Integrate End to End Early and Often

Felix H. Bachmann, Software Engineering Institute

Luis Carballo, Bursatec

James McHale and Robert L. Nord, Software Engineering Institute

This column is all about stories, and this one is as exciting as a paperback whodunit. The details are all included, and I hate to spoil it, but there's a happy ending. The story is about something old—designing and implementing a new system when the old one was really old (two decades!) and something new—using outside research consultants to save the day with a secret sauce. Enjoy! *–Linda Rising, associate editor*

ON 3 SEPTEMBER 2012, with single-order latency under 100 microseconds and sustained throughput of 100,000 transactions per second, *la Bolsa Mexicana de Valores* (BMV; the Mexican Stock Exchange) stock-trading engine joined the world's largest high-performance exchanges—Nasdaq OMX, NYSE, Euronext, Deutsche Börse, and the London Stock Exchange.[1]

The software development project to build the new trading engine was completed in house by Bursatec, BMV's technology arm. The Bursatec development team faced a significant challenge in designing and implementing the new trading system: the last system they developed was more than 20 years old, implemented in COBOL, and running on a mainframe. In addition to transitioning to modern software engineering practices and technologies, Bursatec wanted to combine stock market trading with derivatives trading on the same platform. Achieving these goals would reduce operating costs and provide a single, high-throughput, low-latency interface to external financial markets.

To meet these demands, the lead architect (author Luis Carballo) brought in experts from the Software Engineering Institute (SEI) to help the Bursatec team select and adapt the appropriate methods, processes, and techniques to ensure the development of a very fast and highly reliable system, a must in modern financial markets. The Team Software Process (TSP) helped developers avoid mistakes or fix them early, rather than during multiple test phases. In addition, architecture-centric engineering (ACE) guided the design and implementation of a system architecture that not only supports what BMV needs today but also enables the system to evolve to support envisioned future features.

Interplay of Architecture and Process

ACE methods focus on what to build; TSP methods focus on how to build it. TSP provides process discipline for management and measurement across the project life cycle and for building high-performance teams.[2] ACE provides the technical discipline for designing and implementing a system that meets the organization's business objectives.[3]

Blending architecture and process discipline provided Bursatec with a strategy to integrate early and to address technical risk in the form of uncertainty, complexity, and the cost of developing and maintaining quality systems.[4,5] In this article, we call out three interesting integration points: at the project's start to establish criteria; nine months later, when the first hard evidence is available; and at the project's end as marked by system, acceptance, and user testing. Because technical transparency is a benefit of

TABLE 1 Project plan for BMV stock-trading engine.

Cycle	Duration	Activities
1	14 weeks	Developed two architecture versions, evaluated communication packages, and built initial testing framework
2	10 weeks	Developed third architecture version for review and built early core framework
3	18 weeks	Developed, integrated, and tested basic round-trip trading functionality/performance
4	2 weeks	Performed technical reviews with external Java and communications experts
5	25 weeks	Developed, integrated, and tested full trading-day functionality/performance
6	21 weeks	Developed, integrated, and tested maintenance and changes/extensions to functionality

the combined approach, this strategy meant that both the development team and their managers knew what was going on—and why—and could act early to investigate areas of uncertainty and prevent small problems from turning into the large ones that kill projects.

After the project kickoff in August 2009, one of the first steps was to conduct a Quality Attribute Workshop (QAW) that refined, extended, and validated early formulations of the system's quality attributes with stakeholders into the quality attribute scenarios that are standard, measurable formulations of the architecturally significant requirements. Not surprisingly, given the importance of speed for the new system, the stakeholders identified runtime performance as one of the most important quality attribute scenarios. For the developers, one benefit of defining quality attributes is that the practice placed significant emphasis on ensuring that the attributes be measurable. For example, the performance attribute was measured in two ways: the time for individual transactions (how fast each one was processed) and the throughput (how many transactions per second were processed on an ongoing basis). These quality attribute scenarios also acted as a contract between the developers and the stakeholders because they captured objective criteria of what a good system would be from the stakeholders' perspective.

The week after the QAW, the initial TSP launch took place, bringing together a project team of 14 members to produce the necessary planning artifacts. The team divided the project into six cycles, each with specific deliverables that either BMV's users or management would see, and subdivided each cycle into iterations that built on each other to complete the cycles (see Table 1). During the launch, the team split into two groups: the core architecture team and the developer team. They built detailed plans for both the architects and developers that covered the first two iterations of cycle 1. The goal was to plan the architecture activities in the context of supporting the team within their existing time and budget constraints, as the architecture was the first deliverable in the project's cycles.

To guide system design, the architects used the architecture-driven design (ADD) method, which is based on transforming quality attribute scenarios into an appropriate design. It's also an iterative approach; the architects might need to rethink a decision that they made earlier when incorporating additional quality attributes or reviewing feedback from analysis of further decompositions. The performance quality attribute scenario coupled with the high-availability requirements led the architects to realize that conventional approaches—such as a three-tier architecture separating responsibilities into presentation, business services, and data—didn't provide the best solution for their new system. Consequently, the architects spent the next two weeks exploring alternative solutions and their potential negative outcomes. While the architects spent approximately three months on the initial architecture, the developers worked on prototypes (mostly for nonproduction or throw-away code), evaluated high-speed communications packages, and built different pieces of frameworks that they could integrate into the system as the architecture evolved.

The architects subdivided the ADD process into several two-week periods. At the end of the first two-week period, they presented their findings with evidence (rationale, design decisions, tradeoffs, and measures) that their chosen approach was correct to SEI software architecture experts, who challenged each scenario using rigorous scenario-based peer review techniques. Every two weeks thereafter, the software architects presented solutions with appropriate evidence for the scenarios they had created and any changes and additions made to earlier scenarios. For example, with respect to the performance requirement, the architects demonstrated how a stock order would traverse the system, estimating and measuring the timing required for every step. With each review, SEI coaches identified risks associated with a particular approach. Among the risks they identified with respect to performance was that synchronizing with backup systems would affect the timing. In all, there were three iterations of the architecture, each lasting six weeks.

At a replanning event six weeks into the project, a TSP coach helped the architects adjust the plan for the next six weeks. The architects based adjustments to the plan both on the architecture's current state—crude at that point, but with a good idea of what the next set of challenges was—and on the data that they had gathered so far from their own work. The developers made similar adjustments that included working on a critical capability for automating transaction testing and building evaluation prototypes for competitive commercial packages for high-speed, redundant, persistent messaging, a key component of the new system.

At the start of the second iteration, the SEI architecture coaches brought in the developers to begin working on prototypes, specifically focusing on risks (such as the timing of querying complex data structures and internal queuing) that couldn't be addressed solely via software architecture. This important step allowed developers to deepen their understanding of the architecture and familiarize themselves with the problems, which was a lengthy process. The developers had six weeks to implement the prototypes; at the beginning of the third iteration, the developers returned and presented their results to the architects. This process enabled the architects to finalize their architecture design using the results from the prototypes.

One important activity during architecture design was to create a prototype to evaluate the quality attributes of the messaging components and how the product fit in the architecture. As a result of the findings, the architects decided to create a specific layer that encapsulated that functionality and incorporated the high-availability features of the system while reducing the estimated size of code to be written. The architects also measured the queuing mechanisms used internally, which informed their decision on the best implementation to use.

During the early prototype production and performance measurement, Java's automatic garbage collection to reclaim memory became an issue, so the team created development guidelines to minimize the impact of the overhead garbage collection adds that can affect performance in order processing latency.

After two cycles and three major iterations over five calendar months, the team participated in a two-day Architecture Tradeoff Analysis Method (ATAM) review to evaluate the architecture. The review provided a final stakeholder validation of the quality attributes as well as independent verification that the proposed architecture should work to implement those attributes.

Once the architecture was stable enough, as evaluated by the ATAM, the SEI architecture coaches conducted an active design review in which the architects communicated the entire architecture to the developers in a structured way. Next, the team conducted conformance reviews for which the developers needed to provide evidence to the architects that the systems they were building conformed to the architecture. These reviews reinforced the rationale that the whole system would meet stakeholder needs. Subsequently, the architects and developers regrouped into a single, integrated team, removing the potential issues that sometimes arise when software architects throw their artifacts "over the wall" to developers. The architects dealt with issues and revised the architecture as necessary while shouldering a normal development workload. The team identified role managers to focus on issues surrounding performance and garbage collection, two implementation issues critical to the trading system's success.

Figure 1 summarizes the iterations that occurred during the first three cycles. The team established two main feedback loops—architecture design and implementation—that constantly informed each other, for which the architecture and the development plan acted as the coordination mechanism. The architecture design loop enabled the architects to react to new and changing requirements from the stakeholders, while the implementation loop ensured that the developers would implement those requirements correctly. The team measured not just its designs and code but also its own working processes. Together, these loops ensured the development of a system that behaved as envisioned while making progress visible to management.

Soon after, the integrated team implemented enough "round-trip" functionality that they could make meaningful performance measurements for basic trades. Based on early integration feedback, architectural and nonarchitectural tweaking pushed performance

The review validated the quality attributes as well as independent verification that the proposed architecture should work.

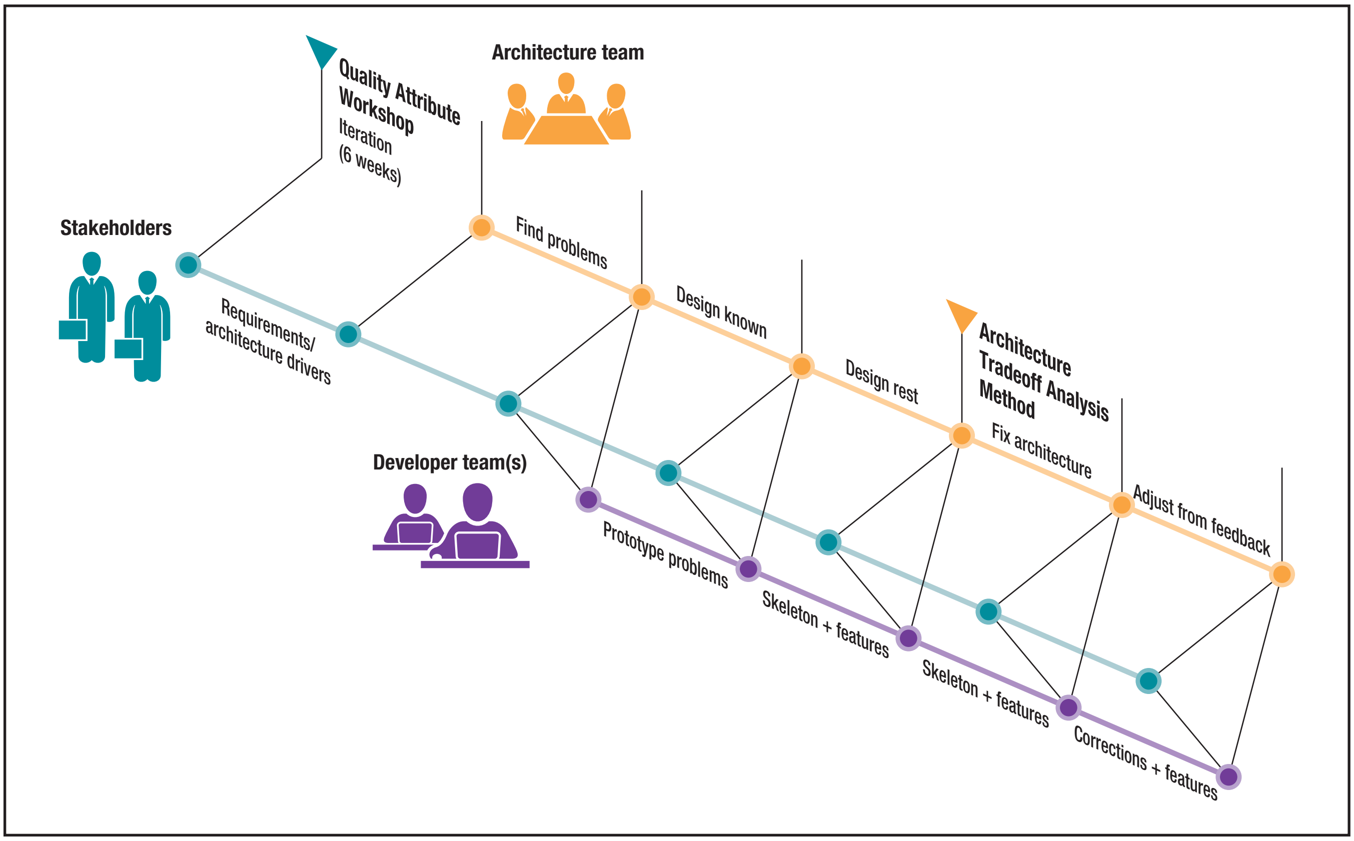

FIGURE 1. Architecture-centric engineering with TSP. At recurring intervals: stakeholders communicate requirements; the architecture team assigns tasks to developers and delivers status reports to the stakeholders; and the developer team delivers the latest version of software to architects and release version updates to the stakeholders.

to levels exceeding the initial goals of 1 millisecond and 10,000 transactions per second. The team knew that they would likely need that cushion because more complicated order types would take longer as new and changed requirements continued to arrive throughout the project. Still, almost a year before the scheduled end of development, the team knew that it would achieve the project's most challenging performance goal.

Accomplishments

The new trading system's development progressed on schedule and within budget. Moreover, early tests confirmed that the trading system's performance far exceeded initial expectations. The combination of TSP and ACE brought discipline, measurement, and a set of robust architectural techniques.

Through six major development cycles including 14 or so iterations over 21 months, the overall team developed over 287 KLOC, spending about 12 percent of their effort on architecture and approximately 14.5 percent in unit testing and integration testing. In addition, performance testing occurred more frequently during unit testing early in development and in integration testing later.

In contrast, the SEI would normally expect almost twice as much testing effort, with potentially much more in system testing, to push the overall total close to or beyond the 50 percent mark—an unfortunately realistic expectation in our industry. Validation testing showed a very low defect count, less than 50 defects in more than 200 KLOC (less than 0.25 defects per KLOC, well below the 1,000 to 2,000 that's more typical in our experience); fixing the defects hasn't modified the architecture. The testing framework allowed for a smooth, continuous integration. Due to the early investment in architecture and a detailed, data-driven approach to managing both schedule and quality, the system required less testing throughout development. Bursatec put the system into production on 3 September 2012, and as of this writing, it has worked without major defects, including support of the largest initial public offering in Mexican market history.

Another benefit to using TSP and ACE is that the team of developers were prepared for inevitable changes in architecture requirements—indeed, for changes of any sort—over the 21 months of development. When the team received new requirements, it could evaluate them quickly for technical impact and implementation cost in terms of time and effort, using the architecture that accurately reflected the current implementation.

With the quality attributes formally captured, the architecture in place, and detailed development plans at every step, a project with enormous risk potential in both technical and business terms ran on time, within budget, and generally free of the drama that large development efforts often exhibit. One of the quality attributes that can drift easily is performance. Measuring performance in terms of latency and throughput on a constant basis and including those measurements in the continuous integration process enabled the team to tightly control any changes that could affect that attribute. It also allowed the team to consider external factors such as the correct configuration of the Java Virtual Machine and the server's BIOS parameters. At the end of the project, latency and throughput greatly exceed initial expectations.

Investment in early architecture and team practices drives a development effort's life cycle and plays a role in managing risk. Constant integration allows early detection and correction of any inconsistency or problems. End-to-end integration is preferred, and integrating all the components (or as many as possible, depending on which stage the project is in) will facilitate a smooth integration test while also revealing any system bottlenecks or requirement and coupling changes that are needed.

Investment in architecture doesn't mean a big design up front. Delaying implementation while waiting for the architecture to be complete—and therefore the requirements to be complete—wasn't necessary. Some of the development started at very early stages of the design process. Early implementation allowed the team to measure some of the design decisions they made at early stages, allowing them to confirm that those decisions were correct and providing feedback to help them modify the design as early as possible. It also gave the stakeholders a feel for the system at early stages and kept everybody informed. Setting up the architecture design as an iterative process enables the development team to start with the architecture design before all the requirements are clear and ensures that they can react to new and changing requirements quickly.

This strategy helps development teams ensure as early as possible that the end-to-end integration works. As in any inspection process, the outcome will

ABOUT THE AUTHORS

FELIX H. BACHMANN is a senior member of the technical staff in the Software Solutions Division at the Software Engineering Institute. Contact him at fb@sei.cmu.edu.

LUIS CARBALLO is the software engineering director at Bursatec. Contact him at lcarballo@bursatec.com.mx.

JAMES MCHALE is a senior member of the technical staff in the Software Solutions Division at the Software Engineering Institute. Contact him at jdm@sei.cmu.edu.

ROBERT L. NORD is a senior member of the technical staff in the Software Solutions Division at the Software Engineering Institute. Contact him at rn@sei.cmu.edu.

probably include feedback and recommendations to change or correct something. Plan some time to adjust the system based on review input. Identifying and resolving integration defects earlier reduces the cost of rework. An iterative and incremental approach fosters collaboration and facilitates handoffs, reducing the cost of delay. It also allows better communication among teams, team members, and stakeholders.

Acknowledgments

This material is based on 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. ATAM and Architecture Tradeoff Analysis Method are registered in the US Patent and Trademark Office by Carnegie Mellon University. TSP and Team Software Process are service marks of Carnegie Mellon University.

References

1. A. Puaar, "BMW Targets HFT with Trading System Revamp," *TRADE News*, 20 Sept. 2012; http://thetradenews.com/newsarticle.aspx?id=9696.
2. W.S. Humphrey and J.W. Over, *Leadership, Teamwork, and Trust*, Pearson Education, 2011.
3. L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed., Addison-Wesley 2013.
4. R. Nord, J. McHale, and F. Bachmann, *Combining Architecture-centric Engineering with the Team Software Process*, tech. report CMU/SEI-2010-TR-031, Software Eng. Inst., Carnegie Mellon Univ., 2010; www.sei.cmu.edu/library/abstracts/reports/10tr031.cfm.
5. W. Royce, "Measuring Agility and Architectural Integrity," *Int'l J. Software and Informatics*, vol. 5, no. 3, 2011, pp. 415–433.