. In the new structure, the "wall" was moved onto the project wiki. A simple table with columns representing teams and rows representing iterations was used to capture the project plan. In addition to this high-level plan, the iteration plan contained separate tables for each team that provided an overview of the key activities of each team member.

Another result of managing a larger team that included many new people was that we found it necessary to introduce additional processes. These processes included a formal scope change requests that alerted everyone of major changes to the plan. We also introduced written documentation of key steps and work products required by developers when implementing features.

Overall, we found that we were able to scale our adaptation of OpenUP to handle a much larger team with the introduction of an acceptable amount of additional process overhead.

Bibliography

[1] W. S. Adolph, "Cash cow in the tar pit: reengineering a legacy system," *Software, IEEE*, vol. 13, pp. 41-47, 1996.
[2] P. S. Adler and B. Borys, "Two Types of Bureaucracy: Enabling and Coercive," *Administrative Science Quarterly*, vol. 41, pp. 61-89, 03 1996.
[3] N. Harrison, "Liberating Form," 1998.
[4] G. L. Stewart, "A Meta-Analytic Review of Relationships Between Team Design Features and Team Performance," *Journal of Management*, vol. 32, pp. 29-55, February 1, 2006 2006.
[5] S. Adolph, "Are we ready to be unleashed? A comparative analysis between agile software development and war fighting," in *Agile Conference, 2005. Proceedings* Denver, 2005, pp. 20-28.
[6] S. Sawyer and P. J. Guinan, "Software development: processes and performance," *IBM Syst. J.*, vol. 37, pp. 552-569, 1998.

Appendix 1 Abridged Mapping of OpenUP Work Products to SECO5 Work Products

OpenUP	SECO5	Comment
Architecture Note Book	Developers Handbook	Focus on Architectural patterns.
Developer Tests	Junit Tests	
Iteration Plan	Iteration Plan	
Project Plan	Project Plan Presentation	Delivered as a power point presentation
Risk List	Included in Iteration Plan	
Work Item List	“Wall of Wonder”	
Use Case	Use Cases	
Vision	e-mail	
Test Case	?	