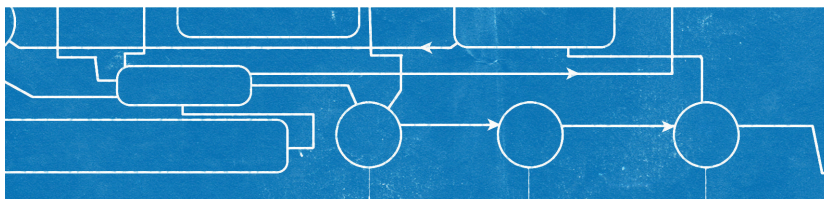


Toward Agile Architecture

Insights from 15 Years of ATAM Data

Stephany Bellomo, Ian Gorton, and Rick Kazman,
Software Engineering Institute

// To gain insight into which quality attributes are of greatest concern to agile practitioners, researchers analyzed stakeholder concerns from 15 years of Architecture Trade-Off Analysis Method data across 31 projects. Dominant concerns include modifiability, performance, availability, interoperability, and deployability. //



CONCERNS ABOUT LONG-TERM deterioration in large-scale agile projects' quality (called technical debt¹) have renewed the agile community's interest in software architecture.^{2–4} This interest indicates a growing consensus that a strong architectural foundation lets teams rapidly evolve complex software systems using iterative, incremental development. This position is consistent with arguments in *IEEE Software's* Mar./Apr. 2010 special issue on Agility and Architecture.

One major challenge agile teams face in building an architectural

foundation is balancing two competing concerns:

- delivering near-term functional requirements (based on the agile philosophy of delivering user value early and often) and
- meeting near- and long-term quality attribute goals (without which the project can grind to a halt because system complexity makes efficient modifications impossible).

Quality attribute prioritization in particular can be difficult in early increments, and a wrong decision can

result in hard-to-modify, unreliable, slow, or insecure systems.⁵

So, we sought to obtain empirical evidence on the most common quality attribute concerns that projects must address and their relative importance. We performed a meta-analysis on the results of several Software Engineering Institute analyses carried out over the past 15 years using the Architecture Trade-Off Analysis Method (ATAM). The ATAM leverages quality attribute scenarios that project stakeholders have identified and prioritized and uses this information to evaluate the architecture and identify architectural risk. (For more on the ATAM, see the related sidebar.) Here, we examine our analysis results and describe how agile teams can use them to prioritize and allocate quality attributes.

ATAM Study Results

We present results from two studies of ATAM data collected from 31 projects. In Study 1, Ipek Ozkaya and her colleagues analyzed ATAM scenario data from 1999 to 2006.⁶ In Study 2, we analyzed ATAM scenario data from 2006 to 2013. The quality attribute scenario data examined in both studies was generated during ATAM phase 1 and collected from the elicited utility trees.⁷ (For more on the two ATAM phases and utility trees, see the ATAM sidebar.)

The sidebar "Our Analysis Methodology" describes our method. SEI's ATAM analyses cover a range of project types across government and industry domains. Table 1 categorizes the data into IT versus embedded systems, and government versus nongovernment projects. Ozkaya and her colleagues' paper detailed Study 1's project profile.⁶ Study 2 included nine projects and

348 quality attribute scenarios; these projects were more heavily weighted toward IT (78 versus 42 percent).

Figure 1 compares the two studies' results, which don't include categories with fewer than three scenarios. (We didn't include them because their scenarios tended to be more context specific or less generalizable.) The top quality attribute concerns were consistent. For example, modifiability was the most common quality attribute in both studies. Performance was the second-most-important attribute in Study 1 and third (by a small margin) in Study 2. This suggests that although the technical environment for software-intensive systems has evolved over the years, the most important stakeholder and architectural concerns haven't changed much.

Table 2 lists the top 20 quality attribute concerns of Study 2, ranked from highest to lowest according to their frequency in the dataset. This view reinforces the importance of modifiability, performance, interoperability, availability, and (to a lesser extent) usability. However, it also suggests that project teams prioritize specific quality attribute concerns in a finer-grained way than what Figure 1 shows. For example, consider the top three quality attribute concerns: "modifiability: reducing coupling," "performance: latency," and "interoperability: upgrading and integrating with other system components."

Data Analysis and Discussion

We now look at this data's implications.

Modifiability Leads

As we mentioned before, modifiability ranked highest in both studies (24 percent in all scenarios for Study 1, and 26 percent for Study 2). As Table



THE ARCHITECTURE TRADE-OFF ANALYSIS METHOD

The Architecture Trade-Off Analysis Method (ATAM) lets developers analyze software and system architectures with respect to quality goals.¹ The ATAM has a long pedigree—government and commercial organizations have been using it for more than 10 years—and substantial supporting documentation, including books, papers, and training courses. The ATAM normally involves two phases. Phase 1 elicits information about the architecture from the architecture team; phase 2 elicits project stakeholder needs.

The ATAM's component techniques have been developed and refined over the years. One such technique is *quality-attribute-scenario analysis*, which captures architecturally focused requirements from stakeholders. Such analysis follows a structured format defined in six parts: an explicit stimulus, a stimulus source, an explicit response, a response measure, the environment, and the artifact the scenario affects.

Another technique is a *utility tree*, which is a hierarchically organized affinity grouping of quality attribute concerns and scenarios derived from a particular project.¹

Finally, *quality attribute tactics* are architectural design primitives that ATAM evaluators look for in the architecture to quickly assess the architects' strategies.¹

The structured nature of these techniques helps ATAM teams perform elicitation and analysis of quality attribute information in a consistent manner.

Reference

1. P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*, Addison-Wesley, 2001.



OUR ANALYSIS METHODOLOGY

We conducted our analysis in four stages using three independent coders—analysts who assign tags (categories) to scenarios.

In stage 1, we collected Architecture Trade-Off Analysis Method reports for 2006 through 2013 and consolidated them into one dataset.

In stage 2, we conducted independent coding of a sample of the data (100 records) using a shared set of tags to assess and address variability across analysts. This stage culminated in a session reconciling coding discrepancies.

In stage 3, we conducted independent coding of all remaining records in the dataset and carried out final reconciliation.

In stage 4, we performed the categorization, summation, and analysis of the full dataset, producing the results we summarize in the main article.

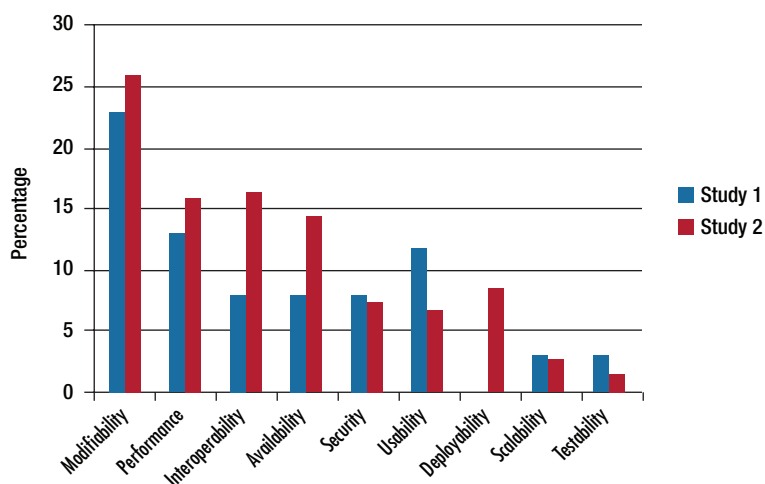


FIGURE 1. The distribution of quality attributes covered in the two studies. The top quality attribute concerns were consistent across both studies.

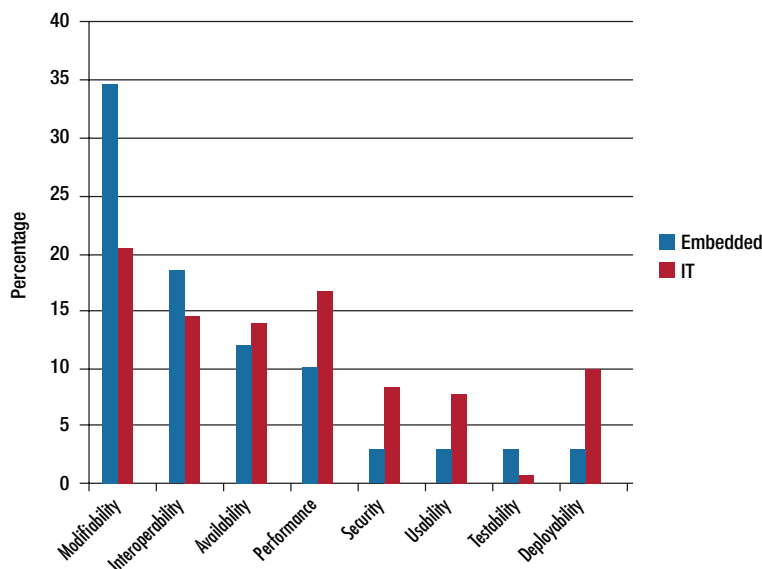


FIGURE 2. The distribution of quality attribute concerns for IT and embedded-systems projects from Study 2. Modifiability concerns were much more prominent in embedded-systems projects.

2 shows, five of the top 20 quality attribute concerns were modifiability related. So, we performed a detailed analysis of these concerns.

Table 3 summarizes and ranks the modifiability concerns. The top two concerns in Table 3 comprise 47 percent and focus on anticipating software changes. A variety of system types are represented in the ATAM data; anticipated changes included adding or replacing third-party components, adding new sensor types, replacing algorithms, and changing UI content. However, stakeholder concerns related to change weren't limited to the software application alone. A significant number of the scenarios focused on changes external to the software, such as OS replacement, network modifications, new technology integration, and hardware platform upgrades. A list such as this, derived from real project data, can help teams reason about future changes they should consider and avoid being caught off guard by an impactful, unanticipated change.

Performance, Interoperability, and Availability

Performance, interoperability, and availability concerns also occurred frequently. Because of space limitations and because the interoperability concerns found in the dataset were often similar to the modifiability concerns, we focus here on performance and availability.

Tables 4 and 5 summarize and rank the performance and availability concerns. A common theme emerges. Most scenarios focused on maintaining critical capability and consistent response times in the face of partial or major system failures (80 percent of the performance scenarios and 64 percent of

TABLE 1

Architecture Trade-Off Analysis Method project profiles.

Study	Time frame	Project type (%)			
		IT	Embedded	Government	Nongovernment
1	1999–2006	58	42	88	12
2	2006–2013	78	22	66	34

TABLE 2

The top 20 quality attribute concerns from Study 2.

Rank (by no. of scenarios)	Quality attribute	Concern
1	Modifiability	Reducing coupling
2	Performance	Latency
3	Interoperability	Upgrading and integrating with other system components
4	Modifiability	Designing for portability
5	Usability	Ease of operation
6	Availability	Fault detection
7	Interoperability	Ease of interfacing with other systems
8	Modifiability	Designing for extensibility
9	Availability	Fault recovery
10	Performance	Resource management
11	Deployability	Minimizing build, test, and release duration
12	Modifiability	Reusability
13	Availability	Preventing faults
14	Scalability	Increased processing demands
15	Security	Authorization
16	Interoperability	Resource and data sharing
17	Security	Resisting attacks
18	Deployability	Configuration or dependency management
19	Modifiability	Configurability and composability
20	Deployability	Backward compatibility or a rollback strategy

the availability scenarios). Failure scenarios included loss of hardware, software failures, network outages or failures, dramatic load increases, and synchronization failures. We suggest that besides modifiability, teams should consider these types of failure scenarios when reasoning about work they might need to plan for.

IT versus Embedded Systems

Figure 2 shows the differences in quality attribute concerns between IT and embedded-systems projects from Study 2. Modifiability accounted for almost 35 percent of the embedded-system concerns—15

percent more than any other quality attribute category. In contrast, although modifiability was the most important concern for IT projects, there wasn't as much of a gap between it and the other quality attributes. Most IT project quality attribute counts, including modifiability, ranged from 14 to 20 percent.

These numbers could reflect either the system's underlying quality attribute requirements or the design's current perceived problems. If the former is the case, this finding suggests that for embedded projects, developers must consider requirements that promote portability and

encapsulation to reduce the impact that changes have on external subsystems (such as hardware or networks) over time. For IT projects, the more even spread across quality attributes suggests a need to balance requirements considerations more broadly. Although no one-size-fits-all list of quality attribute concerns exists, we propose that these project-specific views can be useful when developers are considering how to prioritize in an IT-versus-embedded context.

Deployability: An Emerging Concern

Deployability represented 10 percent of the quality attribute concerns for

TABLE 3

A summary of the modifiability concerns.

Concern	Percentage of scenarios
Adding, replacing, or modifying a functionality or component (for example, a third-party component, new sensor, adaptor, or algorithm)	32
Configurable software (order throughput, UI content, display, feature toggle, reporting, or simulation)	15
Operating system or registry changes	11
Replacing or changing the networked environment (for example, a new protocol or network platform)	10
Changing, adding, or replacing a model or tool capability (run-time and non-run-time)	8
Software component composition or reuse (from the core asset base)	7
Porting or integrating a new technology, device, or hardware platform (a new vehicle, a device, sensors, a dual core, and so on)	6
Changing message content and formats (translation might be required)	4
Adding or replacing services (such as messaging)	3
User- or context-driven adaptation (for example, ActiveX not allowed)	3
Replacing or upgrading middleware, a webserver, or a database	2
Minimizing merge complexity	2

TABLE 4

A summary of the performance concerns.

Concern	Percentage of scenarios
Maintaining response time for a system capability (simulation, display, backup, run report, and so on)	25
Maintaining response time for critical capabilities with limited or intermittent resources	25
Maintaining response time while the hardware or platform changes (for example, a different machine or new CPU architecture)	21
Maintaining response time while the load or usage increases (increased load, increased service demands, increased fidelity, and so on)	9
Maintaining start-up or shut-down response time threshold	8
Monitoring system or network performance	6
Processing transactions with an increased load within an acceptable degraded range (for example, increased transactions or larger files)	4
System capability working as expected	2

the Study 2 IT projects. This quality attribute has emerged only recently (it was only in the 2006–2013 data), so we looked more closely at the related scenarios.

Table 6 lists the deployability-related scenarios from the ATAM

data. Although these concerns cross-cut many existing quality attribute areas, such as modifiability, we found that the goals driving deployability were different, as were the response measures. Scenarios 1 through 4 aimed to reduce

complexity and reduce deployment and testing cycle times. Scenarios 5 and 6 emphasized minimizing ripple effects due to change; however, the overarching stakeholder concern was change management for a complex distributed deployment environment.

TABLE 5

A summary of the availability concerns.

Concern	Percentage of scenarios
Critical operational capabilities (server, network card, and so on) remaining intact after a hardware failure (or blocked access)	34
Detecting faults, updating logs, or monitoring outage trends (for example, a software component failure or bad node)	19
Critical operational capabilities remaining intact after a software failure or bad data input	15
Restarting after a failure without losing data or state information (rollback, resynchronize, and so on)	11
Critical operational capabilities remaining intact after a network or messaging failure	9
Critical operational capabilities remaining intact after a synchronization failure (for example, clock time or data sources)	6
Visibly notifying the user if hardware or software components fail	6

TABLE 6

Deployability-related scenarios.

Scenario	Description
1	A new release is being planned; the final upgrade plan must be developed in a week (considerable time is spent on configuring specific installations).
2	An upgrade is being pushed out; the software is robust enough to handle 99 percent of errors, leaving 1 percent for manual handling.
3	All full controller builds must be completed in less than 5 minutes (even under the build farm's peak load).
4	The average time to create or build a new system-level test case is one day.
5	One team of developers modifies release version x , while another team makes architectural changes to versions x and $x + 1$ that aren't synched with the first team's changes. The first team integrates the modifications into version $x + 1$ within eight weeks.
6	A new version of the spec isn't compatible with previous versions. The software application and adapter should be backward-compatible, besides supporting the new version.

Reducing deployment and automated testing cycle time is a focus for books on this topic, such as Jez Humble and David Farley's *Continuous Delivery*.⁸ These scenarios could give teams a starting place for reasoning about ways to enable their deployment goals.

Using the Data

The data and concrete examples we present here can help agile teams determine which qualities to focus on when working on early increments and planning future increments. Teams can incorporate high-ranking

quality attributes and quality attribute concerns (including examples) as checklists at designated points during the incremental life cycle.⁹ This will help them quickly assess whether they need to consider a specific quality attribute concern to support an anticipated business need.

Evolving a system's architecture in an agile project presents unique challenges. We suggest weaving quality attribute concerns into the life cycle that a project team is using; this provides natural opportunities to incorporate the considerations we present here. For example,

during agile retrospective meetings, the team might brainstorm about quality-attribute-related deficiencies that surfaced in the prior increment. During sprint planning, the architect could propose addressing a quality attribute concern and, if the product owner agrees, add it to the sprint backlog. This data also provides ammunition for the agile team lead in making the business case to the product owner by justifying the importance of dealing with such concerns architecturally.

Table 7 can also serve as a checklist for team members reflecting on


TABLE 7

Quality attribute concerns grouped by quality attribute and ranked according to the number of scenarios in which they appeared.

Quality attribute	Concern
Modifiability	Designing for portability
	Designing for extensibility
	Reusability
	Configurability and composability
	Increasing cohesion
	Adding or modifying functionality
Performance	Latency
	Resource management
	Throughput
	Performance monitoring
	Initialization
	Accuracy
Interoperability	Upgrading and integrating with other system components
	Ease of interfacing with other systems or components
	Resource and data sharing
	Data integrity
	Compliance with standards and protocols
Availability	Fault detection
	Fault recovery
	Preventing faults
	Transaction auditing and logging
	Graceful degradation

quality attribute concerns. Quality attribute concern data can help agile developers create review questions for reasoning about design trade-offs, such as, “Did we negatively impact another architecturally significant quality attribute concern with this design decision?” Jungwo Ryoo and his colleagues described an example of applying quality-attribute-concern and design-tactic data in this way.¹⁰

Our ultimate aim is to provide systematic support for iteratively addressing architectural concerns in agile projects. Techniques for efficiently identifying, prioritizing, and allocating architecture tasks to iterations and measuring their ongoing effects on project velocity are fundamental to this work’s success. Regarding future applicability, data such as this

could also serve as input to decision-support-system models¹¹ that provide automated support for reasoning about quality attribute tradeoffs. 

Acknowledgments

This article is based on research funded and supported by the US Department of Defense under contract 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-0002047

References

1. N. Brown et al., “Managing Technical Debt in Software-Reliant Systems,” *Proc. FSE/SDP Workshop Future of Software Eng. Research*, 2010, pp. 47–52.
2. S. Ambler, *Disciplined Agile Delivery: A Practitioner’s Guide to Agile Software Delivery in the Enterprise*, IBM Press, 2012.
3. D. Leffingwell, *Scaling Software Agility: Best Practices for Large Enterprises*, Addison-Wesley, 2007.
4. S. Bain, *Emergent Design: The Evolutionary Nature of Professional Software Development*, Addison-Wesley, 2008.
5. S. Bellomo, R. Nord, and I. Ozkaya, “A Study of Enabling Factors for Rapid Fielding: Combined Practices to Balance Tension between Speed and Stability,” *Proc. 2013 Int’l Conf. Software Eng.*, 2013, pp. 982–991.
6. I. Ozkaya et al., “Making Practical Use of Quality Attribute Information,” *IEEE Software*, vol. 25, no. 2, 2008, pp. 25–33.
7. P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*, Addison-Wesley, 2001.
8. J. Humble and D. Farley, *Continuous Delivery*, 1st ed., Addison-Wesley, 2010.
9. F. Bachmann et al., “Integrate End to End Early and Often,” *IEEE Software*, vol. 30, no. 4, 2013, pp. 9–14.
10. J. Ryoo, P. Laplante, and R. Kazman, “In Search of Architectural Patterns for Software Security,” *Computer*, vol. 42, no. 6, 2009, pp. 98–100.
11. N. Ernst and I. Gorton, “Using AI to Model Quality Attribute Tradeoffs,” *Proc. IEEE 1st Int’l Workshop Artificial Intelligence for Requirements Eng. (AIRE 14)*, 2014, pp. 51–52.



Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.

IEEE computer society NEWSLETTERS

Stay Informed on
Hot Topics

COMPUTING NOW
TRAINING SPOTLIGHT
TRANSACTIONS CONNECTION
WHAT'S NEW BUILD YOUR
IN COMPUTER CAREER COMPUTING DIGITAL
LIBRARY NEWS FLASH
CS CONNECTION NEWS FLASH
DIGITAL LIBRARY NEWS FLASH
CONFERENCE CONNECTION
NEW IN COMPUTER
BUILD YOUR CAREER
TRANSACTIONS MEMBER
CONNECTION CONNECTION
COMPUTING NOW
TRAINING SPOTLIGHT
CS CONNECTION



computer.org/newsletters

ABOUT THE AUTHORS



STEPHANY BELLOMO is a senior member of the technical staff on the Software Engineering Institute's Architecture Practices team. Her research interests include architecting for agility, technical debt, and DevOps and continuous delivery. Bellomo received an MS in software engineering from George Mason University. She's a senior member of the IEEE Computer Society. Contact her at sbellomo@sei.cmu.edu.



IAN GORTON is a senior member of the technical staff on the Software Engineering Institute's Architecture Practices team, where he investigates issues related to software architecture at scale. His research includes designing large-scale data management and analytics systems and understanding the inherent connections and tensions between software, data, and deployment architectures. Gorton received a PhD in computer science from Sheffield Hallam University. He's a senior member of the IEEE Computer Society. Contact him at i.gorton@neu.edu.



RICK KAZMAN is a professor at the University of Hawaii and a principal researcher at the Software Engineering Institute. His primary research interests are software architecture, design and analysis tools, software visualization, and software-engineering economics. Kazman received a PhD in computational linguistics from Carnegie Mellon University. He's a senior member of the IEEE Computer Society. Contact him at kazman@sei.cmu.edu.

Call for Articles

IEEE Software seeks practical, readable articles that will appeal to experts and nonexperts alike. The magazine aims to deliver reliable information to software developers and managers to help them stay on top of rapid technology change. Submissions must be original and no more than 4,700 words, including 200 words for each table and figure.

Author guidelines: www.computer.org/software/author.htm
Further details: software@computer.org

www.computer.org/software

**IEEE
Software**