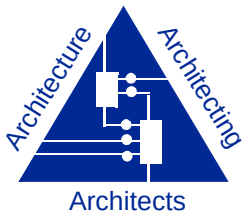

Architecture as Business Competency

Over the past decade, more and more organizations have been appointing architects to address "the software problem." And, really, architects can play a key role in freeing the organization from the legacy noose that is choking it**,** and creating strategic advantage. But the rest of the organization has a critical role to play too!

In this article, we explore the role that architecture plays in business success, and we explore what the organization must become good at, to allow architecture to play this role.

by Ruth Malan and Dana Bredemeyer
Bredemeyer Consulting
ruth_malan@bredemeyer.com
dana@bredemeyer.com

Introduction

In this article, we explore the role that software architecture plays in business success, looking at what happens in the absence of architecture. We explore how architecture addresses the problems encountered quite broadly across software development organizations, be they product development organizations or IT groups supporting the business. Since architectures are undone even more quickly than they are done, we explore what goes wrong when the whole organization is not involved in the cultural shift toward higher levels of architecture competence. We also extend our notions about the realm of the architect's decision focus and responsibility, and discuss related implications for the architect skillset. And finally, we discuss the relationship between architecture and strategy, and the critical big change that strategic business managers need to make if they want to achieve alignment between business strategy and system development.

The Problem with the Status Quo

Everywhere software development is discussed, we hear the same litany of concerns: we face unpredictability in development schedules and system behavior, and quality issues with hard to find defects, hard to fix problems, and fixes that introduce new problems. Our systems are increasingly difficult to change, making it harder to respond to market shifts. Software development costs more and takes longer. Reuse opportunities are lost because it is hard to isolate chunks to reuse, or the chunks are highly tuned to a specific product. There are morale problems and it is hard to keep good people, and hard to energize teams for the next brutal round of development. All this has the result that we are losing ground in the market, our competition can do more with less, and we are not responsive to our customers.

The whole situation is exacerbated because, in spite of this list, we are generally quite successful. Our market tolerates our schedule slips and grudgingly bears with us through our quality issues, and our organization goes through round after round of bloating growth coupled with displays of developer heroism. But with each success, our pain gets worse, until suddenly we find we are not successful, and we fail implosively!

Architecture As the Solution

At the heart of the software problem is a tangled mass of code. The solution is to break up this mass into manageable chunks. To be manageable, each chunk must obey the principles of

- encapsulation and information hiding
- high internal cohesion within the chunk, and low coupling between chunks.

Object-oriented methods went after this problem, and indeed made a significant contribution. The fact that they did not entirely solve the problem highlighted the issue of chunk size. An object, or its generalization, the class, is too fine-grained to effectively address the problem of creating "manageable" chunks in complex systems. When the number of classes rises into the hundreds, the *system* is too complex to be dealt with at the level of classes.

Software architecture, then, decomposes the system into several large-grained elements which are generally called "components" in dominant definitions of software architecture (e.g., Bass, et al., 2003). In very large systems, these architectural elements may be called subsystems, and these are further decomposed into components, which are themselves large-grained. By creating these "levelled" views of the system, it is made intellectually tractable:

- the whole system can be analyzed and understood at a high level,
- components are abstractions that hide unnecessary detail and simplify the system, and
- components can be understood in terms of their role in the larger system, but a component's details only have to be understood by those charged with designing and implementing the component.

In decomposing the system, architects generally follow the principle of separation of concerns, so that component owners can focus on a cohesive set of component responsibilities, applying and building specialist skills, and working more productively with greater independence and focus of attention. Further, this property of high cohesion and low coupling ensures that the impact of changing requirements is better contained or localized, and impacted components can be more effectively traced.

Interfaces define the component's access points, determine its externally visible services and properties, and enable component plug-and-play. They serve as a contract between component providers and clients. A well-defined interface makes it easy to identify and understand the purpose and behavior of a component, and how to use it. Interfaces serve to increase developer productivity, allowing them to focus on assembling and linking proven components via their interfaces.

Thus, architecture also helps bring the project management problem under control. Components provide better work partitioning with decreased, or at least or more manageable, dependencies. We escape the problems that come with tightly coupled systems, where it is hard to isolate problems and their fixes, and there is a ripple effect that cascades from quality problems to predictability problems, to schedule pressure and morale problems. Or do we?

The Problem with the Status Quo—Again

The rate of innovation, already astounding, is ever accelerating, and competing products are proliferating apace, even while revolutionary new products reshape the order of competition. Add to this ethical, political and economic upheaval, and the business world is characterized by turbulent change in every sphere—in the competitive landscape, the supply chain and more broadly the value network, technologies, regulations, the national and global economy, and so on.

Typically, we respond to the shifts in the environment with quick-fix adaptations to the socio-technical systems impacted. In enterprise systems, these accommodations to the architecture give the immediate appearance of responsiveness, but they increase inconsistencies, divergence and idiosyncrasies that make the systems very hard to integrate or leverage, ultimately ossifying the organization along the lines of its increasingly rigid systems.

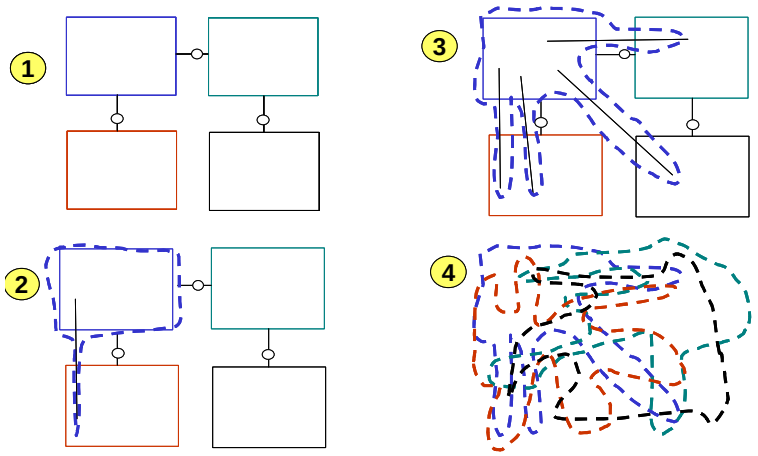

Figure 1: Illustrating how architectures are undone. Each decision to bypass interfaces creates coupling. The system quickly devolves into a tangled mass of code.

In software systems, these accommodations to the architecture amplify the (mis)conception that software is infinitely pliable, but they increase coupling and erode the architecture. The accommodations are like rebar in concrete—the coupling turns the system into a rigid, monolithic unit. With each shift in the environment, and each urgent and well-intentioned, even rationalized, accommodation to the architecture, the architecture degenerates. To make matters worse, these devolving "architectures" tend to be around a long time, and certainly much longer than we think they will be.

The Organization's Role in Architecture As the Solution

The solution has to be to start with architecture, and to *follow it through*! This is important for a small project, and indispensable for a large project. For projects of any complexity (due to the number of people

involved, difficulty of the problem, broad scope, etc.), architecture presents the only way to address significant issues. But it is insufficient to create a "box and line" drawing showing the system decomposition, although this is a step along the way to partitioning the problem and making it comprehensible and manageable. It is even insufficient to create well-crafted and documented interfaces, although this is a step along the way to making the architecture precise and actionable. A great architecture is simply a bundle of paper until it is implemented. Along the way, all manner of intentional and unintentional "accommodations" to the architecture are likely to compromise its structural integrity and render critical architectural mechanisms unrecognizable—unless the organization accepts and understands the architecture and the costs of subverting it.

The Kludge is the Antithesis of Architecture

We have to work on changing the organizational culture that allows, even encourages, quickie compromises to the architecture. Our business culture typically encourages a focus on results today. It would be hard to get away from this orientation, and one would even question whether that would be the right approach. But the short-term view creates a cultural hotbed fostering the kludge approach. Scrambling to be quick can be the quickest route to slowing the project down. With each accommodation that increases coupling, integration and quality issues rise and the current project immediately starts to bear the consequence with unpredictability and delays. Yes, the kludge, the antithesis of architecture, is the root cause of the software mess!

We need to move away from the notion that the kludge approach is to be sanctioned. Don't fool yourself thinking that we'll "re-architect" the system before coding begins on the next release. That rarely happens. Moreover, it is not just a matter of creating a nightmare tangle for the next release, but it is matter of getting *this* project done on time, on budget and within spec.

This means changing our value system and our precepts from the ground up. Developers need to abhor the kludge. Project managers must not create sanctuary for kludge-makers. And business managers, when they tell the development community to "do what it takes" must be careful not to reinforce the notion that "the architecture is up for grabs."

Goodwill is the Real Silver Bullet

The other historical legacy that we have to overcome is the tendency to "throw darts" at the architecture. Everyone has a unique vantage point, and from that perspective, is going to see problems with the architecture. From their vantage point, they are, generally, absolutely right. Indeed, that is why we architect! If the the decision could adequately be made by someone with limited, local scope of decision control, it should not be part of the architecture. It is precisely when optimizing at local scope will suboptimize the overall solution that we need the system architects to make the call. We each need to recognize that best for the whole necessarily entails compromise for some of the parts, and fighting to optimize my part is only going to cause the overall solution to be less than optimal.

Even if we just cannot bring ourselves to buy-off on the architectural approach, we need to remember that "good enough" is better than "none at all" when it comes to architecture. And that is worst case—if we do give the architects the benefit of the doubt, we will free them up to create a superb solution! What is required is a healthy dose of goodwill. Indeed, we like to emphasize that the real silver bullet that will slay the proverbial software dragon is goodwill. Just as surely as we need good *leaders*, we need good *followers*. And we need people who will gracefully *get out of the way* when they have no other contribution to make.

The Problem with the Solution

So, we need to create a system architecture and ensure that it is carefully evolved rather than making haphazard accommodations in response to whim, misunderstanding or even real changes in the environment. But, given that we need to decompose the system to address "the software problem," a new set of problems

emerges. Once the system is decomposed, what we get is a set of pieces. The issue then is, do the pieces *fit* together? This is only partly a matter of interface or "connectability" between the pieces. Good fit—that is fit that maintains system integrity—also has to do with whether the system, when composed of the pieces, has the right properties[1].

The Architect's Role in Architecture As the Solution

The responsibility of the architect, then, does not simply entail decomposing the system into components. The architect (or architecture team) is responsible for all technical decisions that have broad scope and high impact across the system, and specifically for addressing system properties.

Cross-Cutting Concerns

We refer to broad-scoped qualities or properties of the system as *cross-cutting concerns*, because their impact is diffuse or systemic. It may be a matter of preferring not to isolate these concerns because the decomposition is being driven by other concerns, or it may be that no matter how you might "slice-and-dice" the system, multiple parts are going to have to collaborate to address these cross-cutting concerns. At any rate, to effectively address cross-cutting concerns, they must be approached first at a more broad-scoped level. Many system qualities (captured in service-level agreements or non-functional requirements) are of this nature. They include performance, security and interoperability requirements. To make the picture more complicated, the system qualities may conflict, so that trade-offs have to be made among alternative solutions, taking into account the relative priorities of the system qualities.

Who Should Address Cross-Cutting Concerns

Surely components are where we take care of the non-functional requirements—the properties—of the system? This thinking has a beguiling logic, since the only coding that happens, happens within components, so ultimately, yes, properties are made or broken at the level of the component. But NO! We have to address broad-scoped, systemic properties at the level of the system—first. At the level of the component, our decision-scope is limited and our view is, I'm afraid, too parochial, to address the tradeoffs that must be made in order to achieve system-level goals. Yes, sometimes the way we address a cross-cutting concern allows us to localize requirements and allocate them to components via their interface contracts that specify their externally visible properties. Thus we may specify an upper bound on resource usage or computation time, for example. But often architectural mechanisms must be specifically designed to deliver the property. This is true, for example, of security and interoperability. These mechanisms may specify structural elements (e.g., interfaces) of components involved, as well as behaviors. And sometimes the best we can do is direct attention, for example, through architectural principles that guide design and implementation decisions so that they are in alignment with the architectural intent and maintain system integrity.

When a System is Less Than the Sum of Its Parts

A system is only greater than the sum of its parts, if the parts, when assembled together, yield a system that works as needed! This cannot be taken for granted. Parts that are designed for one system context will not have the same properties as parts designed for a different system. The situation is even more indeterminate when parts are developed without any system context. If you plug them together, you will not be able to make any guarantees about the system properties; instead these will emerge from the combination of parts. It is the architect's responsibility to design a system that when decomposed into components, can be recomposed into a system that has well-behaved properties that satisfy stakeholders' prioritized system capability goals, where capabilities are defined in terms of system functionality and system properties.

1. In seminars, Russell Ackoff puts this challenge to his audience (we have paraphrased, based on memory): Collect together a team of the best automotive design engineers in the world. Assign them the task of selecting the best car component of each type. Will they be able to create the world's best car from these components? No, of course not!

If architecture is taken to be merely a "recommended approach" which developers can take or leave as they see fit, we cannot expect to reap the benefits of architecture. We discussed the coupling issue that arises when the architecture is accommodated. But, as we have just seen, architecture is not just about decomposition and reduced coupling. In particular, if the architecture is applied in a haphazard way, system properties will be emergent from the assemblage of system parts, and system acceptance will be a matter of compromise on the part of stakeholders. In contrast, when the architecture is systematically created, applied, and evolved, then the system properties will be the result of explicit and reasoned tradeoffs and negotiation. Architecture is, in essence, about gaining control of system properties.

Architects Make Decisions, Not Just Recommendations

To achieve this control, architects must own decisions that have systemic impact. Moreover, architects must own decisions about which decisions they own, and which to delegate to component owners and developers. That is, it cannot be up to developers and business analysts to decide what parts of the architecture to apply and what parts to ignore. Put another way, if you treat your architects as consultants to the development team, making recommendations but not decisions, do not expect to get a system that is necessarily greater than the sum of its parts. Rather, all you can expect to get is an assemblage of parts with unpredictable properties.

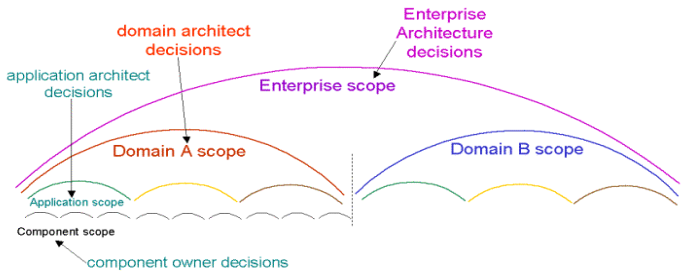

Figure 2: Illustrating architect decision scope. For example, if a decision could be made at the component level without adverse impact at the application level, then it should be made at the component level.

Minimalist Architecture

At the same time, architects must apply the Minimalist Architecture Principle: Keep your architecture decision set as small as it possibly can be, while still meeting your architectural objectives (Malan and Bredemeyer, 2002). Two guidelines can be applied to achieve a minimalist architecture:

- only address high-priority architecturally significant requirements. A requirement is architecturally significant if it cannot be met if it dealt with at a lower level (meaning more narrow scope) than architecture. If a decision could be made at a more narrow scope, defer it to the person or team who is responsible for that scope.
- evaluate each decision from the point of view of its impact on the overall ability of the organization to adopt the architecture. A decision may address a highly critical concern, but if it would cause the architecture effort to be derailed, it should be put aside for now.

Architects are highly valuable, essential technical assets of any company, and their attention should not be squandered on decisions that are not, truly, architectural. Similarly, designers and implementers are also part of the critical capacity to produce innovation and value, and their ability to do this should not be unnecessarily restricted but rather channeled appropriately to fulfill the architectural vision and the business strategy it implements. The only justifiable reason for restricting the intellectual freedom of designers and implementers is demonstrable contribution to strategic goals and systemic properties that otherwise could not be achieved.

Architecture Documents: Necessary but Insufficient

Architecture, then, embodies a set of decisions about the system's overall structure and addresses cross-cutting concerns. These need to be conveyed in a way that enables developers to build a system that embodies the required qualities and maintains system integrity. An architecture model documents a set of

decisions, but does not capture the assumptions or rationale behind them, nor explain the impact and ramifications of these decisions. Models are essential, but they cannot stand alone. Documentation, showing traceability to requirements, explaining why a decision was made and how the architecture addresses concerns, provides a critical record. But, though a good writer can help transfer insight and some of the thinking that goes into architectural choices, we have to be real about the amount of time developers are going to spend reading before they do the visible coding work that they are rewarded for. Yet developers need to understand the architecture, as it relates to them, and how to apply it.

Architecting Doesn't End When Construction Begins

Rather than protest that it is the responsibility of developers to read the architecture documents, architects need to play a proactive role. Workshops designed around exercises applying the architecture are great at getting participants' minds and hands engaged. It also takes ongoing consulting and constant educating and influencing through the construction phase of the project to ensure the architecture is understood and adhered to, and to take corrective action when ambiguities or misunderstandings surface. The architect's role at this point is guide and coach, generally setting direction but drawing a firm line when needed.

Further, the architect needs to be attentive to any emerging decisions that require architecture intervention or assistance. The architect also serves as a resource to the developers—a sounding board, someone to assist with problem-solving, a mentor. The more effective and credible the architect, the more likely it will be that developers seek the participation of the architect when they encounter challenges that might be architectural (broad scope, high impact, or affecting system integrity). This provides the architect with much-needed opportunities to stay current with the challenges emerging in the domain, while providing developers the benefit of the architect's talent and experience and deep insight into the architecture.

The Problem with the Status Quo—Yet Again

With our ears to the ground we hear about "the software problem" that we have been discussing, but as we go up the management ladder we hear about the "need for alignment between strategy and IT," or put more strongly, the "failure of IT to support the business strategy." Yet, it is surprising how many organizations have "no business strategy," according to their architects. This means that either they really do not have one, or the strategy process and its results are being so well hidden that even the architects do not know about it. This state of affairs is barely better in organizations that do have a clearly communicated strategy, for it is thrown over the wall to the architects. There is a broadly established belief that strategy is the domain of "the business" and that the management hierarchy forms the appropriate communication chain for strategy.

In today's decentralized organizations, with high value being (appropriately) placed on individual creativity and contribution, everyone makes strategic decisions as part of their day-to-day job. Taken together, the disconnect between strategy and technical leadership, and the culture of autonomous technical decision-making, means that the real strategy of the business is emergent from all the myriad technical decisions made by developers. Because these decisions are being made in a strategy vacuum, where the emergent strategy will take the organization is a matter of luck and heroics, not a matter of business leadership.

Strategic Management's Role in Architecture As the Solution

We reap the seeds we sow. If we want a new crop, we must sew different seeds! If we want strategy to drive the business, then the business strategy must be explicitly stated and shared with architects who will translate it into a technical strategy (Malan and Bredemeyer, 2003) that will be the foundation for this generation of business strategy, and evolve to be the foundation for the coming generations of business strat-

egy. This technical strategy shapes the architecture, and ultimately the products that use the architecture—now, and for the next 5 to 10 years!

The business strategy is intended to direct what organizational capabilities must be sustained and built, but the technical strategy determines what the product capabilities are, and determines what capabilities are even feasible. If there is a disconnect between the business strategy and the technical strategy, the product capabilities will not necessarily match what the business strategy intends.

Technical strategy, and its expression in the architecture, is what provides the context for all the individual technical decisions to be aligned with the vision and strategic direction. It is what provides focus and leverage, and concentration of energy, powering the technology capability essential to execution of the business strategy.

Data Does Not Transmit Insight and Energy

Architects need to be brought into the strategy process. They need to inform business strategy of opportunities and threats only technologists see, and they need to translate business strategy into technical strategy so that the technical platform of the business will support the business strategy. Remember, data (like a list of strategic objectives) does not transmit insight and energy. If you want your architects to passionately pursue the strategy, include your most talented, technically-strategic thinkers in the strategy process. The business strategy will be better for their involvement, and this inclusion will imbue them with insight into the analysis and decision making that goes into the strategy, and commitment to the chosen direction. They can then take this insight and energy to their team of architects. This enables a top-down roll-out of strategy in terms that are most meaningful to the technical community.

Top Management's Role

Top management, then, has a critical role to play in changing the top-to-bottom culture of the organization to make it an environment where an architecture-orientation will thrive. This starts with a visible commitment to inclusion of technology leadership in the strategy-setting process, bringing the chief architect explicitly into the corporate strategy process, and championing the role of solution or portfolio architects in the business unit strategy process, and product architects in the product strategy process. It also entails fully committing to architecture, finding and nurturing the talent, and supplying the tools and other resources throughout the architecting process. And remember, architecture is not something that is done upfront during the project lifecycle and then put on a shelf while the architects are shunted into a coding hotspot. Architecture requires constant investment, and the architect role must be full-time and ongoing.

Strategic Gain

What you gain from this investment, is a unique point of leverage, given the architecture maturity of the majority of companies. Good architects are relatively rare: they are technically experienced, effective at addressing interacting, cross-cutting concerns and structural stresses, and at the same time they are organizationally adept leaders and strategy setters. An organizational culture that fosters good following is relatively rare too. Put them together and you magnify the skills of the uniquely talented few at the apex of your organization's system-design prowess, and stand to build a strategic advantage that is hard to match.

Conclusion

Architecture allows us to be more creative, more productive, and *more driven by our strategy.* But in order for architecture to make this kind of difference, we have to do things differently. It is not enough to create an architect job title and promote the group's leading technologists to this role. The competencies required of an architect include technical expertise, as well as leadership, strategy, consulting and organizational politics (Bredemeyer and Malan, 2001). As talented as this group of architects may be, and as much as the organization might invest in building their skills in all of these domains of competency, the success of architecture is not their responsibility alone.

The entire organization has a role to play. In the development community, project managers and developers need to change the current mindset where, in the name of speed, “anything goes.” The architecture has to be understood and adhered to, and this involves a partnership between architects and the development community where the primary ingredient for success is goodwill.

Further, to reap the ultimate benefit of architecture—that being more effective strategy execution—management needs to recognize that architects form a critical bridge between strategy and implementation. To be most effective, key architects should be included in the strategy-setting process, but at a minimum they need to be informed of its outcome.

References

Bass, Len, Paul Clements and Rick Kazman, *Software Architecture in Practice*, Addison-Wesley, 2003.

Bredemeyer, Dana and Ruth Malan, “The Role of the Software Architect,” published on the *Resources for Software Architects* web site at http://www.bredemeyer.com/papers.htm

Malan, Ruth, and Dana Bredemeyer, “Less is More with Minimalist Architecture”, IEEE's *IT Professional*, September/October 2002.

Malan, Ruth, and Dana Bredemeyer, “Architecture Strategy”, published on the *Resources for Software Architects* web site at http://www.bredemeyer.com/ArchitectingProcess/ArchitectureStrategy.htm

Acknowledgments

We would like to thank all the architects that we have worked with over the past decade for freely sharing their insights and hard-won lessons with us. We are most privileged to be able to pass on their learnings through our papers and workshops. We would also specifically like to recognize Mark Simos, who coined the term “accommodations.”

Restrictions on Use

About Bredemeyer Consulting

Bredemeyer Consulting provides a range of consulting and training services focused on Enterprise, System and Software Architecture. We provide training and mentoring for architects, and typically work with architecture teams, helping to accelerate their creation or renovation of an architecture. We also work with strategic management, providing consulting focused on developing architectural strategy and organizational competency in architecture.

We manage the ***Resources for Software Architects*** web site (see http://www.bredemeyer.com). This highly acclaimed site organizes a variety of resources that will help you in your role as architect or architecture program manager. A number of Bredemeyer Consulting's *Action Guides*, presentations and white papers are on the Papers and Downloads page (http://www.bredemeyer.com/papers.htm). You may also be interested in our *Software Architecture* and *Enterprise Architecture* Workshops, as well as our *Architectural Leadership* class. For more information, please see http://www.bredemeyer.com/training.htm.

We are writing a book titled *Software Architecture Action Guide*. It is short and oriented to guiding action. It has two parts, with the first part providing context and a guide to the process. The second part is the full set of Action Guides, one for each discrete technique, model or template that is used in the Visual Architecting Process. For a preview of our Action Guides, please look at examples under Downloads on the *Papers and Downloads* page of our web site at http://www.bredemeyer.com/papers.htm.

BREDEMEYER CONSULTING
Bloomington, IN 47401
Tel: 1-812-335-1653
http://www.bredemeyer.com