Evolutionary Improvements of Cross-cutting Concerns: Performance in Practice

Stephany Bellomo, Neil Ernst, Robert L. Nord, and Ipek Ozkaya
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA, USA
{sbellomo, nernst, rn, ozkaya}@sei.cmu.edu

***Abstract*—As industry continues to embrace incremental software development, many projects run into the challenge of incrementally evolving cross-cutting concerns such as performance. To better understand how projects are handling this challenge in practice, we captured experiences from two financial services that made a series of performance improvements over several months. We discovered some commonality in how these projects refine the work, enabling incremental requirements analysis and allocation of work. In this paper, we describe two key aspects of this evolution: refining the concern by breaking it into its constituent parts to drive design tasks and allocating the parts to iterations as the software evolves. Two practices we observed that support this evolution include ratcheting broadened to conceptually describe the refinement approach in dimensions of response to stimuli in a given context and analysis conducted concurrently and loosely coupled from implementation work. This refinement supports ongoing exploration of the problem and solution, and evolutionary development, such as course changes, when new information is acquired.**

Keywords—performance; quality attribute; requirements; cross-cutting concerns; refinement; allocation; evolution; maintenance; incremental iterative development

I. Introduction

A worst-case scenario for any project under frequent delivery pressure is to discover an impactful requirement late, after it has become a significant impediment to the user, with no ability to break the work into manageable chunks to quickly resolve the problem. Cross-cutting concerns such as performance, security, and availability are particularly hard to break apart into smaller increments since, by their nature, they impact many aspects of the system. Why is it that some projects sustain their established cadence when faced with this situation and others do not?

Determining satisfaction criteria, development effort, and value is a fundamental activity in managing the scope of functional requirements for iterations during evolutionary development. The requirements analysis process involves refining and separating abstract stakeholder concerns into constituent parts and understanding their interrelationships so they can be allocated to iterations in the software development process.

What makes this a particularly hard problem is that cross-cutting concerns (such as quality attribute or non-functional requirements) and the work associated with them are not as independent as features. Dependencies between other software elements must be considered in packaging the pieces into units that must be treated together or sequenced over iterations.

We describe examples from two financial services projects making performance improvements as the systems evolved. The paper is structured around our exploration of these two questions: (1) How do these projects parse important tangible constituent parts of the cross-cutting performance concern? (2) How do the parts get sequenced as the software evolves?

II. Exploratory Study

In this paper, we explore two examples of system evolution captured through semi-structured interviews with the technical leads of the projects. We began each interview by collecting background information to establish project context and then asked the interviewees to describe examples in which their teams evolved a quality attribute requirement. After the first round of interviews, we analyzed the results and selected a commonly shared concern, performance, for deeper exploration. We collected further details via additional interviews, emails, and phone calls.

A. Project A

Project A develops financial support software for a mid-size firm. The software supports the buying and selling of financial securities. Performance is a key concern, particularly at the close of the financial day. Customers may have to pay interest if they have to borrow a large sum of money to hold sell orders overnight if they were not processed by the close of trading. Project A described its performance evolution as a set of state transitions. The software was used for weeks or months at a time between these states while customers provided feedback to the developers that informed the work in the subsequent state. State transitions included analysis, design, and implementation work described as user stories below.

A-S1: Baseline financial order. This is the baseline state in which orders are manually submitted and processed.
A-S2: Autopilot feature. Customers voiced a concern that transactions processed near the close of the financial day are not acknowledged fast enough by the system. The team

1063-6773/14 $31.00 © 2014 IEEE
DOI 10.1109/ICSME.2014.91

determined that automating user interaction will improve order throughput. The autopilot solution was implemented.
A-S3: Data caching. Customers reported latency issues with individual order processing. The team added data caching to reduce latency of database read and write operations.
A-S4: Data write delay. Customer feedback showed that latency was still an issue. Rather than requesting to reduce latency, the business articulates a measurable response that order processing must be completed in less than one second. The team's analysis revealed that one cause of latency was the time spent updating the database with post-transaction results. The team implemented an enhancement to delay writing data to the database until after the orders were processed.
A-S5: Prioritize transactions. Analysis of production logs revealed that transactions for smaller amounts were acknowledged before transactions for larger amounts. The team added a feature to prioritize messages by highest dollar value so that those have a higher probability of getting processed.

1) Refining Requirements into Constituent Parts: Each state transition was a performance improvement to the basic order-processing capability. Improvement was measured by the criteria in Table I. The performance requirement was refined enough to determine satisfaction and value. The associated work was refined enough to determine effort. The effort for each state varied depending on the time needed to do the work to show something of value to the customer. The states were broken down into smaller internal iterations for allocation (not shown in Table I).

In the quality attribute requirement (QAR) parsing column in Table I, we capture the requirements at each state along three dimensions summarized from the quality attribute scenario: *Stimulus*, *Context*, and *Response* [3]. The stimulus dimension describes the action that initiates the system response, the context dimension describes the evolving environmental condition (e.g., increasing number of users), and the response dimension describes how the system responds to the stimulus and may include a response measure. We analyzed how Project A evolved its performance requirements using these dimensions to understand how it parsed work into constituent parts. Examples are described below.

- *Stimulus:* A-S4 to A-S5 illustrates refinement in the stimulus dimension, moving from a single- to multi-user perspective. A second example is A-S2 to A-S3, which illustrates moving from batch processing to individual transaction processing.
- *Context:* The initial requirement focused development on the system behavior (simple baseline case) to test ideas before dealing with complexities and uncertainties of the environment. A-S1 to A-S2 illustrates refinement in the context dimension, moving from manual processing to the autopilot solution, by adjusting the boundary of the system with respect to its environment, including the user's role. Moving from A-S4 to A-S5 accounts for the complexities of the rotary algorithm in the environment.

TABLE I. PROJECT A PERFORMANCE IMPROVEMENT EVOLUTION

	QAR Parsing	Value	Effort
A-S1	**Stimulus:** Customer initiates process (multi-user) **Context:** Users processing transactions with system; deadline approaching **Response:** Process volume of transactions	Baseline order feature	1x
A-S2	**Stimulus:** Customer initiates automated process **Context:** System processing transactions (single-user) **Response:** Process volume of transactions; new time less than current time	Enhanced "Autopilot" feature with performance focus to reduce batch processing latency	3x
A-S3	**Stimulus:** Order process initiates transaction **Context:** System processing transaction; deadline approaching (single-user) **Response**: Process individual transaction; new time less than current time	Improved individual order capability with reduced latency	1x
A-S4	**Stimulus:** Order process initiates transaction **Context:** System processing transaction; deadline approaching (single-user) **Response:** Process individual transaction; processing time less than or equal to 1 s	Improved individual order capability with further reduced latency	2x
A-S5	**Stimulus:** Customer submits orders **Context:** System processing transactions**;** rotary algorithm; deadline (multi-user) approaching **Response:** Process and prioritize transactions	Enhanced batch-level prioritization feature with performance focus on reducing batch-latency	1x

- *Response:* A-S3 to A-S4 illustrates refinement in the response dimension by ratcheting the response measure [2]. The response measure value was refined to less than 1 second for processing individual transactions.

These states are not a linear progression toward a predefined goal. The evolution of the quality attribute requirement is driven by feedback given by the development team and the business owners/users (e.g., concern that design won't scale, users get impatient with single-order latency). The team analyzes this feedback and, in some cases, the work is prioritized, divided into smaller pieces, and parsed into reasonable chunks through trade-offs. With three dimensions to adjust (stimulus, context, response), ratcheting in one dimension may require easing up on another to make progress.

2) Allocating Performance Work to Increments: Project A explained that separating performance-related from feature-related requirements in the backlog or when planning and allocating work to iterations was not useful. Project A followed evolutionary incremental development by doing performance-related analysis work concurrently and loosely coupled from implementation sprint work (Fig. 1). This is a practice that we observe in industry [1]. This ongoing analysis helps break up the problem so that the implementation sprint

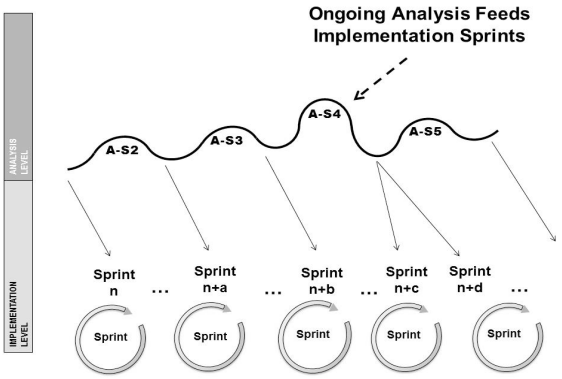

Fig. 1. Evolutionary incremental development.

work can proceed at a more uniform cadence. As analysis was completed, work was allocated to sprints (shown with arrows in Fig. 1). Well-understood changes refining features, such as A-S2 and A-S3, were allocated to implementation sprints with minimal analysis. However, in cases where significant analysis was needed (e.g., A-S4), the team created a prototype to explore the problem and investigate alternative solutions while continuing to mature the system and implement ongoing requirements. For A-S4 the changes were more substantial, so the work was allocated to multiple sprints.

B. Project B

Coincidentally, Project B, from a different company than Project A, was also a financial system with very stringent performance requirements. The system functionality included high-speed stock order processing. Like Project A, the performance iterations are described as state transitions. This project was in the pre-release phase, focusing on evaluating the design to ensure it could meet requirements. The team used a scenario-driven approach to investigate performance risks. Output of scenario analysis is input to the evolving system design and architecture of the next state, as described below.

B-S1: Create order. Baseline state.
B-S2: Order-processing algorithm improvement. During scenario-based design review, a concern surfaced over the uncertainty of whether the algorithms were fast enough in all situations, so peak load was identified to process 10,000 orders in 0.1 millisecond (ms). In response, the Java algorithms were improved for processing orders.
B-S3: Queue latency improvement. Queues were incorporated into the design to improve performance in response to latency concerns. The question addressed in this state was "How much overhead does the queuing approach generate?"
B-S4: Queue limitation exploration. An exploratory requirement of processing stock orders for larger organizations drove further investigation of the single-server, multi-core design. The analysis increased understanding of feature limitations under stress to improve the design or plan for mitigation options. In this case, the team determined the design change would be too impactful, so they planned for the mitigation option to buy another server and distribute the load.
B-S5: Garbage collection investigation. After discovering a problem during testing, the team investigated how the system runs when garbage collection is turned off.

1) Refining Requirements into Constituent Parts: Table II summarizes the evolution states of Project B.

The analysis of this example also shows changes to the stimulus, context, and response, as summarized below.

- *Stimulus:* BS-3 shows variations in the stimulus with the artifact changing to focus on a specific part of the system, the queue.
- *Context:* B-S2 shows variations in the context by increasing the concurrently processed orders to 1,000 orders while maintaining response time at 0.1 ms.
- *Response:* The response goal remained consistent at 0.1 ms throughout all the state transitions. This was a target that they reached over several rounds of tweaking the design, analyzing the response under increasingly stringent conditions, such as described in B-S2, and making incremental improvements.

2) Allocating Performance Work to Increments: Project B did not separate feature-related and performance-related requirements in their software development lifecycle either. Like Project A, they conducted exploratory performance analysis and design work concurrently with maturing other features and continuing implementation. Work was integrated into implementation sprints as it became better defined

TABLE II. PROJECT B PERFORMANCE IMPROVEMENT EVOLUTION

	QAR Parsing	Value	Effort
B-S1	**Stimulus:** Order request **Context:** System operating under normal conditions (under 10,000 orders) **Response:** Process order less than 0.1 ms	Baseline order feature	1x
B-S2	**Stimulus:** Order received by system **Context:** System (specifically Java algorithms) operating at peak load (over 10,000 orders) **Response:** Process order under 0.1 ms	Improved order-processing feature with reduced latency	0.1x
B-S3	**Stimulus:** Order submitted **Context:** Artifact focus is queue **Response:** Queue overhead should allow for same response measure as 0.1 ms	Improved confidence that order-processing feature will hold up under strenuous circumstances	2x
B-S4	**Stimulus:** Order **Context:** More orders than core/queue architecture allows **Response:** Process order under 0.1 ms	Increased understanding of feature limitations under stress (to improve or plan for mitigation options)	0.5x
B-S5	**Stimulus:** Order submission **Context:** Garbage collection turned off **Response:** System runs during trading hours maintaining response of 0.1 ms	Improved order feature performance (with garbage collection off)	0.2x

through analysis of design artifacts, prototyping, or both. We see the challenge of managing dependencies during allocation in this example. In B-S4, a concern was identified in which the stock for an initial public offering could potentially have too many orders against it, which could push the design beyond its limits to scale and maintain throughput and latency. The team explored solutions, and refactoring involved changes to multiple interdependent system components. Ultimately, the team determined they could not afford to disrupt cadence and stop delivery of features. They decided not to fix the problem. The mitigation plan was to purchase a second server if this problem emerges and redesign at that time.

III. DISCUSSION

Ultimately the purpose of refining cross-cutting concerns is to decompose a stakeholder need or business goal into iteration-sized pieces. Allocation then takes those pieces and determines when to work on them. This is both an analysis and a design activity: refinement and allocation are explorations of the problem and solution spaces, and evolutionary, iterative development allow for course changes when new information is acquired. Developers work toward satisfying cross-cutting concerns in the context of the effort and ultimate value.

We observed in these projects that developers refined performance requirements using a feedback-driven approach. The analysis approach used by the teams allowed them to parse the evolving performance requirement to meet increasing user expectations over time (expressed as state transitions). Within each state transition, developers refine cross-cutting concerns into requirements by breaking them into their constituent parts in terms of the scope of the system and *response* to *stimuli* in a given *context*. The system and cross-cutting performance requirements evolve as stimuli, context, and response are ratcheted. The concerns that drove performance improvements became regression tests in subsequent states to ensure that changes did not increase latency.

We see these projects using exploratory analysis techniques to elicit, refine, and evolve emerging requirements in an integrated manner. In addition to obtaining user feedback, they investigated anticipated questions and concerns using analysis and design spikes. They allocated work to iterations by considering dependencies and conducting analysis concurrent with implementation to keep up the cadence. Project A mentioned that they had redesigned early in the project to promote modularity. This raises the question whether modular architecture may have been an enabler for allocating work to complement the exploratory analysis techniques in the software development and evolution process.

The projects were successful in breaking the challenging cross-cutting concerns into smaller pieces. However, the reality of such requirements is that they cannot always be refined into chunks of equal size as required by software development processes such as Scrum that enforce fixed iteration lengths. The examples demonstrate varying efforts that span multiple iterations. Exploratory analysis techniques help to some extent, but challenges remain to further smooth the process.

IV. CONCLUSION

Incorporating cross-cutting concerns such as quality attributes throughout the software development and sustainment lifecycle is not always a straightforward process. One challenge is a refinement problem: to correctly specify the quality attribute requirement to a level of detail that is measurable and valuable and then find design fragments that can be completed within the release cadence. Another challenge is an allocation problem: to correctly allocate design fragments to iterations to optimize the relationship between cost and value. This relationship is a complex one. In some situations, needless over-preparation defers implementation that may lead to cost of delay. In other situations, expedient implementation choices made to meet the constraints may make the system less adaptable and lead to costly rework.

We see evidence of projects that are better able to sustain their cadence with a combination of refinement and allocation techniques guided by measures for requirement satisfaction, value, and development effort. As we retrospectively analyzed these examples, we found that these teams did not follow a formal technique; however, they did have common elements in how they refined the work into smaller chunks, enabling incremental requirements analysis and allocation of work into implementation increments.

Fowler describes ratcheting performance thresholds in terms of tightening the threshold over time to improve the value of a response measure [2]. Based on what we have learned by examining these examples, we suggest that this ratcheting concept can be broadened to conceptually describe the refinement approach in other dimensions. For example, changes in the evolving context, such as increasing the number of orders to be processed or reducing the performance threshold for a single order, allow for breaking a cross-cutting concern to a reasonably sized chunk of work for analysis, allocation, or testing. We suggest that these examples, which demonstrate ratcheting in multiple dimensions, could be useful for teams struggling with how to break up and evolve cross-cutting concerns during iterative and incremental development.

ACKNOWLEDGMENT

We acknowledge Felix Bachmann, Luis Carballo, and Salient Federal Solutions for their technical contribution to this paper.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. This material has been approved for public release and unlimited distribution. DM-0001452.

REFERENCES

[1] S. Bellomo, R. Nord, and I. Ozkaya, "Elaboration on an integrated architecture and requirement practice: prototyping with quality attribute focus," International Confereence on Sofware Engineering, Twin Peaks Workshop, San Fransisco, CA, May 2013.

[2] M. Fowler, "An appropriate use of metrics," website posting http://martinfowler.com/articles/useOfMetrics.html#MetricsAsARatchet

[3] I. Ozkaya, L. Bass, R. L. Nord, and R. S. Sangwan, "Making practical use of quality attribute information," IEEE Software, vol. 25, no. 2, pp. 25–33, March/April 2008.