Architects as Service Providers

Roland Faber, *Siemens*

By providing architecture as a service to application developers, architects participate in coding activities and sustain the architecture's effectiveness throughout a project's lifetime.

Architects traditionally emphasize the stability of software systems' overall structure, fixing it into diagrams and descriptions for communication with developers. But as implementation progresses, those early-defined structures and rules often prove suboptimal when the growing knowledge about the system implies structural changes and the static nature of the architectural settings becomes an obstacle. To resolve this, a more general concept of software architecture should include structural system aspects as part of a wider concept of system qualities. Architects provide those system qualities as values to their customers, communicating and implementing them in close cooperation with developers. In this way, architects also can and should play an important role in agile development projects.

Motivation

Years ago, dedicated teams at Siemens implemented the architectures and frameworks of medical diagnostic imaging systems by hardening the architecture before application development started. Only in exceptional cases did architects communicate directly with application developers. So, requests for architectural changes were frequently raised too late for fulfillment in the required time frame. Therefore, application development efficiency suffered from costly adaptations to the frameworks. Huge investments in modifying existing applications made architectural rework difficult.

From this experience, software development leads at Siemens concluded that communication between architects, framework providers, and application developers must start as early as possible and intensify throughout the projects. This is achievable only when all participants continuously benefit from communication, which demands a clear agreement on the partners' roles, expectations, and responsibilities. The approach I established for my team of architects follows these basic elements:

- The goal of architectural work is the fulfillment of nonfunctional system and implementation requirements.
- Architects are responsible for nonfunctional system qualities; developers are responsible for functionality.
- Architects create benefits for developers by actively contributing to their realization.
- An early phase of preparation enables architects to successfully support developers later.
- Direct, personal interaction and close cooperation is preferred over written communication.
- Architecture communication is actively stimulated by dedicated roles and architecture boards during implementation.

0740-7459/10/$26.00 © 2010 IEEE

There should be as much documentation as necessary, not more, and direct communication should accompany it.

Jutta Eckstein describes architecture as a service for the development team.[1] In my team's method, architects act as service providers for their clients, both application developers and customers, in the agile and conventional software development processes.

The Value of Service

A service provides value to a client. The value that architects provide to their clients is how well they address the requirements behind the architecture. Len Bass and his colleagues describe architecture's value as the system-specific fulfillment of nonfunctional quality requirements.[2] Consequently, architects are the stakeholders of overall system quality: they balance the investment in certain system qualities by satisfying specific requirements. Philippe Kruchten and his colleagues describe this activity as the selection of concrete design options.[3]

The ISO 9126 standard groups qualities into characteristics—namely, reliability, usability, efficiency, maintainability, and portability.[4] Bass and his colleagues add the characteristics of availability, modifiability, performance, security, and testability.[2] Linda Rising, during her visit to Siemens in June 2009, also mentioned consistency and minimalism: each software product is characterized by its specific qualities and associated priorities. Whereas qualities such as modifiability, testability, and minimalism directly affect development, others are more customer-related. This illustrates that the architects' clients are both developers and customers.

For intensive communication, architects should participate closely with application developers' work, including hands-on coding. Although it depends on the particular context—for example, on the project's size, from our experience—working closely with development is more efficient than using written specifications. Alistair Cockburn emphasizes the effectiveness of immediate feedback in direct communication.[5]

> ***The missing feature.*** *For improving an image-processing framework, I interviewed experienced application developers. They complained about an important feature they missed. I asked the architects; they claimed that this feature was a core quality of the framework. They didn't emphasize it in the documentation because it seemed so obvious and the developers didn't ask.*

Further reasons for direct communication include the following:

- Written documentation is frequently outdated.
- Writing and reading specifications bear multiple risks for misunderstandings.
- Architects validate their concepts best by experiencing them themselves.
- Architects directly implement system qualities—for example, by developing frameworks.
- Architects gain useful experience in the application domain.

But some documentation might still be required. Architecture specifications are useful to give newcomers or key stakeholders a system overview, especially to guide them through the system or to provide information that can't be incorporated immediately into code. There should be as much documentation as necessary, not more, and direct communication should accompany it.

Oliver Creighton and Matthias Singer explain that architects must "create a certain atmosphere of trust and motivation" to succeed.[6] Through close cooperation, our architects show that they can effectively support the developers. This is key for opening communication about problems with the architecture, because under pressure, developers often try to solve problems quickly on their own. They share problems only when they're convinced they can get help effectively without delay. So, it's important that the architects have a mix of capabilities, knowing when to listen attentively and when to provide direction to the team.

Service Provider Architect Roles

Architects take different roles when interacting with developers and other stakeholders. On the one hand, they must deeply understand the system quality requirements. On the other, they must guide the development to the right balance of quality investments. Depending on the situation, the architect either listens carefully or firmly directs the team's activities.

The Humble Apprentice

As service providers, architects realize that the architecture isn't final and static but an object of change as understanding of the system increases during its development. To keep up-to-date, architects must continuously actualize their application domain knowledge. They get direct access to information sources by directly involving themselves in development activities, including coding with the developers. From there, it's easy to achieve intensive communication with other stakeholders for customer needs such as product manage-

Table 1

Mind-set of a service provider architect between the extremes of weak and excessive guidance

Mind-set	Architect guiding too weakly	Service provider architect	Architect guiding excessively
Client orientation	Changes concepts according to client wishes	Balances concepts with client needs	Expects clients to adapt their needs to his or her concepts
Communication	Asks clients for concepts	Drives concepts in close communication loops	Creates concepts in solitude and publishes them
Learning	Changes his or her mind on the basis of each instance of feedback	Turns feedback into improvements	Ignores feedback
Change management	Lets architecture just grow	Organizes architecture change process	Defends architecture against change requests
	Avoids frameworks	Improves frameworks	Defends frameworks
Practical support	Works as a developer	Supports developers by hands-on coding	No coding, avoids developer contact
Process	Avoids rules	Sets up rules, but helps break them when necessary	Forbids rule-breaking, which nevertheless happens undercover

ment and requirements engineering. Ian Gorton calls this the architect's liaison role.[7] Interacting as humble and attentive listeners, as apprentices who want to learn, architects stimulate open exchange with their partners.

> ***The self-centered architect.*** *Once, I created a simple state machine framework to improve program structure. According to theory, the framework supported a single event per state transition. One colleague was interested but requested the capability of multiple state transition events. We debated for half an hour. While I insisted on the theory, he brought out examples that I couldn't implement easily. In the end, I lost a client and an opportunity to improve the program structure.*

In practice, an architect should take a sound position between the extremes of weak and excessive guidance. Table 1 shows the middle ground for a service provider architect.

The Master Programmer Teacher

As the stakeholders of system qualities,[2] architects must act as masters themselves. While carefully listening to the developers' needs, they actively guide them in the solutions to the problems at hand[6]—for example, developing the required frameworks and showing how to use them. By actively sharing their implementation experience, architects earn the developers' trust to provide effective guidance. If developers don't believe architects are helpful, they won't ask them for guidance.

> ***Building trust.*** *My team established regular teleconferences with a remote group of application architects and achieved some framework improvements for them from the German providers. Later, they accepted our suggestion for an architectural solution, although they preferred a different approach. The architecture remained consistent on the basis of their good experiences with our cooperation.*

Craig Larman and Bas Vodde describe an architect as a master programmer teacher who ensures the system's overall code quality.[8] Usually, developers don't differentiate too much among "architecture," "coding guidelines," "framework," "basic libraries," and "tools." At best, they'll ask the architect, so the architect should have the answers. Once again, this illustrates the importance of practical development skills for building trustful liaisons.

It sounds difficult to act as a humble apprentice and master at the same time, and in practice architects must struggle for acceptance. They achieve it by a clear separation of responsibilities—features for the developers, system qualities for the architects—and by their dedicated support of the developers.

The Architecture Process

To provide their service, architects must first build up knowledge and then transform it to balance system quality. Therefore, I view the architecture process as two phases: preparation and support. Figure 1 gives an overview of important tasks in both phases of our method.

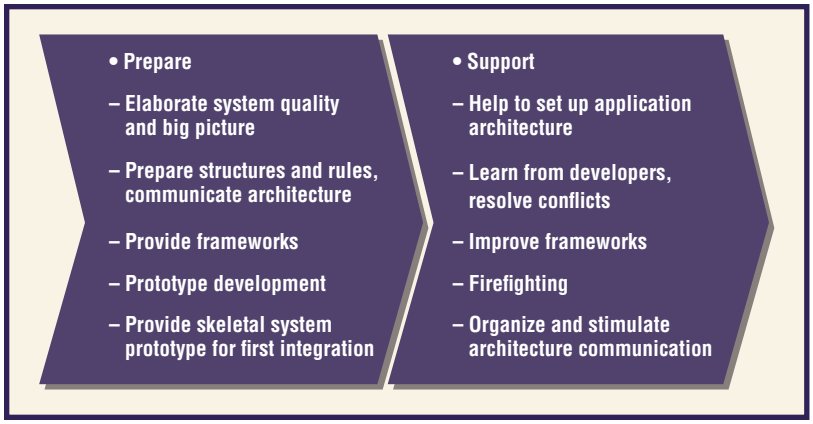

Figure 1. Tasks relevant for our method during preparation and support. During preparation, architects build up their technological and system knowledge, applying it to implement system quality during development support.

Preparation

Architects must gain sufficient technological knowledge for effective development support, and sufficient system knowledge for the start of development, especially in terms of system quality requirements.

System quality specification. Preparation should start with a description of specific system qualities that can later be transformed into tasks for the development teams, such as in system functional use cases. The result is a complete collection of prioritized descriptions of system qualities. Bass introduces the quality attribute scenario, a useful method to describe system quality attributes that supports measurable quality specifications, especially for testing.[2] Gorton describes a simplified variant[7] that I use in the following example:

> *For the treatment of brain stroke, the therapy decision must be made quickly. After completion of a specific magnetic resonance examination the lesion detection and tissue parameter visualization by the radiologist for the diagnostic reading must not take longer than five minutes:*
>
> Stimulus. *Diagnostic images are available for radiologist's reading.*
>
> Result. *Radiologist completed lesion detection and visualization of tissue parameters for therapy decision.*
>
> Quality measure. *Result available at least five minutes after stimulus.*

Big picture. To prepare the developers' work in a project, the overall architecture description should provide sufficient information to

- identify the system parts that are crucial for the system qualities,
- implement the skeletal system, and
- organize the development team into subteams.[2]

The details should provide the guidance developers need to begin implementation.

Structure and rules. Conventions that apply to the entire system are costly in later modifications—for example, underlying frameworks, user interface style guides, coding guidelines, and overall error handling and messaging. Involving application developers early helps architects avoid the one-rule-fits-all pitfall.

> ***Separation overdone.*** *We separated the user interface from business logic into different system processes but after months of work found the investment into related interfaces infeasible. So, we had to modify the rule, which wouldn't have been required if we'd involved the developers earlier.*

Prototypes. The big picture shows crucial system parts for mission-critical qualities—for example, performance—and architects must ensure their feasibility with appropriate prototypes.[7]

> ***Critical performance.*** *We measured the performance of a critical network connection with a certain minimum load for bidirectional traffic. With a simple prototype hooked into the system's skeleton, we proved that quality specifications were fulfilled months before product code was available.*

Skeletal system. What's more helpful to developers than working code? In addition to written specifications, the result of the preparation should be a prototype that shows the new system's connections and interfaces. It doesn't contain functionality (the "flesh") but shows the framework ("bones") and interfaces ("joints") at work. It supports system behavior prototyping, and enables developers to integrate their results early into something similar to the final system. This minimizes later refactoring that system environment modifications cause. The skeletal system's information content is about the same as with an architecture specification—only it

consists of running code. Written documentation serves as a guide through the code.

Architects gain experience with the technology by working hands-on during preparation. After that, they can teach the developers how it's done.

Support

I've already described how architects should participate hands-on in implementation with developers in their teams, fulfilling the role of master programmer teachers. For this, architects frequently work outside their "home" team, so they need to be flexible concerning their workplaces. Architects should go to the development hot spots: the crucial applications, the critical system qualities, and the broadly used frameworks.

Set up application architecture. Being fluent in overall system qualities, architects are skilled at defining application architectures. Together with the application development team they create a skeletal application that's consistent with the skeletal system. Later, the team will fill the skeleton with the "flesh" of functionality, transforming the prototype into product code, and the architects can move on to the next hot spot.

> ***The coy developer.*** *A certain application needed frameworks that had never been combined. Because framework documentation was available, the developer implemented the application right away, without bothering the "busy" architects. Debugging sessions that involved the architects showed that combining the frameworks required additional means of synchronization. Refactoring took several weeks. All participants, including the developer, stated afterwards that a half-hour talk beforehand with an architect would have avoided the problem completely.*

Get feedback, improve frameworks. Participating in application implementation provides an excellent opportunity to get feedback about architecture and framework quality.

> ***Making the framework complete.*** *An architect helped developers of a diagnostic imaging application set up its structure. He soon realized that the framework didn't properly support a required functionality: a huge number of small images in a layout serving as sources for drag-and-drop. Owing to this early finding, a modification of the framework was possible, empowering the team to help themselves.*

Firefighting. The broad knowledge of practically experienced architects qualifies them to solve complicated problems throughout the system. For example, architects from my team intensively support performance optimization during system integration in all projects in which we participate.

Static code analysis can detect architectural flaws, but this happens after coding, and, because of project pressure, developers often postpone those late issues as long as the code does its job. So, it's important to invest as early as possible into the application's architecture before the bulk of the development starts.

Architects shouldn't involve themselves too deeply in the minute details of application development. Otherwise, they become developers taking care of functionality instead of architects ensuring overall system quality. They should focus on parts of the system that multiple teams reuse and therefore have more impact on system quality.

Help Breaking the Rules

Here's the heart of the idea of architects as service providers: architects should help break their own rules. I consider this the most challenging aspect of the approach—for the architects. They love their rules. They've carefully created them. And now they should break them. Why?

Because keeping the rules isn't the goal; the goal is the overall system's quality. Also, architects can't inhibit rule breaking completely, even if they try. The pressure of requirements and the project schedule is too high. Rather, architects must help break the rules correctly. If they don't, developers under stress break rules without notice, likely ignoring the quality balance, thereby leading to inconsistencies or even flaws in the architecture.

> ***Unintended use.*** *A team outside of Europe had to provide a remote control for a user interface implemented in Germany. Because the automation interface was incomplete, they decided to use internal interfaces instead without notifying the architects. The system integration suffered from unexpected problems due to uncoordinated interface changes, and costly refactoring was unavoidable.*

This illustrates the need to gain the developers' trust. If they don't expect the architects' support, they simply won't communicate when they're going to break the rules. In the best case, the architects will find the problem after coding, and the project lead will accept the flaw because "the code does its job."

If they don't expect the architects' support, they simply won't communicate when they're going to break the rules.

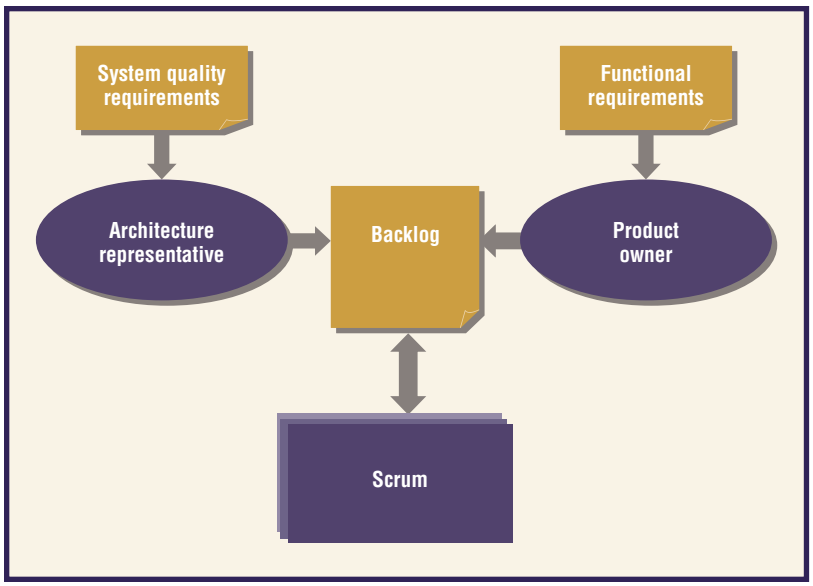

Figure 2. Requirement harvesting by the product owner and architecture representative. They balance the scrum's backlog, filling it with tasks from the functional and system quality requirements.

> ***Keeping quality.*** *Because we helped break the rule of user interface separation from business logic and allowed for a merge into a single process, we could convince the developers to assure decoupling through a well-crafted interface.*

To handle broken rules further, architects transfer them into the next system release preparation phase. Together with the stakeholders—for example, people responsible for product definition, project leaders, and business managers—the architects will decide how and when to repair architectural flaws. In the case of user interface and business logic separation, we decided to abandon the rule of different system processes.

Adaptations for Agility

In the agile approach, a scrum typically consists of a software development team moderated by a scrum master (www.controlchaos.com). A product owner accounts for system functionality and prioritizes the requirements backlog. Full development cycles with fixed time spans, the sprints, iteratively release product versions that the product owner approves. The continued transformation of requirements into development tasks for the sprints is often called requirement harvesting.

We found that the strong role of a functionally oriented product owner requires a counterpart who's responsible for system qualities: the "architecture representative." Eckstein describes a similar role, the chief architect.[1] Currently, we have one architecture representative for four scrums. In close cooperation with the product owner, this person elaborates the backlog, balancing the priorities of tasks for system qualities with those for functionality. Figure 2 shows the requirement-harvesting process involving both the product owner and the architecture representative.

Without the architecture representative, tasks for functionality were frequently prioritized higher, which led to high refactoring efforts when system quality deficiencies became apparent. Changes in system qualities are often more expensive to realize than changes in functionality because the system quality has far-reaching effects on the code. So, it's important to ensure that the system qualities are addressed adequately and as early as feasible.

It's All about Communication

Much more than technology, communication is the key to a software project's success, which Cockburn calls a "cooperative game of invention and communication."[5]

Because architects are responsible for overall system qualities,[2] they need—and must organize—communication with all project roles and stakeholders, even among multiple projects—for example, in product line development. In addition to an adequate organizational structure, this requires the ability to build effective communication networks.[7] For this, architects must be passionate communicators, and this isn't a typical attitude of a classically educated engineer. It requires management attention to convince everybody that communicating architectural issues is important and intended.

Architects whose work is distributed over many teams need to organize their communication. An architecture board enforces communication among architects and provides the platform for architectural decisions of general concern. This is especially important if the teams are distributed over multiple continents and time zones. Figure 3 shows the communication the architecture board supports.

Our experience shows that architectural communication must be constantly stimulated. It's too tempting even for architects to come up with quick solutions on their own instead of consolidating these solutions with others beforehand, even if they're directly concerned. "Discussions slow down my progress" is a frequent argument. Developers and architects feel the pressure of the project and not the future obstacles they'll create with quick-shot solutions, thus causing system qualities to become out of balance.

Traveling architects. *A telecommunications project involving 400 people worldwide began with a single scrum of architects. Later, these architects traveled around the world to support the teams with the most critical tasks at the time.*

This example shows that the more teams work for a project, the more architects that must sustain direct and personal communication to all the teams.

Achievements

Compared to former projects, we achieved seven improvements with this method:

- The amount of application code decreased because framework functionality fulfilled application needs. Before, a typical application for diagnostic imaging contained approximately 10,000 LOC for framework integration. This glue code is now reduced significantly.
- Framework refactoring increased, leading to less architectural rule breaking. For example, architects merged an important improvement for viewing layout flexibility with the frameworks, avoiding separate maintenance by developers in the applications.
- We achieved early integration of prototypes and partial product code into a skeletal system. Thus, a crucial performance requirement could be validated after six months for a five-year project.
- We achieved consistent application architecture in the project through direct involvement of the architects and the architecture board. Before, each of three groups of teams located in different continents independently developed their own approach and metaframework.
- Reuse of concepts and code increased because architects directly supported related development. In one case, we convinced negative developers to use an established batch-processing pipeline for interactive purposes by showing them that the performance was sufficient for their use cases.
- We avoided costly application design approaches by establishing communication with architects. In one case, Asian and American teams had to integrate their components, so we promoted a little framework change that let us configure them into a single process, avoiding a costly solution based on remote interfacing.

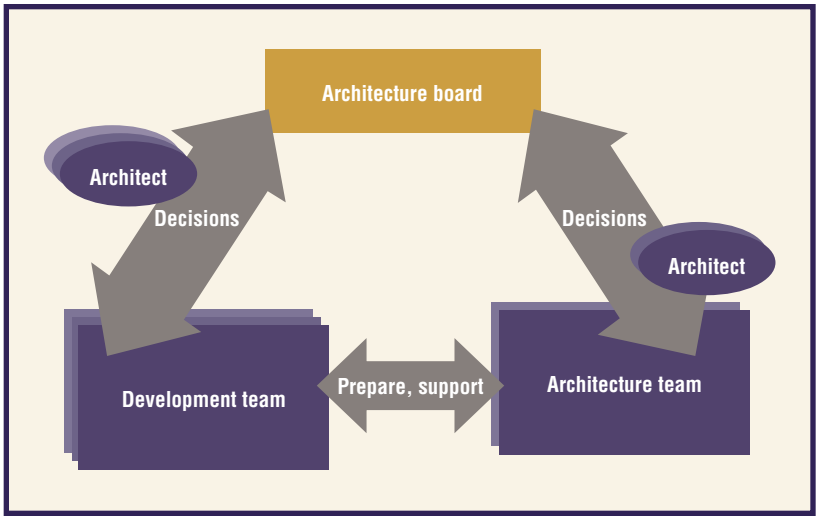

Figure 3. Communication supported by architecture board. Architects from architecture and development teams carry general architectural questions to the architecture board for decisions. The architects directly communicate these decisions back to the teams.

- We gained early knowledge about framework improvements for the next system release because of the architects' direct application development experience.

What can't be measured is the confidence our architects gained from direct contribution to the implementation work in the development teams. This confidence will let us further improve our software architecture and promote this method to other departments.

This method doesn't reference a specific technology or process and can be applied to many software architectures and development approaches. However, for a successful introduction, the architects' consent is crucial and difficult to achieve: they feel a loss of power because they no longer solely determine the system structure. Early successful support to developers strengthens the architects' position and helps gain the developers' trust for guidance. So, careful preparation is the key to success.

Introducing the method finds developers skeptical about how they'll benefit from the architects' practical support. Again, it helps if the developers recognize the architects as skilled and successful developers.

Developers fear that the architects will dictate the implementation details, reducing their prominence to that of inferior programmers. It's important to emphasize the separate responsibilities—functionality versus system qualities—

About the Author

Roland Faber is a senior software engineer in the software development department of magnetic resonance imaging (MRI) at Siemens Healthcare. He leads a team of software architects for the development of MRI applications and workflows for medical diagnostics. For 25 years he's been developing product software related to image processing, algorithms, remote service, databases, standards for digital imaging and communications in medicine, and workflow systems. Faber has a Diplom in physics—equivalent to a master's degree—from the Friedrich-Alexander University of Erlangen. Contact him at roland.faber@siemens.com.

to make it clear that developers have the same creative position as before.

It's rare to find individuals who can master technology, communication, and guidance at the same high level, but team members together can represent all the required abilities. Our team member architects support one another, and the team leader frequently promotes communication.

Most software engineering curricula don't sufficiently anticipate the importance of communication. The education of future software engineers should emphasize social skills and practical teamwork experience. At Siemens, a qualification program for senior software architects addresses this issue.[6]

References

1. J. Eckstein, *Agile Software Development in the Large*, Dorset House, 2004.
2. L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Addison-Wesley, 2003.
3. P. Kruchten, R. Capilla, and J.C. Dueñas, "The Decision View's Role in Software Architecture Practice," *IEEE Software*, vol. 26, no. 2, 2009, pp. 36–42.
4. *ISO/IEC 9126, Software Engineering Product Quality*, ISO, 2001.
5. A. Cockburn, *Agile Software Development: The Cooperative Game*, 2nd ed., Addison-Wesley, 2006.
6. O. Creighton and M. Singer, "Who Leads Our Future Leaders? On the Rising Relevance of Social Competence in Software Development," *Proc. 1st Int'l Workshop Leadership and Management in Software Architecture*, ACM Press, 2008, pp. 23–26.
7. I. Gorton, *Essential Software Architecture*, Springer, 2006.
8. C. Larman and B. Vodde, *Scaling Lean & Agile Development*, Addison-Wesley, 2009, pp. 182–184.