

Evolving Beyond Requirements Creep: A Risk-Based Evolutionary Prototyping Model

Ryan A. Carter¹, Annie I. Antón¹, Aldo Dagnino², Laurie Williams¹

¹ College of Engineering, North Carolina State University, Raleigh, NC 27695-7534

² Asea Brown Boveri Inc., Power T&D Company 1021 Main Campus Drive, Raleigh, NC 27606

¹ {aianton, racarter, lawilli3}@eos.ncsu.edu, ² {aldo.dagnino@us.abb.com}

Abstract

Evolutionary prototyping focuses on gathering a correct and consistent set of requirements. The process lends particular strength to building quality software by means of the ongoing clarification of existing requirements and the discovery of previously missing or unknown requirements. Traditionally, the iterative reexamination of a system's requirements has not been the panacea that practitioners sought, due to the predisposition for requirements creep and the difficulty in managing it. This paper proposes the combination of evolutionary prototyping and an aggressive risk-mitigation strategy. Together, these techniques support successful requirements discovery and clarification, and they guard against the negative effects of requirements creep. We embody these techniques in a comprehensive software development model, which we call the EPRAM (Evolutionary Prototyping with Risk Analysis and Mitigation) model. The model was intentionally designed to comply with the Level 2 Key Process Area of the Software Engineering Institute's Capability Maturity Model. Validation is currently underway on several software development efforts that employ the model to support the rapid development of electronic commerce applications.

1. Introduction

Prototyping has gained acceptance in the software engineering community as a credible model for software creation. It has been listed with more traditional process methodologies, such as the Waterfall and Spiral models [Jal00]. Due to the rising costs of software and the lower frequency of systems meeting customer needs, evolutionary prototyping has become a viable software development model [Dav92]. It addresses a number of problems that are not adequately addressed in more traditional software process models: numerous maintenance requests [Gra89], premature freezing of requirements [Som95], difficulty in revisiting previous model stages [Pre01, Som95], lack of consideration of user-system interfaces [Gra91], and miscommunication between developers and customers [Gra91]. Evolutionary prototyping provides other benefits including:

- the clarification of management and user requirements [LSZ93];
- the ability to uncover missing or previously unknown requirements [Gra91, Dav92];
- the flexibility to meet changing constraints for software systems [NC98, FD89];
- the provision of a method whereby users, management, and developers can communicate about systems [Gra89, Gra91, LSZ93];

- the easing of maintenance tasks [Gra89, Gra91];
- the creation of 'better' user interfaces [Gra89, Gra91, LSZ93];
- prototyping with quality [Dav92, Roy90]; and
- the ability for developers to reflect on lessons learned during system development [FD89].

Requirements creep refers to significant additions or modifications to the requirements of a software system throughout the lifecycle, resulting in extensions to and alteration of the software's functionality and scope. Requirements creep can be especially troublesome to developers when it is not properly managed, due to the detrimental impact such changes may have on cost, resources, quality, or the ability to deliver a system that incorporates the new requirements on time. While it can be argued that the majority of software applications have unstable requirements and that some degree of requirements creep is observed in all requirements methodologies [Jon96], evolutionary prototyping is particularly susceptible to significant changes in requirements [Gra89, Gra91].

Electronic commerce (e-commerce) software developers are under pressure to develop applications at a record pace. The widespread lack of process discipline and procedural guidance available for such market-driven environments handicaps e-commerce software developers' ability to handle requirements creep. For this reason, requirements creep is rampant in e-commerce as the innovator often discovers requirements as the system is incrementally implemented. This paper proposes a risk-based evolutionary prototyping model, the EPRAM (Evolutionary Prototyping with Risk Analysis and Mitigation) model, that explicitly addresses the challenges inherent in these small, rapid development efforts.

We ensured that our model follows the intent and guidance (thus adhering to the spirit) of the CMM by first tailoring the Software Engineering Institute's (SEI) Capability Maturity Model (CMM) [PWC94] for use by small software development teams of no more than 12 individuals. We used this tailoring as the basis for the EPRAM. The EPRAM model and its associated techniques facilitate the construction of a complete and consistent set of system requirements, and mitigate the ill effects of requirements creep by supporting the early detection and resolution of requirements conflict.

The EPRAM model is currently undergoing validation in various e-commerce projects in which security and privacy are paramount; early indications demonstrate that the EPRAM model is effective for rapid software development. The model is being employed in five projects for three companies (Newton Instruments, Blue Cross Blue Shield of North Carolina, and Haht Software)

within the North Carolina State University (NCSU) E-commerce Studio [AE00]. Once the model is refined based on the lessons learned during these five development efforts, it will be further validated during the development of Web applications at Asea Brown Boveri (ABB).

The remainder of this paper is organized as follows. Section 2 provides an overview of the relevant related work. Section 3 presents the EPRAM model. In Section 4, we explain our tailoring of the CMM and its impact on the iterative process. Section 5 discusses our preliminary validation efforts as well as our plans for future work.

2. Background and Related Work

An evolutionary prototyping model must address several practical issues, which we discuss below in the context of related work.

2.1 Prototyping in Requirements Engineering

In requirements engineering, prototyping is employed to: generate user interface prototypes in conjunction with scenarios [EKK99]; support walkthroughs with stakeholders to elicit and validate requirements [Sut97, SR98]; reduce ambiguity, incompleteness, and inconsistency during requirements capture [ABR94, RL93]; and incrementally replace specifications with implementations [OS94]. Evolutionary prototyping has been successfully employed in “real world” projects [LSZ93, NC98]. On the other hand, evolutionary prototyping models lack adequate support for requirements evolution and risk management [BS96, Gra98]. Baskerville *et al.* advocate risk analysis for controlling the evolutionary prototyping process [BS96].

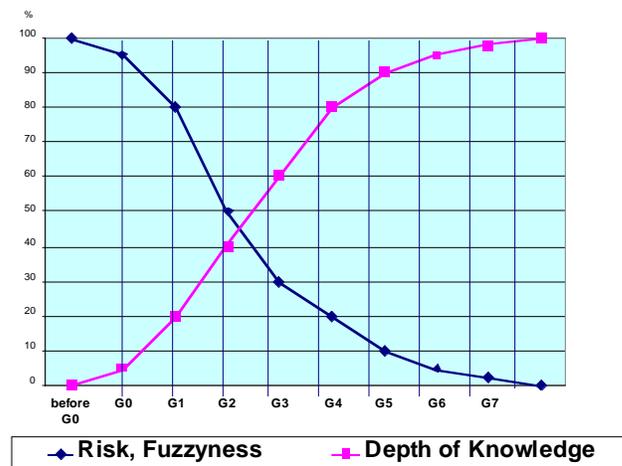
Walker Royce stresses the importance of quality metrics such as flexibility and ease of change throughout the life of a software project [Roy90]. This can be especially meaningful in operational, iterative, and evolutionary prototyping. Davis notes the advantages of operational prototyping, such as built-in quality, as well as the pitfalls, including the inability to quickly implement experimental features [Dav92]. Iterative prototyping is classified as either vertical or horizontal. In vertical prototyping, one or more system components are developed with full functionality in each prototype release; whereas in horizontal prototyping, all system components are developed with limited, yet increasing functionality with each release [Nie93]. We characterize our evolutionary prototyping model as horizontal.

2.2 Managing Risk During Requirements Engineering

Risk management in iterative software development is a key factor for project success [BS96]. However, managing risks during the software lifecycle and, in particular, during requirements engineering activities, is especially challenging in rapid software development. Boehm’s Spiral and Win-Win models have explicit risk resolution sectors that serve to manage risks [Som95, BI96], and Hall addresses identification, analysis, resolution, mitigation, and management of software risks [Ha98]. Risk management is also employed within the context of aligning e-commerce application requirements with corresponding security and privacy policies [AE01]. The EPRAM incorporates the risk and compliance assessment activities of Antón and Earp [AE01] and the risk identification and ranking processes of Baskerville *et al.* [BS96] to form a comprehensive risk mitigation strategy.

Many industries use decision making business models to manage risk during the development phases of a product. For example, ABB uses a phased business decision-making model to ensure that a project meets the business criteria of an organization at all times during the development lifecycle. This business decision-making model is referred to as the Gate Model and has checkpoints that represent Go/No-Go (Stop/Change) decision points. The model ensures that a customer will receive a high-quality and high-value product. Each gate in the model is a decision point at which Senior Management and a Gate Model Committee determine whether to continue or terminate a project based on its benefit, status, risk, resource, and technology considerations. The diagram in Figure 1 represents the objectives of progressing from one gate to another during a product development cycle. When a project begins, the level of risk and fuzziness is very high; as the project proceeds through the gates, the level of risk ideally decreases. At the same time, as the project evolves and moves through the gates, the level of knowledge and maturity in the project team increases. Successive refinements to the EPRAM model will involve adaptation to this gate model to facilitate its adoption by ABB.

Figure 1. Levels of Risk and Knowledge in the Gate Model



2.3 Tailoring the Capability Maturity Model

A growing number of software organizations are turning to the CMM as a viable mechanism to build quality into their software processes [JB97]; however, small organizations are experiencing difficulties in adopting the CMM due to lack of resources, tools, and training [OC99]. The CMM is often customized to support the varying needs of different organizations, and tailoring the CMM to adapt to organizational needs is an accepted practice [GQ95, Jal00, JB97, JB00, KHT00, OC99, Pau98, Pau99]. Some practitioners have tailored it to fit smaller, non-traditional organizations [KHT00, OC99] with mixed results. Such tailorings are undoubtedly required to adequately support the smaller team structure found in common rapid development environments.

Consider Winapp, a company of five employees that develops PC based client server software applications [OC99]. Winapp has attempted to improve its software

process maturity. As the number, size, and complexity of the company's projects grew, the company faced increasing difficulty tracking projects, requirements, and resources. Winapp looked to the CMM for guidance for their process improvements, but found the CMM a bit overwhelming. Winapp initially experienced difficulties in adapting the CMM to meet their specific needs for several reasons, including their inability to support the amount of required documentation, their lack of the resources called for by the CMM, and the fact that their flat team structure is not supported by the CMM. To remedy this, the company extracted the applicable concepts of CMM process improvement and established an improvement framework according to selected key process ideals. The creation of a CMM-inspired framework at Winapp yielded the perception of significant improvements among the developers. Despite the overhead incurred, developers perceived an improvement in the requirements analysis activities of 157% and a decrease in the number of requirements faults by 57% [OC99]. While these figures are admittedly subjective, they indicate the need for process improvement and process guidance within small organizations. The catalyst for our work was our analysis of five teams within Asea Brown Boveri during the Summer of 2000. We observed that many of their development efforts were evolutionary in nature, yet they lacked rigorous support for requirements management. Thus, we tailored the CMM to more appropriately support the demands of such efforts, and we used that tailoring as the basis for the EPRAM model. Before describing our tailoring of the CMM, we introduce the EPRAM model.

3. The EPRAM Model

We now define the model and its subsequent phases. Traditional evolutionary prototyping models include requirements engineering, design, coding, testing, customer evaluation, and enhancement reporting phases [Dav92]. The EPRAM model extends this traditional approach by incorporating risk management as it pertains to e-commerce systems in particular, while imposing adherence to the Level 2 KPAs (Key Process Area) in the CMM.

This section provides an overview of the EPRAM model, shown in Figure 2. In this figure, rectangles represent process activities, dog-eared boxes (pages) indicate documentation artifacts, thick arrows represent major control flows through the process, narrow arrows indicate data flowing to and from the activities, and the block arrow depicts the risk mitigation process maintained throughout the lifecycle. In this paper, we focus primarily on the requirements activities – specifically, those requirements activities that are unique to our evolutionary prototyping model.

During each prototyping cycle, requirements are reevaluated for completeness, consistency, understandability, and compliance with any overriding policies. The requirements considered during each iteration may stem from many sources including, but not limited to: previously established requirements; policies (e.g. security and privacy policies [AE01]); stakeholders (customers, users, and developers); organizational responsibilities; other systems; or additional projects. Requirements are agreed to during a negotiation process stemming from the risk mitigation discussion; the agreed-to requirements are then added to the requirements

documentation and are reflected in each subsequent prototype release.

As is the case in most software processes, design of new prototype releases occurs as the designers decide how the new sets of requirements will be incorporated into the revised prototypes. The architectural design, subsystem and module specification, file structure, global data, and interface design are revised and minimally documented as necessary to ensure a solid system design and prototype structure. The existing design document, if there is one, is also updated to reflect the changes.

At the end of each prototyping cycle, the stakeholders are shown the tested prototype so that they may judge whether or not it meets their expectations. If the customer is satisfied, the project is deemed successful and no additional iterations are necessary; the system is delivered and the project may then wrap up. If, however, there are areas that are not addressed by the prototype, or the prototype must be modified in some way, the new requirements generated during customer evaluation are recorded and a new evolutionary prototyping cycle begins. This iterative approach all but invites requirements creep, which we manage via risk analysis and mitigation.

Risk Mitigation

One of Royce's top ten management principles is the establishment of an iterative life-cycle process that confronts risk early on [Roy98]. Our model extends the traditional evolutionary prototyping process with an aggressive risk mitigation strategy that helps control the process and deter requirements creep. In the EPRAM model, each requirement is assessed for risks and their potential impacts [AE01]. The adoption of new technologies, such as auctions [PFI99], cause changes in the requirements which may introduce a conflict with respect to the associated policy. Heuristics, available in [Dem00], are applied to prevent such conflicts from being overlooked during the analysis process. We are also adapting our model so that it will be compliant with the risk management structure of the Gate Model at ABB.

The EPRAM's risk mitigation procedure borrows from the strategy given in [BS96] to neutralize the negative consequences of requirements modification. A risk list of identified trouble areas is maintained and reexamined at the beginning of each iteration. New risks are added by the analysts as risks are surfaced by system developers or stakeholders, as well as due to changes in the application domain, problem domain, computer systems, and/or the development environment. Each risk is evaluated to determine its consequences, and the team assigns priorities to those risks so they may be emphasized and mitigated accordingly. Both the severity and the probability of each risk are ranked on a scale from low (1) to high (5). The severity and probability values are multiplied together to form the risk score. The higher that this score is, the more risk is associated with the given issue. Finally, the development team determines resolution strategies for those risks of most concern (those with the higher risk scores). The team first addresses the higher-risk issues, developing and documenting mitigation strategies for those specific issues. Whereas analysis of each risk may be considered a subjective process, analysis of each new requirement is more systematic since each requirement must undergo a policy compliance and requirements

conflict check before its incorporation into the subsequent prototype release.

To demonstrate how the risk analysis and mitigation process of the EPRAM model works, we provide a brief example in which there is a risk that the customer will fail to remain actively involved with a project. In such a case, the developers add this failure to the risk list during the risk analysis phase of the evolutionary prototyping cycle. Within the risk meeting, the team determines that the risk is

very likely to occur (thus receiving a probability score of 5) and will be moderately severe if it does occur (thus receiving a severity score of 3). These values are multiplied together such that the risk receives a risk score of 15. If this score is one of the higher risk scores, the team determines a mitigation strategy for this risk – possibly including requests for an on-site customer representative or weekly meetings with customer representatives.

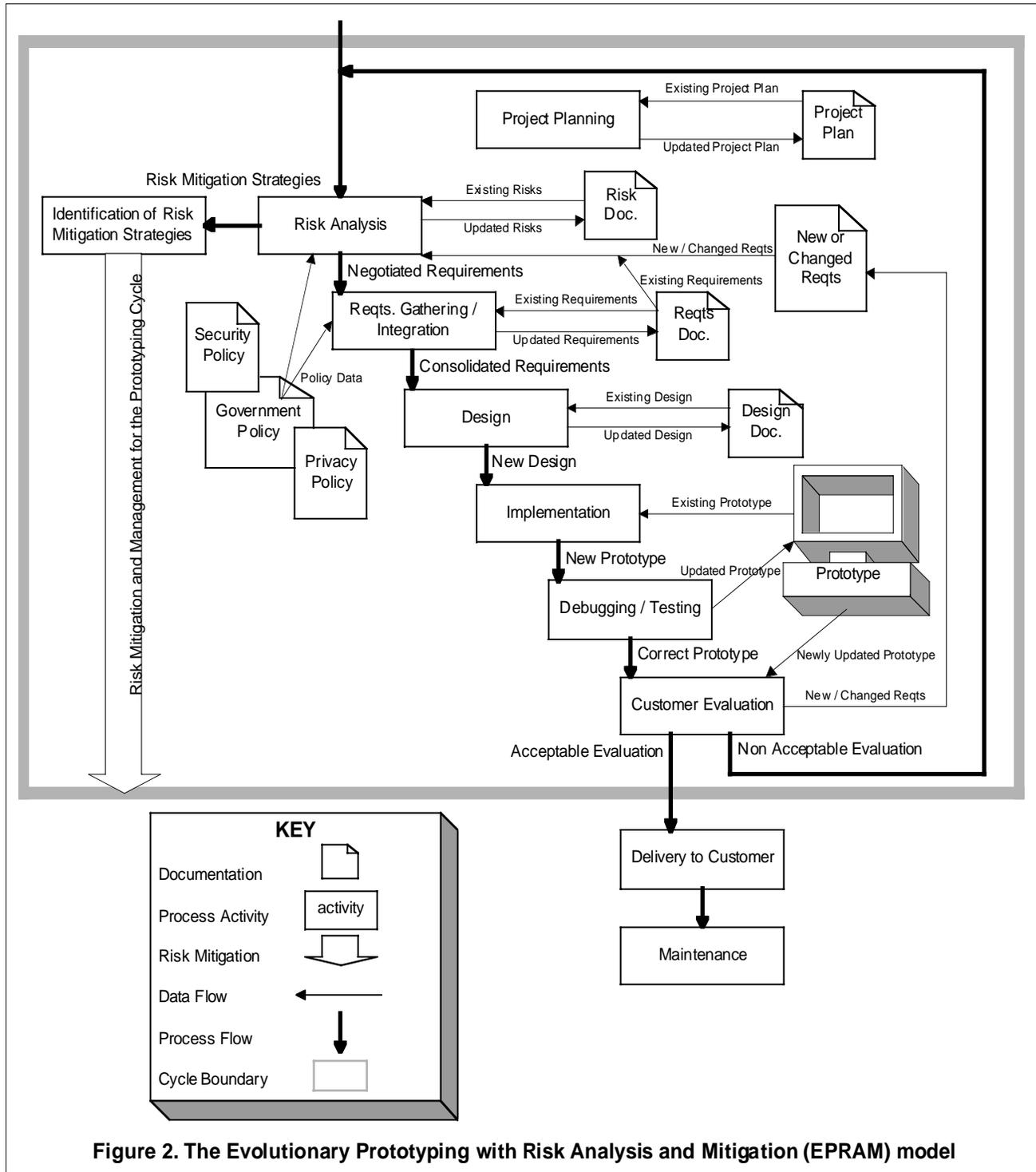


Figure 2. The Evolutionary Prototyping with Risk Analysis and Mitigation (EPRAM) model

The EPRAM model also employs risk analysis to check proposed requirements for compliance with established policies. Consider the stakeholder requirement for users' names and addresses to be automatically transferred to a third party that will fulfill users' order. Such new requirements are discussed during the risk analysis meeting and compared to existing policies. In the case of this requirement, it is discovered that the organization's privacy policy states that any user's personally identifiable information will not be shared with those outside the organization; therefore, the requirement and the privacy policy are in conflict. Since the likelihood that the conflict will occur is certain (thus receiving a probability score of 5) and the credibility of the organization as a guardian of privacy is at stake (thereby receiving a severity score of 5), a suitable solution, such as modifying the requirement or the policy, must be developed. The alternatives are negotiated with the stakeholders and an adequate resolution is achieved.

Requirements Evolution

Stakeholders are allowed to change the system requirements in response to new releases of the prototype, and the prototype is iteratively modified in response to these new requirements. This may result in an increase to the system scope. The question thus arises: *How will the EPRAM model benefit the development organization by guarding against requirements creep?* The EPRAM model's risk mitigation support helps practitioners identify requirements changes that create significant risk within a project. It allows them to decide what requirement additions are valid from a policy or development standpoint; negotiate cost, scope, quality, and schedule with the customer; and modify the prototype accordingly. The EPRAM model encourages development teams to negotiate risk mitigation strategies with the stakeholders to alleviate the effects of requirements changes. Finally, during requirements evolution we identify and resolve inconsistencies across the requirements and policies by providing specific heuristics [Car01, Dem00]. The ultimate goal is to ensure consistency between the requirements, all policies, and the system's functionality [AEP01].

4. Customizing the CMM for Small E-Commerce Teams

As previously mentioned, the EPRAM model is intended for use by teams of no more than 12 individuals in organizations yearning for a more disciplined and mature approach to software development. To this end, the SEI's CMM is an important standard by which the "maturity" of a software process can be judged. The CMM includes a set of rigorous practices that impose large amounts of overhead that may be especially detrimental for small resource-challenged organizations. It is widely applicable given a willingness to tailor it and the SEI advocates such tailoring [PCC93]. While we were initially hesitant about using the CMM due to its overhead and rigor, we found it possible to incorporate the practices that support our process model while tailoring the remaining practices so that they made sense in the context of the environments in which we would employ the EPRAM model.

The CMM contains five levels of increasing maturity. It reinforces a series of best practices through its use of Key Process Areas (KPA) for each level of maturity. Each KPA

is subdivided into goals that must be met before that KPA is satisfied. Additionally, all KPAs must be satisfied before a process obtains the recognition of having achieved a given maturity level. For purposes of this paper, we focus on the Repeatable Level (Level 2) which provides a suitable basis for process quality without introducing overhead unnecessary for small teams. We examined five of the six KPAs in the Repeatable Level of the CMM: Requirements Management, Software Project Planning, Software Project Tracking and Oversight, Software Quality Assurance, and Software Configuration Management [PCC93], but omitted Subcontract Management, as it is not often relevant to the entrepreneurial rapid development environments targeted by our efforts.

As previously mentioned, the CMM does not directly support rapid, iterative, poorly-resourced software development efforts. For this reason, we were initially concerned that our EPRAM model may not meet the CMM's strict practices for process maturity, and yet, we realized the need to incorporate the ideals of the CMM. We opted for "some" improvement, recognizing that the introduction of any process improvements would, at a minimum, yield far better quality in the organization as well as the resulting process. The EPRAM model, introduced in Section 3, accommodates small teams, working on small e-commerce applications. The CMM Level 2 KPAs are incorporated as much as possible so our model upholds the spirit of the CMM and benefits from its process improvement guidance [JB00].

Our CMM tailoring began by determining the roles and responsibilities for a team's members. We established nine roles for each team; one person may have multiple roles or multiple people may fulfill a single role. Each role is assigned to one of three groups (the Software Engineering Group, the Software Development Group, or the Software Quality Assurance Group) that collaborate in specific areas or for a particular purpose during a project. Once the roles and responsibilities were defined, we extracted and discussed each CMM Level 2 goal, commitment, ability, activity, measurement, and verification. In many cases, the tailoring simply required a bit of "wordsmithing" and/or clarification to more adequately reflect the needs of the smaller organizations for which the model is intended; however, others required more significant changes. There are 115 elements (Goals, Commitments, Abilities, Activities, Measurement, and Verification) in the five KPAs that we considered for Level 2. Of these, 43 were modified, 9 were rejected as not applicable, and 1 was marked as optional. Consider the examples provided in Table 1 which show a small sample of the efforts expended to tailor the CMM. Commitment 1 concerns the documentation of the procedures used to manage requirements as contained in an organization's project policy. We restructured the original CMM commitment to include project roles and policy sections. In the second example (Ability 4 in Table 1), we broadened the definition of 'training' for requirements management activities which is more narrowly defined under the original context of the CMM. We maintained the spirit of the CMM throughout our tailoring efforts as it reflected new team roles, established a hierarchy of team leadership, accounted for limited resources, satisfied the demands for rapid development, and removed overhead that was not applicable to the new process model for our target

Table 1. Example tailored CMM elements

CMM Element	Rewrite / Rerword / Clarify	Rationale
Commitment 1: The project follows a written organizational policy for managing the system requirements allocated to software.	The project follows a written project policy for managing the system requirements allocated to software. This policy typically specifies that allocated requirements are documented; the allocated requirements are reviewed by the requirements engineer under the project manager and individuals responsible for the tasks listed; and the software plans, work products, and activities are changed to be consistent with changes to the allocated requirements.	<ul style="list-style-type: none"> • Changed to reflect roles established for the project. • The project policy is the organizational policy. • Changed to denote sections within the policy.
Ability 4: Members of the software engineering group and other software-related groups are trained to perform their requirements management activities.	Members of the project team and other software-related groups are trained to perform their requirements management activities. Training can take the form of on-demand lecture, self-study, peer study, out-sourced training, or other applicable methods.	The methods of training must be flexible since smaller companies can not afford extensive training (time or cost).

audience. The final tailored CMM served as the basis for the EPRAM model.

5. Discussion and Future Work

Fowler argues that the requirements engineering community often loses sight of the fact that requirements should be modifiable [Fow00]. He asserts that software should be just that – soft, meaning that it should be as malleable as possible. Perhaps this is even more appropriate for e-commerce development environments. Requirements that are appropriate today may not be appropriate throughout the duration of the entire project due to, for example, increased pressure to “go live” before one’s competitors.

Evolutionary prototyping, strengthened by risk analysis and mitigation, ensures the continual revisiting of a system’s requirements through each prototyping cycle. Such a process provides the stability to minimize the ill-effects of significant requirements creep while providing the freedom to continually improve the requirements for a software system. In light of this flexibility, we contend that the EPRAM model is a lighter-weight method; it is people-oriented and readily adapts to changing requirements.

Traditional software development methodologies and the CMM impose a disciplined process, making an organization’s software process and practices more predictable and efficient. However, these methodologies are considered cumbersome to follow, imposing a high degree of formalism, and slowing the development process. These methodologies are often referred to as heavy methodologies [Fow00]. In response to heavy methodologies, a new generation of software development methodologies, referred to as lightweight or agile, has arisen. These methodologies are characterized by their adaptability, speed, collaboration, and efficiency. Light methods thrive on change, deliver fast results, and must be adaptive as well as people-oriented, while still meeting formal documentation and project management criteria. Lightweight methodologies are targeted at software teams for which external competition creates extreme pressure on the delivery process and changes in project requirements are the norm.

The requirements engineering and software engineering communities have begun to examine and implement lightweight methodologies to judge their impact on the creation of quality software. Such an interest can be seen in methodologies such as extreme programming, Crystal, open source, Scrum, and others [Fow00]. We are currently

validating the “lightweightness” of the EPRAM model by closely monitoring its use and evaluating the “denseness” of the artifacts produced throughout the lifecycle.

According to Fowler, lightweight methods are adaptive rather than predictive and people-oriented rather than process-oriented [Fow00]; the EPRAM model fulfills these criteria. The model recognizes that stakeholders often do change the requirements and provides methods to manage changing requirements. Evolutionary prototypes, by their very nature, evolve to meet the expectations of the stakeholders. The EPRAM model also encourages a people-centric view of software creation as the model, itself, builds on the skills and strengths of the individuals filling the roles for the projects. Though roles are defined, they can be combined and planned for so that interests and assets of the individual practitioners are the underlying sources of the team’s strength. Moreover, the evolutionary prototyping process involves stakeholders at regular intervals throughout the process, which in turn ensures that stakeholder viewpoints and requirements are considered throughout the development effort. Lightweight methodologies call for less-formalized documentation; future work includes an analysis of the amount of documentation required by our model to determine the appropriate degree of informality, documentation, and sufficiency. Currently, we require requirements documents to focus solely on enumerating the functional and nonfunctional requirements as well as any policy-based requirements. Based on the data we are currently collecting, we may decide to further “lighten” what is required in all documentation artifacts produced while employing the EPRAM model.

The corporate climate is receptive to lightweight methodologies due to their relevance in the rapid development of continuously changing software applications that must meet customers’ expectations. This is evidenced by ABB’s interest in the application of these methodologies and their investment in the development of the EPRAM model. Additional work is currently underway to justify the applicability of the risk mitigation component of our lighter-weight methodology as related to previously-established corporate risk analysis strategies such as ABB’s Gate Model.

The EPRAM model is not limited to those seeking lighter-weight processes; it also aids practitioners seeking to manage requirements creep. According to Jones, prototypes by themselves can reduce requirements creep by 10 to 25% [Jon96]. Requirements inspections also

minimize the rate of requirements creep because they enable practitioners to systematically and methodically identify errors and inconsistencies. Requirements inspections reduce requirements creep by nearly 30% [Jon96]. The EPRAM model introduces requirements inspection via a series of risk assessment meetings held to discuss newly identified requirements. Requirements are then assessed for compliance with the governmental, security, and privacy policies, as well as the existing requirements [AE01].

Stakeholder involvement is also central to other lightweight methodologies. Extreme programming, for example, offers developers and customers the opportunity to compromise on four variables (cost, time, scope, quality) [Bec00]. Customers are allowed to choose the desired values for three of these four variables, and the developers determine the value of the last variable. For example, a customer might state that they want "a high quality release" on May 1 for \$x, and the developers can tell them which of the customer-prioritized requirements might make it into that release. Or, a customer might state that they want a high quality release with specified features for \$y, and the developers will determine when they can deliver the release. In this same vein, the EPRAM model allows the development team to determine when the added or modified requirements, discovered during the customer evaluation stage, will significantly change the scope. The team analyzes and judges the impacts of the proposed changes. Risk mitigation enables team members to dissuade stakeholders from significant changes at the beginning of each cycle; thereby encouraging the team to evaluate the possible options and enter into negotiations with the customer to reevaluate scope, quality, cost, and schedule for the project. Significant changes often provide high project risks and therefore, must be mitigated.

Even though we were very familiar with and had a deep understanding of evolutionary prototyping and risk mitigation strategies, it was important to tailor the CMM before actually developing the EPRAM model to prevent us from discarding CMM activities just to fit our model. By tailoring the KPAs early on, we ensured that the tailored version included the strongest guidelines rather than the easiest-to-implement guidelines. Additionally, we ensured that the CMM was tailored in such a way as to support resource-challenged, small organizations.

During the development of the tailored Repeatable Level, it became apparent that the CMM was resilient in its ability to adapt to the changes that we were proposing and the new development environments of e-commerce applications. The maturity activities provide a credible basis for the EPRAM model, and one that improves its relevance to those organizations seeking to align themselves with the SEI's CMM. We are confident that the CMM and the EPRAM model complement each other and together build a solid approach to software development.

In closing, the EPRAM model is based upon three solid elements: a firm CMM basis for maturity, an adaptive evolutionary prototyping structure to accommodate flexibility, and a comprehensive risk analysis and mitigation strategy to ensure software quality and process reliability. This triumvirate offers the foundation, techniques, and methodologies needed by small resource-challenged entrepreneurial software development teams seeking process maturity for their organizations. Our

immediate plans involve further validation of the model, including a study of the model in relation to other established lightweight methodologies and the appropriateness of the model's artifacts in terms of level of formality, consistency, completeness, and density.

Acknowledgements

This work was partially funded by Asea Brown Boveri Grant #530493. The authors wish to thank Hema Srikanth, Ashish Sureka, Kai Yang, and Lingyun Yang for their assistance and discussions while tailoring the CMM model upon which the EPRAM model is based.

References

- [ABR94] R.D. Acosta, C.L. Burns, W.E. Rzepka & J.L. Sidoran. A Case Study of Applying Rapid Prototyping Techniques in the Requirements Engineering Environment. *First IEEE Int'l. Conf. on Requirements Engineering*, pp. 66-73, 1994.
- [AE00] A.I. Antón & J.B. Earp. A Multidisciplinary Electronic Commerce Project Studio for Secure Systems, *4th National Colloquium for Information Systems Security Education (NCISSE)*, Washington, D.C., unnumbered pages, 23-25 May 2000.
- [AE01] A.I. Antón & J.B. Earp. Strategies for Developing Policies and Requirements for Secure Electronic Commerce Systems, To appear in *Recent Advances in Secure and Private E-Commerce*, Kluwer Academic Publishers, 2001.
- [AEP01] A.I. Antón, J.B. Earp, C. Potts & T.A. Aslpaugh. The Role of Stakeholder Privacy Values in Requirements Engineering, submitted to *IEEE Int'l Symposium on Requirements Engineering (RE'01)*, 2001.
- [BI96] B. Boehm & H. In. Identifying Quality-Requirement Conflicts, *IEEE Software*, 13(2), pp. 25-35, March 1996.
- [BS96] R.L. Baskerville & J. Stage. Controlling Prototype Development Through Risk Analysis, *MIS Quarterly*, 20(4), pp. 481-504, December 1996.
- [Bec00] K. Beck. *Extreme Programming Explained*, Addison-Wesley, 2000.
- [Car01] R.A. Carter. North Carolina State University M.S. Thesis, April 2001.
- [Dav92] A.M. Davis. Operational Prototyping: A New Development Approach, *IEEE Software*, 9(5), pp. 70-78, September 1992.
- [Dem00] J.H. Dempster. *Inconsistency Identification and Resolution in Goal-Driven Requirements Analysis*, M.S. Thesis, NC State University, Raleigh, NC, May 2000.
- [EKK99] M. Elkoutbi, I. Khriess & R.K. Keller. Generating User Interface Prototypes from Scenarios, *IEEE Int'l. Symp. on Requirements Eng.*, pp. 150-158, 1999.
- [FD89] D.A. Fern & S.E. Donaldson. Tri-Cycle: A Prototype Methodology for Advanced Software Development, *22nd Annual Hawaii Int'l. Conf. on System Sciences*, Vol.II: Software Track, pp. 377-386, 1989.
- [Fow00] M. Fowler. Put Your Process on a Diet, *Software Development*, 8(12), pp. 32-36, December 2000.
- [GQ95] M. Ginsberg & L. Quinn. *Process Tailoring and the Software Capability Maturity Model*, SEI Technical Report, CMU/SEI-94-TR-024.
- [Gra89] I. Graham. Structured Prototyping for Requirements

- Specification of Expert Systems, *IEEE Colloquium on Expert Systems Lifecycle*, pp. 5/1-5/3, 1989.
- [Gra91] I. Graham. Structured Prototyping for Requirements Specification in Expert Systems and Conventional IT Projects, *Computing & Control Engineering Journal*, 2(2), pp. 82-89, March 1991.
- [Hal98] E. Hall. *Managing Risk*, Addison-Wesley, 1998.
- [Jal00] P. Jalote. *CMM in Practice – Processes for Executing Software Projects at Infosys*, Addison-Wesley, 2000.
- [JB97] D.L. Johnson & J.G. Brodman. Tailoring the CMM for Small Businesses, Small Organizations, and Small Projects, *Software Process Newsletter*, No.8, IEEE Computer Society, Winter 1997.
- [JB00] D.L. Johnson & J.G. Brodman. Applying CMM Project Planning Practices to Diverse Environments, *IEEE Software*, 17(4), pp. 40-47, July/August 2000.
- [Jon96] C. Jones. Strategies for Managing Requirements Creep, *IEEE Computer*, 29(6), pp. 92-94, June 1996.
- [KHT00] K. Kautz, H.W. Hansen & K. Thaysen. Applying and Adjusting a Software Process Improvement Model in Practice: The Use of the IDEAL Model in a Small Software Enterprise, *22nd Int'l. Conf. on Software Engineering*, pp. 626-633, 2000.
- [LSZ93] H. Lichter, M. Schneider-Hufschmidt & H. Zullighoven. Prototyping in Industrial Software Projects – Bridging the Gap Between Theory and Practice, *15th Int'l. Conf. on Software Eng.*, pp. 221-229, May 1993.
- [NC98] N. Nunes & J.F. Cunha. Case Study: SITINA - A Software Engineering Project Using Evolutionary Prototyping, *Proc. CAiSE'98/IFIP 8.1 EMMSAD'98 Workshop*, 1998.
- [Nie93] J. Nielsen. *Usability Engineering*, Academic Press London, 1993.
- [OC99] S. Otoy & N. Cerpa. An Experience: A Small Software Company Attempting to Improve its Process. *Proc. Software Technology and Engineering Practice, STEP '99*, pp. 153-160, 1999.
- [OS94] M.B. Ozcan & J.I.A. Siddiqi. Validating and Evolving Software Requirements in a Systematic Framework, *First IEEE Int'l. Conf. on Requirements Engineering*, pp. 202-205, 1994.
- [Pau98] M.C. Paulk. Using the Software CMM in Small Organizations, *Joint 1998 Proc. Pacific Northwest Software Quality Conf. and the Eighth Int'l. Conf. on Software Quality*, pp. 350-361, October 1998.
- [Pau99] M.C. Paulk. Using the Software CMM With Good Judgment, *ASQ Software Quality Professional*, 1(3), pp. 19-29, June 1999.
- [PCC93] M.C. Paulk, B. Curtis & M.B. Chrisis. *Capability Maturity Model for Software. Version 1.1*, Software Engineering Institute Technical Report, CMU/SEI-93-TR, February 24, 1993.
- [PFI99] Policy Framework for Interpreting Risk in eCommerce Security. CERIAS Technical Report, <http://www.cerias.purdue.edu/techreports/public/PFIRES.pdf>, Purdue University, 1999.
- [PWC94] M.C. Paulk, C.V. Weber, B. Curtis, M.B. Chrissis et al. *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison Wesley Longman, Inc, 1994.
- [Pre01] R.S. Pressman. *Software Engineering – A Practitioner's Approach*, McGraw-Hill, 2001.
- [RL93] B. Ramesh & Luqi. Process Knowledge Based Rapid Prototyping for Requirements Engineering. *1st IEEE Int'l Symp. on Requirements Eng.*, pp. 248-255, 1993.
- [Roy90] W. Royce. Pragmatic Quality Metrics for Evolutionary Software Development Models, *Proc. Conf. on TRI-ADA '90*, pp. 551-565, 1990.
- [Roy98] W. Royce. *Software Project Management : A Unified Framework*, Addison Wesley, 1998.
- [Som95] I. Sommerville. *Software Engineering*, Addison-Wesley, 1995.
- [SR98] A.G. Sutcliffe & M. Ryan. Experience with SCRAM, a Scenario Requirements Analysis Method, *3rd Int'l. Conf. on Requirements Eng.*, pp. 164-171, 1998.
- [Sut97] A. Sutcliffe, A Technique Combination Approach to Requirements Engineering, *Third IEEE Int'l. Symposium on Requirements Engineering*, pp. 65-74, 1997.