Complementing Agile SDLC with Agile Architecture

By David Shilman

Today's highly competitive and customer-centric market conditions have pushed software and solution delivery organizations beyond the traditionally accepted limits of software development and delivery capabilities. Lean methodologies such as Lean Six Sigma and DevOps can help improve operational solution delivery capacities through

- Streamlining of solution delivery process
- Improved software quality
- Automation of system operations
- Self-administration of system operations by development teams

Agile methodologies augment such operational improvements with their own enablement of faster time to market (TTM) by transforming the Lean concept of value-added activities into value-added product features. Agile software architecture augments solution delivery organizations' Agile software development life cycle (SDLC) capabilities with flexible architectures that facilitate future product development.

Agile methodologies extend product lifecycles through faster TTM and continuous delivery to ensure high quality of service (QoS) defined by functional feasibility, business and technology capabilities. The Agile SDLC produces the Minimally Viable Product (MVP), which is continuously enhanced with value-added product features. While most architects have limited knowledge of future trends in software products, they must still design software to make future seamless enhancements possible throughout the entire product life cycle. This type of application architecture fits the definition of Agile. Agile application architecture must support and complement the Agile SDLC through the following architecture principles, which are derived from the Agile Manifesto principles:

- Application architecture must be extendable: Build for change
- Application architecture must enhance developers' productivity
- Application architecture must not contribute to technical debt
- Application architecture must continuously improve

"Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage." - Agile Manifesto Principle #2

Build for Change

A former CIO of a Fortune 300 company once said "Enterprise architects must be at least six months ahead of the solution delivery organization they support". This can be understood as advice to enterprise architects to keep current with technology trends and educate their peers

in order to influence organization's solution delivery capabilities. Although that might have been his intended message, in the context of Agile SDLC, there is another interpretation: Application architects must produce architecture that will *empower* rather than just support solution delivery teams in delivering and maintaining software with the optimal balance of schedule, cost and non-functional quality.

Application architects provide true value to solution delivery teams by anticipating and initiating technological advancements in areas such as customer communication, information management, security and integration, and computing platforms in their organizations. They prepare for these changes with simple, structurally and behaviorally loosely-coupled application architecture that is agnostic of the client device and hosting infrastructure. In other words, agile application architectures enable developers to replace components and change how they communicate with each other through software configuration. (Table 1). Such loosely-coupled architectures enable their owners to rapidly adopt successive generations of mobile and cloud technology as business needs dictate.

Table 1. Examples of design patterns for structural and behavioral loose coupling

Pattern	Gang of Four Design Pattern	Type	Description
Inversion of Control (IoC)	Factory	Structural	IoC is a modern day reinvention of the Factory Gang of Four (GoF) design pattern. IoC, also known as Dependency Injection, decouples an application component interface from its implementation, allowing application developers to swap application components through software configuration rather than extensive code changes (Figure 1).
Messaging	Mediator	Behavioral	This message-centric integration design pattern implemented with Message Oriented Middleware (MOM) allows application developers to configure run-time component interactions without writing code. (Figure 2).

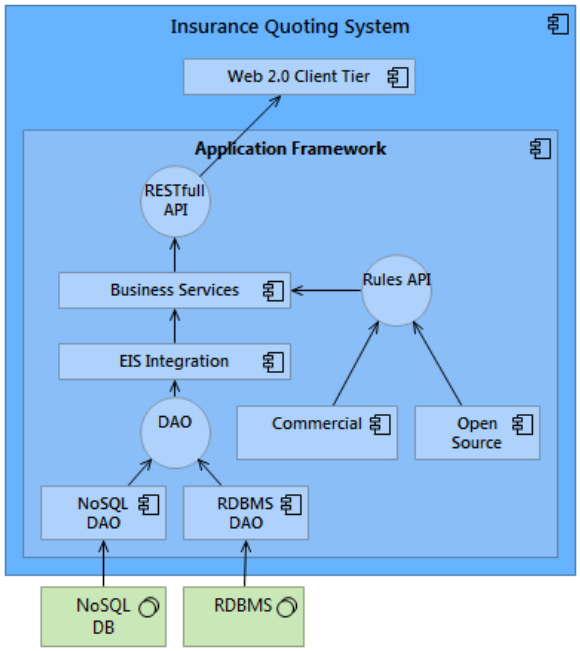

Figure 1. Structurally loosely-coupled application architecture using Data Access Objects (DAOs) to facilitate replacement of the underlying persistence mechanism.

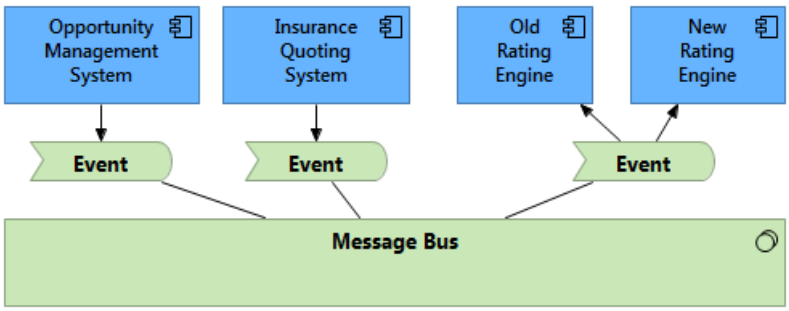

Figure 2. Behaviorally loosely-coupled system architecture using a configurable Message Bus to control component interactions.

"Continuous attention to technical excellence and good design enhances agility." - **Agile Manifesto Principle #9**

Application Architecture Must Enhance Developers' Productivity

The Kanban method introduced the concept of managed flow constrained by Work-In-Progress (WIP) limits, which quantitatively describe an organization's operational capacity. WIP limits are usually influenced by solution delivery organizations' head count and SDLC process efficiency. Good application architecture can also improve solution delivery capabilities by increasing the WIP limits.

Various studies show that most IT budgets allocate as much as 80% to software maintenance. Application architects must not only enable rapid solution delivery, but also reduce product maintenance cost throughout the software's entire life span by enhancing quality of service (QoS) and developer productivity.

Application architects should strive to free application developers of all non-functional coding by allowing them to focus on and efficiently implement features with direct customer value. Table 22 illustrates the agile application architecture practices that best empower application developers.

Table 2. Agile Application Architecture Practices.

Pattern	Description
Loose coupling of application tiers	Loose coupling of application tiers through JavaScript Object Notation (JSON) or eXtensible Markup Language (XML) allows application developers to program application components simultaneously based on a flexible tagged data structure rather than a data object that creates compile dependencies between application components. Developers can program loosely-coupled components in parallel without impacting one another, and reconcile their changes as they merge their work.
RESTful application programming interfaces (APIs)	Exposing server side data and business components through stateless RESTful APIs based on straightforward web URLs allows reuse of application components a wide variety of distributed clients. This approach significantly reduces server-side application footprint and the prospect of technical debt.
Open source commons libraries	Popular open source frameworks such as Apache Commons and Google Guava help reduce development effort by eliminating boiler-plate code and reducing application footprint
Single-page applications	Single-page applications, which enhance user experience with rich graphical user interfaces and instant response, are typically engineered with browser-based technologies like: • Rich Internet Application JavaScript frameworks • Ajax • CSS
Responsive design	Incorporating responsive design into a web application UI allows access from a variety of web/mobile client devices, which reduces the prospect of technical debt as client technologies progress.
NoSQL (Not Only SQL) databases	Among other uses, NoSQL databases excel at storing transient data and dynamic data structures without extensive support from data architects and database administrators.

Together, these practices shorten the critical path of development projects. They enable parallel development with reduced dependencies between application components and tiers, decrease application footprint with increased software reuse, reduce data administration and operations efforts with newer database technology, and therefore improve prospects for continuous delivery.

"Simplicity--the art of maximizing the amount of work not done--is essential." - Agile Manifesto Principle #10

Application Architecture Must Not Contribute To Technical Debt

There is a strong demand from business partners and customers to access end-user facing applications from a variety of web and mobile devices. Modern web application architecture must incorporate multichannel content delivery and responsive design in order to meet this business demand with efficiently. Failure to adapt to customer needs quickly will reduce the solution delivery organization's competitive advantage and contribute to its technical debt. Application architects need to challenge themselves and embrace new and unfamiliar technologies geared to enhance end-user experience (UX) with minimum redundancy in technology assets (Figure 3).

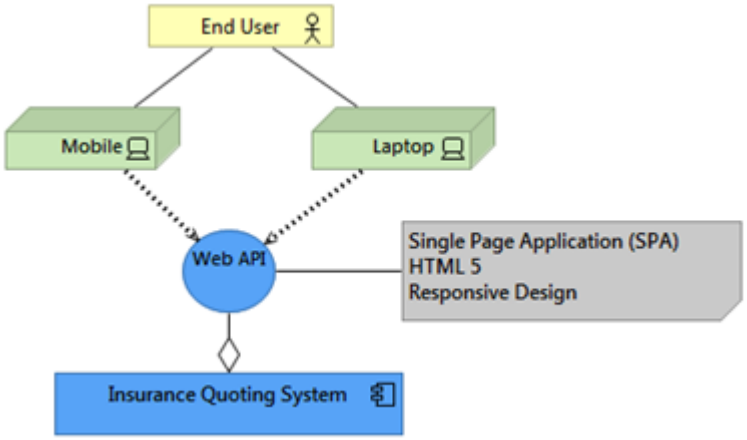

Figure 3. An Agile application architecture that enables reusability across client platforms.

"At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly." - Agile Manifesto Principle #12

Application Architecture Must Continuously Improve

During the scrum retrospective, Agile teams improve their sprint execution capabilities. Enterprise Architecture also benefits from continuous improvement. As new technologies and patterns are adopted, architects should incorporate them in the Standards Information Base within the organization's architecture repository. Enterprise architects and development teams need to continuously identify opportunities to improve collaborative architecture development and governance and to respond faster to architecture changes in order to provide value and build agility.

Conclusion

In today's economy, business and IT stakeholders often make decisions to buy versus build based on software solution quality, cost and time to market delivery capabilities. Furthermore, customers' expectations of user experience have continued to rise. IT organizations can no longer measure application quality by yesterday's internally-focused standards without running the risk of producing applications classified as "legacy" rather than "modern" after only a short time in production.

An earlier version of this article originally appeared in the Cutter Consortium Business and Enterprise Architecture Practice -
http://www.cutter.com/content/architecture/fulltext/advisor/2014/ea140723.html

About the Author:

David Shilman is an Enterprise Architect with a Fortune 300 company. David started his professional career as a programmer. Throughout his career development, David has embraced technology and SDLC best practices. In his previous role as an application development manager, David was an early adopter of Lean and Agile practices and methodologies, which he utilized to streamline and improve the solution delivery capabilities of his team. In his current role, David spearheads TOGAF awareness and adaptation in his organization. David has a B.S. in Mathematics, and holds advanced certifications in Enterprise Architecture, Project Management and Lean Six Sigma. David is a father of five, and enjoys spending time and playing sports with his kids.