Delivering Software-Reliant Products Faster

Take action to help your organization gain speed without sacrificing quality

1. Producing high-quality, robust products and delivering them faster depends on software development that's probably not delivering everything needed— that's the bad news.
2. The good news is that achieving high performance and speeding time to market can be accomplished by making better use of software architecture.
3. The better news is that there are steps you can take now to use software architecture more effectively.

Leading organizations that develop software-reliant products emphasize getting to the market or field faster with offerings of exceptional quality that can meet changing customer needs.[1,2]

Research over the past two decades suggests that taking a product-development approach centered on software architecture significantly improves an organization's chances for achieving this business goal. Software architecture is an engineering blueprint that can guide each phase of product development toward success.[3,4,5]

In this paper, we focus on three important ways to improve software-reliant products by making more effective use of software architecture throughout development:

1. Better quality—delivery of product qualities that fulfill customer needs and expectations
2. Faster delivery—reduction of unnecessary rework that delays product introduction
3. Easier maintenance—improved flexibility for changing the product to meet evolving customer needs

We also suggest some steps an organization can take to learn more and move toward adopting a product-development approach that makes more effective use of software architecture.

Software Engineering Institute
Carnegie Mellon

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890
www.sei.cmu.edu

Delivering Software-Reliant Products Faster

Take action to help your organization gain speed without sacrificing quality

Deliver products that fulfill customer needs and expectations

One prominent study says that just one-third of software projects deliver the features and functionality customers expect.[6] Other studies indicate that half of information-system software development projects fail.[7]

The failure to deliver the qualities that customers want has real consequences. One example is the web-based purchasing system developed for Ford Motor Co.[8] The system, dubbed Everest, was intended to automate manual procurement operations that had proven to be costly, such as the exchange of invoices and other common standard documents. Rich functionality and integration were keys to the success of this system. But suppliers reported that the new system was very time consuming because they had to use the existing environment along with Everest to accomplish the same things they had done before. Lackluster supplier reaction caused Ford to abandon Everest soon after fielding it. The organization lost more than $400 million, along with five years of development. In addition, the organization did not gain benefits that were promised from the automation of manual procurement processes.

Product development that focuses on software architecture from the beginning allows an organization to verify that the product can deliver the system qualities customers want, such as security, performance, modifiability, availability, and interoperability. Software architecture provides (1) a forum in which all product stakeholders can work through tough questions about how the product will behave in terms of its important qualities and (2) the artifact in which those qualities can be verified.

Our experience: A focus on software architecture to better understand and deliver product qualities

The SEI has evaluated 12 large-scale-system development projects for a government organization to gauge the effect of using two software architecture methods, the Quality Attribute Workshop (QAW) and the Architecture Tradeoff Analysis Method® (ATAM®).[9] The QAW provides a means to discover a system's critical qualities early in development. It complements the ATAM, a method for evaluating an architecture relative to those critical qualities.

We asked representatives of the 12 projects to tell us to what degree these methods helped them discover key qualities, assess whether their architectures would deliver these qualities, and identify their risks. All of the projects reported improvement in these areas. Seventy-five percent of the projects characterized this improvement as *significant* or *very substantial*. All of the projects that used these methods early in development reported that their use produced tangible benefits in system quality and effectiveness.

An added benefit emerged from the study as well: better communication among the product stakeholders. Project managers, designers, suppliers, and other stakeholders were able to "achieve a common understanding" that makes it "more likely that the completed product will address stakeholder expectations and user needs."[9]

Projects that used SEI software architecture methods early in development reported tangible benefits in system quality and effectiveness.

® Architecture Tradeoff Analysis Method and ATAM are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

Delivering Software-Reliant Products Faster

Take action to help your organization gain speed without sacrificing quality

Gain speed by eliminating unnecessary rework

Organizations know that they need to field their products faster than the competition. But, just as it often fails to deliver desired qualities, traditional software-reliant-product development also typically does not meet estimated schedules. On average, time to release exceeds planned development time by 84%.[7]

A major contributor to boosting development time is rework to fix errors, accounting for somewhere between 25 and 40% of the total time, analysts say.[10] Moreover, when errors are not caught until *late*—during system-testing or, worse, after the system is delivered to the customers—it takes a lot more rework, and a lot more time, to fix them.[11] One large study of IT systems pegs the additional time and rework for late fixes as high as 30 times more.[12] For its high assurance systems, NASA has determined that the rework factor can grow to 100 times more.[13]

Rework can have consequences in lost revenue as well. For example, the Airbus A380 aircraft was plagued by several delays traced to a lack of interoperability among software applications used in different Airbus design and engineering centers. The interoperability problem contributed to the need for many changes that compounded into more than a year's delay, a loss of $2.5 billion in profit, and the replacing of a project manager and a CEO.[14]

Attention to software architecture can help an organization avoid costly, time-draining, unnecessary rework early in development and during the crucial system-integration phase. Integration is a major problem area for organizations building products from systems developed separately by different third-party organizations or migrating legacy systems to new environments. Architecture evaluation—applied from the earliest stages of product development—brings to light risks to the delivery of important qualities and points to ways to address risks before they introduce errors into products. Architecture modeling and analysis provide improved system understanding that can facilitate integration.

Our experience: A focus on software architecture to reduce rework

The SEI, as part of a global cooperative of aerospace companies, government organizations, and academic institutions, recently completed a study showing how an approach known as virtual integration—as part of an architecture-centric practice—can prevent wasteful rework in the development of aircraft software systems. The study proved how architecture methods, including the industry standard Architecture Analysis and Design Language (AADL), allow virtual integration that flips the traditional development approach of *build then integrate*.

By following an *integrate then build* approach centered on building and analyzing software architecture, system designers can make more informed decisions that avoid errors, and they can detect errors as early as possible. The study estimated, conservatively, that architecture-centric practice can prevent about a quarter of wasteful rework effort in the development of aircraft software systems.[13]

By following an integrate then build approach centered on building and analyzing software architecture, system designers can make more informed decisions that avoid errors, and they can detect errors as early as possible. An industry study estimated, conservatively, that architecture-centric practice can prevent about a quarter of wasteful rework effort in the development of aircraft software systems.

Delivering Software-Reliant Products Faster

Take action to help your organization gain speed without sacrificing quality

Plan for smoother change and maintenance

At some point, a software-reliant product is bound to face circumstances that trigger the need for product maintenance and enhancement such as

- new customer requirements
- need to connect with other systems
- changes to the environments in which it is used
- improvements by the competition

For the organization, responding to circumstances such as these can account for more than half the cost of the product over its lifespan.[15]

When a product proves difficult to modify, the organization can suffer in several ways, as illustrated by a computerized billing and claims processing system that was developed for Oxford Health Systems. A significant increase in data throughout the system choked its performance.[16] Technical problems termed "pervasive and debilitating" arose when the health insurer doubled its business over 18 months and could not upgrade the system to keep pace. Customers enjoyed an unexpected (and unexplained) holiday from paying on their policies for several months. At the same time, the health insurer resorted to advancing reimbursement to hospitals, with an agreement to make good any differences when the system was upgraded. Seeing that the organization had less money coming in and more going out, investment analysts calculated that Oxford had overstated its quarterly revenues by $400 million; predictably, the publicly traded company's stock plummeted—$3.4 billion in value in one day.

Product development driven by deliberate attention to architecture can immunize the product from the effects of change in two ways. In anticipation of changes such as increased demand that might occur after the product is released, the system designer can isolate, during development, the areas of the software architecture that would require modification and evaluate how adaptive the system will be. Or, to meet a need for modification after the system is in use, the system designer can find areas in the architecture where change is needed, make changes to those areas, evaluate the changes, and deploy the modification with less effort.

Our experience: A focus on software architecture to ensure modifiability

Our architecture-centric approach enables an organization to act with *informed anticipation* to avoid *over*-anticipating emerging needs and *under*-anticipating future needs.[17] Acting with informed anticipation, an organization would find that it

- delivers customer-facing features without delay due to exhaustive requirements and design activities and reviews
- maintains a steady focus on continual architectural evolution for ready response to customer needs as they emerge

One example of how an architecture focus can allow an organization to begin to act from informed anticipation is the experience of a large retail organization. This retailer needs to be responsive in its development. It found, however, that its technology infrastructure had become "difficult to evolve"[18] through repeated code updates to meet increasing demand for new systems, more richly featured websites, integration with other products, and security concerns, among other needs. Despite using sound software-development practices, the retailer saw its essential code base devolve into entangled interdependencies that increased its system costs.

The organization changed its focus to increase both visibility and governance of the infrastructure software architecture. As a result, the retailer realized lower cost through a 10% reduction in the number of files changed in making an update and a greater understanding of the interdependencies in its infrastructure.

Gaining Speed Without Losing Quality: A Plausible Scenario

An organization envisions moving a complex system to a new technology environment in order to give customers greater flexibility. Not making the move successfully and as quickly as possible could damage the organization's revenue and reputation.

The organization learns that an *architecture fact-finding* diagnostic approach will help connect essential benefits with the software system qualities that provide them. As part of this approach, architecture experts *model the system architecture* at a high level, revealing that system qualities must be more fully articulated.

To refine system qualities, architecture experts engage all of the system-development stakeholders in forming *scenarios* that show what is likely to happen when one quality is preferred over others, an activity known as a *tradeoff*. Analytical software tools can be used against the architecture to quantitatively *predict system performance* in the new environment. This analysis will spot errors in design long before they prove expensive and time-consuming to fix. Predictable engineering of the system *eliminates unnecessary rework* that delays system implementation; at the same time, it keeps a focus on delivering key system qualities.

As a by-product to the organization's adoption of an architecture-centric approach, its software architects, technical managers, and others gain knowledge and skills through *training in architecture practice*. Like their now-migrated, more-robust system, they are in a position to do more and offer more.

Delivering Software-Reliant Products Faster

Take action to help your organization gain speed without sacrificing quality

Steps to Take Toward Using Software Architecture More Effectively

Many approaches and methods exist for developing, documenting, and evaluating an architecture. At the SEI, in fact, we have created and matured several that can aid an organization in adopting an architecture-centric approach to software-reliant product development. We suggest these steps for moving toward making more effective use of software architecture:

1. Gain insight about whether architecture-centric practices can help your organization

First, gauge the effectiveness of the organization's product development practices by asking key questions such as

- What are our strategies to make the product successful?
- How do we identify and verify qualities that are important to customers?
- Is there any uncertainty that needs to be managed in the development of this product? What is its nature—cost, schedule, functionality, future upgrades or uses?
- Do we know how much rework costs us in time and money?

Then, consider using our **automated self-help diagnostic** about organizational readiness to incorporate an architecture-centric focus. Our automated diagnostic is in development now. If you are interested in helping to test this new tool, or in discussing the implications of questions about architecture practice areas for your organization, call us at +1 412-268-5800, write to info@sei.cmu.edu, or visit http://www.sei.cmu.edu/goto/action.

2. Start a conversation about how architecture-centric practices can help by calculating your potential dividend

Avoiding rework can cut development costs, as well as time. The money saved from reducing rework can be applied to other improvements. Think of this as a dividend from putting a focus on architecture throughout development. Here's the formula

*Cost avoidance = Estimated cost * % Rework * % Requirements errors * Removal efficiency*

Say, for instance, that the total system development cost will be $1M and rework will amount to 50% of that cost. Further, 70% of the rework cost will be due to the discovery of requirements errors in phases later than the requirements phase. If removal efficiency (that is your discovery and fixing of a defect in the phase where it is introduced) were just a conservative 33%, the organization would see a dividend of $115,500 (1M*0.5*0.7*0.33). Share the dividend estimate with the organization's software development teams and let us know, too, at info@sei.cmu.edu.

3. Join a discussion about designing products for adaptability

Get a sense of current practice in other organizations by looking into our SEI Architecture Technology User Network (SATURN). We organize an annual SATURN Conference that brings together architects and others involved in product development in industry and government. To find out more, visit http://www.sei.cmu.edu/goto/action.

4. Become skilled in architecture-centric practice

By gaining essential skills in software architecture, you can really get going on improving how rapidly your organization delivers innovative, robust, market-winning software technology products. Take a first step by enrolling in our *Software Architecture: Principles and Practices* course. Or you might suggest the course to others in your organization who are involved in the development of software-reliant products. For more information and to register, go to www.sei.cmu.edu/go/sapptraining/, contact us as course-info@sei.cmu.edu, or call us at +1-412-268-7622.

About Us

We're the Software Engineering Institute, a federally funded research and development center based at Carnegie Mellon University. We work closely with defense and government organizations, industry, and academia to continually improve software-reliant systems.

A primary focus of the SEI is architecture-centric engineering practice, including

- proven technologies for architecture evaluation
- four widely acclaimed books on software architecture
- a four-course curriculum leading to mastery in software architecture practice
- an annual conference (SATURN) on software and systems architecture practices
- ongoing research into architecture issues facing organizations

Delivering Software-Reliant Products Faster

Take action to help your organization gain speed without sacrificing quality

1 *Measuring and Sustaining the New Economy, Software, Growth, and the Future of the U.S. Economy*, The National Academies Press (available from www.nap.edu/catalog.php?record_id=11587#toc).

2 Aberdeen group study, www.aberdeen.com/Aberdeen-Library/4170/RA-inovation-agenda-2010.aspx

3 For example, see www.dtic.mil/ndia/2008systems/7137baldwin.pdf, www.sei.cmu.edu/architecture/, http://www.zdnetasia.com/companies-clueless-about-software-architecture-61952450.htm and http://www.ibm.com/developerworks/webservices/library/wi-arch13.html.

4 Len Bass, Paul C. Clements, and Rick Kazman, *Software Architecture in Practice, 2nd edition* (www.sei.cmu.edu/library/abstracts/books/0321154959.cfm)

5 Paul Clements. Rick Kazman, and Mark Klein. *Evaluating Software Architectures: Methods and Case Studies* (www.sei.cmu.edu/library/abstracts/books/020170482X.cfm)

6 These figures come from reports that have been provided by the Standish Group since the mid-1990s.

7 Grafton Whyte and Andy Blytheway, *Factors Affecting Information Systems' Success*. (www.emeraldinsight.com/journals.htm?articleid=851592&show=html)

8 http://adtmag.com/articles/2004/11/01/oops-ford-and-oracle-megasoftware-project-crumbles.aspx and http://www.accessmylibrary.com/article-1G1-121520297/ford-giant-purchasing-glitch.html

9 Robert Nord, John K. Bergey, Stephen Blanchette, Jr., and Mark H. Klein. *Impact of Army Architecture Evaluations* (CMU/SEI-2000-SR-007) (www.sei.cmu.edu/library/abstracts/reports/09sr007.cfm)

10 http://www.ibm.com/developerworks/rational/library/347.html and http://www.ikmagazine.com/xq/asp/txtSearch.Measuring/exactphrase.1/sid.0/articleid.7A69C140-4A6A-451D-90A8-AABBFA2C4EA5/qx/display.htm

11 Boehm, Barry & Basili, Victor. "Software Defect Reduction Top 10 List." Software Management (May 2001): 135-137.

12 *The Economic Impacts of Inadequate Infrastructure for Software Testing*, in *NIST Planning report* May 2002, NIST.

13 Peter Feiler, Jorgen Hansson, Dionisio de Niz, and Lutz Wrage, *System Architecture Virtual Integration: An Industrial Case Study* (CMU/SEI-2009-TR-017) (www.sei.cmu.edu/library/abstracts/reports/09tr017.cfm). The proof-of-concept is part of a program called System Architecture Virtual Integration (SAVI). The Aerospace Vehicle Systems Institute (AVSI) launched SAVI. AVSI is administered by the Texas Engineering Experiment station at Texas A&M University.

14 http://usadailycut.com/2010/06/21/malaysian-air-may-cancel-a380-order-if-delivery-delayed-further/, www.bloomberg.com/news/2010-05-04/airbus-a380-superjumbo-risks-turning-into-double-decker-dud-as-orders-lag.html, and http://www.bloomberg.com/apps/news?pid=newsarchive&sid=aSGkIYVa9IZk.

15 http://users.jyu.fi/~koskinen/smcosts.htm

16 http://www.businessweek.com/1997/46/b3553148.htm

17 Nanette Brown, Robert Nord, and Ipek Ozkaya. "Enabling Agility Through Architecture." *Crosstalk* (November/December 2010).

18 http://www.springerlink.com/content/p3208208q6654408/fulltext.pdf