The Role of the Software Architect: Caring and Communicating

Summary: The role of the software architect is a subject of much debate. In this article, I will discuss the qualities and personal traits of the ideal software architect. I will discuss why it is critical that the architect be a caring person who values interpersonal communication above all else.

This article describes the role of the software architect from the perspective of the ideal nature of a successful architect. My exploration touches the octopus/chameleon/parrot/bull-nature that is the heart and soul of the architect. We need to recognize, as software architects ourselves, that what we do is aimed at changing the lives of many different groups of people.

Table of Contents

Architecture is user experience
Architecture is negotiating and bargaining
Architecture is technology
Architecture is form
Architecture is communication
Architecture is artful
Architecture is agile
Architecture is the creation of a better world
About the author

Architecture is user experience

Let's start from the top: What the user of any piece of software sees, feels and experiences when touching the application. The user is there to do a job that needs to get done. Using well-designed, easy-to-use software is enjoyable and simplifies the user's life. The opposite experience, to paraphrase a Star Wars character, leads to frustration, frustration leads to pain, and pain leads to suffering.

The user wants to efficiently and preferably enjoyably use the software. A software architect spends his life realizing this need! The GUI – the attractive and highly usable interface between application and software that makes using the application so worthwhile – depends heavily on the architecture. This is absolutely critical. Whatever capabilities the architecture has underneath it will impact the GUI. The opposite is especially true: if you have great UX people but a lousy architect, you're in trouble. I recently heard that Kent Beck, the creator of Extreme Programming methodology, said, "you can't hide a bad architecture with a good GUI," and I believe this to be absolutely true. There are many things you need to know if you want to create a great user experience. One of the architect's best friends is the UX expert/interaction designer, but the more the architect understands these needs, the better.

In the comparison made in *Architecture as Metaphor*, 1999, the written account of a workshop held at OT ´99, the goal was "to investigate how the software community perceives the similarities and differences between software architects and building architects." One of the most interesting findings was that it is perceived that we software architects are less accountable for our results than are building architects. This is likely due to the fact that many of our results are less visible to the naked eye than the results of building a physical structure. It's sad that this is the case, because I believe that we would get much better use of our creations if we cared more about the end product of our efforts. Perhaps this will make us more accountable? All the better!

Architecture is negotiation and bargaining

Somewhere in every project there is a client. This is where the project usually starts. In order to focus on the client, let's say that it's just one person. To bring the client in on this we must move slightly back in time to before the user was trying to get something done. In our scenario, we must move to the inception of the application: Someone realized that there was a need for some system. Someone had an idea. Whoever that person was, they likely had a vision of the kind of software they needed. This is both a blessing and a curse in software projects. In many projects, I have at times felt it would be nice if there were no clients. Since this is never the case, it's good that I love a challenge. It is very inspiring for an architect to

capture the client's vision and process it. The curse part becomes obvious when you realize that the client simply does not understand the needs of the end user for whom the software is being built. More often than not, the client is not even an end user.

This goes back to the user, I realize, but the two are closely related. If the client is not the end user, an architect must get in touch with the intended end user. The architect can try to imagine the software requirements of the end user, but obviously this is quite difficult. Sadly, this is how many user interfaces are created. Many UIs are created by technicians, who I think we can all agree make very poor UI experts. The architect must be in contact with real end users, either in person or through the proxy of a requirements gathering function.

Two things need to be said here. First, there are several other project roles, including the business analyst function, that can handle much of the work of requirements gathering. Of course, this depends on the size of your project and the lay of the land within your organization. The architect might not have to be personally involved in requirements gathering, but good and sound architecture must be based on the real needs of the end users of the system. No matter how the requirements are gathered, the architect always needs to digest what the end user truly needs.

Second, with regard to client requirements or end user requirements or both, there is a problem with requirements that needs to be bridged. End users don't understand technology and developers don't understand people! The not understanding technology part is often true for clients, too. Some clients understand technology to some extent, but most of them do not, and they certainly do not want to be bothered by highly technical details.

While most clients and end users want to stay far away from technology, it could be argued that the architect should stay one layer above the core technical decisions of the system development process. This is only partly true. The architect must transform requirements into something that will be developed, using hard-core techie stuff, by a team of skilled developers. Thus, the architect needs to have at least a bird's-eye view of the technology that will make up the solution. I'll get back to this topic later in this paper. For now, let's stay focused on the user.

How do we get from perceived user requirements to real requirements? This is no small feat. Naturally, the architect needs to understand the capabilities of the underlying technical systems. While most technical issues can be solved one way or another, the problem is the small matter of making correct architectural decisions. Equipped with an understanding of what technology is potentially in play, the architect must begin mapping available technology to the perceived requirements and start investigating where there might be a mismatch between perception and reality. The architect will, during this work, suggest changes to the perceived set of requirements. This can be done either to correct mistakes made by those who set the requirements, or it can be done to suggest changes to the design due to technological capabilities that are not envisioned by the client or even dimly fathomed by the end users.

Have you ever met a client with a 100% accurate view of what needs to be built, and an equally insightful view of the real needs of end users? There is no such client! If there were, that person would be the architect. Heaven forbid the client who has architectural ambitions! In this dreaded case, you would not be asked to fill this role. In either case, there is usually quite a bit of negotiation among those who claim a requirement or request a feature and the architect who is bringing technical knowledge to the table.

In the end, a client envisions some software system or product and a set of end users, and has a list of needs to be fulfilled. The architect is instrumental in realizing this set of requests and needs. And so my initial claim holds true: An architect changes people's lives. As Spider-Man says, "With great power comes great responsibility." The architect must really care about the client and the end users, and what these people need, in order to realize the best and most efficient software system. The architect must be an investigator who inspects and adapts, but even more importantly, the architect must be a caring individual.

Architecture is technology

In discussing project requirements, we have naturally come to the issue of what is possible to build. Some of us feel that almost anything is possible in technology. I'm sure that's what they said 40 years ago, too! It is critical, however, that a software architect be a good technician. Of course, this is always a balancing act, since there is always going to be much more technology to be skilled in than there is time to acquire during a lifetime. No architect can be an expert in all areas of technology. In fact, I will argue that the better the technician, the less skill

a person likely has in interpersonal communications. I will not delve into this question too deeply here because it is not essential to this article. Of course, there are some really good architects who are also really good technicians.

Now that we have separated the architect from the implementation-level details, let's reverse the perspective. If an architect does not take part in the actual implementation of the project, we run the risk of having all of the architect's beautiful visions making a poor match with the technology being used. It is crucial for the success of project implementation that the architect take time to compare implementation details against the vision. *Organizational Patterns of Agile Software Development* [Coplien & Harrison, 2005], a great book on agile software development, states a pattern for this phenomenon: "Beyond advising and communicating with Developers, Architects should also participate in implementation." This is not the same as saying that the architect should do a lot of coding, but it certainly helps if the architect gains a full understanding of the art of putting the pieces together. An architect who does some coding will gain valuable insights, and this in turn will aid in fitting requirements to technology.

Or is it the other way around? Fit the technology to the requirements? A great way of doing this effectively is to pair up the architect with an experienced developer. This should be done from time to time to get a feel for the experience of coding the architect's design. An additional benefit of this pattern is that developers perceive a buy-in from the architect. Also, there is an improvement in the overall soundness of the fit between framework-level concerns and the application being built. The supporting framework will do just that – support the application better – when the fit with the application is as frictionless as possible. This is more easily achieved if the architect also implements code, because then the architect knows what it is like to use the design.

One final note is that the architect is labeled the keeper of the flame, meaning that the architect owns the principles of the project. The architect is a role model for the entire development organization. This is not a small burden.

Let me now turn to the issue of specialization. True mastery comes from excellence in one or a few areas, combined with a good understanding of many related fields. It is the architect's job to know a little about a lot of relevant technologies. The architect must successfully map the complex requirements from the client and the end users onto a set of technologies suitable for the design challenges at hand. The artifacts coming in from the user side might be a set of drawings, user stories, or something of that sort. These will be transformed into language that must be meaningful to a completely different set of people, namely developers. This must be done to enable developers to work as efficiently and joyfully on the development of the system as humanly possible. Did I say joyfully? Did I say humanly? Yes. A happy developer is a productive developer! And what makes developers really happy? I'll tell you: Working with great technology, creating great applications out of a set of well-laid plans. If the architect listens only to the needs of the end users and not to the developers, the developers will be unhappy and unmotivated. The architect must make the developers excited about the requirements. This is absolutely paramount to the success of the project. The two sets of people, end users (and clients) and developers, really and truly do not understand each other. To ensure success, the architect must understand and satisfy both groups. This work is never complete, and spans the entire project.

It is tempting to label the architect a spineless coward, bending to every whim of clients, end users and developers. This is far from true! After all, it is the architect who, in the end, will drive development forward by bridging the gap between the users and the developers. It is the architect who holds the Rosetta Stone for translating among groups with very different needs. As noted earlier, the architect must inform the client and the end users of the technological possibilities and limitations, set realistic expectations, modify expectations, and even create new expectations. This is scary business and not for the faint of heart.

This means that the architect is both a technician and a humanist, and wields the power of translation from geek to speech, and back again. Simultaneously!

Architecture is form

It is important to explore what architecture is all about. It is easy to think that the goal of creating architecture is to build an intricate yet adequate diagram of objects and classes with relations among each other. It is tempting to think that the completeness of the architecture is somehow measured in terms of how the architectural artifacts of the system look. But this is the naive view of architecture, and it is merely a confusion of structure over form. Structure is classes and code patterns and so on. However, to presume that this *is* the architecture could not

be further from the truth. Architecture is about form: If you do not know the form of a system before you begin building it, you are in over your head. Many Object Oriented (OO) practitioners tend to confuse the one with the other, because it is the structure of a system that we can look at, and teach and learn about, the relations among objects and framework components and such. This is what is tangible.

This in no way implies that structure is unimportant. It just tells us that often when we speak about structure we are actually referring to form. Structure is a critical part of what we do as architects. What layering model of the system do we need? How do the layers define? What code patterns should we use? What techniques are appropriate in the application? To be fair, are these not exactly the questions one would ask oneself while figuring out the structure of a software system? Your previous experiences constructing other software with similar needs would guide you to use techniques that have worked for you before, or perhaps new techniques that you have recently learned but not have yet used. In recent years we have seen the introduction of new concepts into our trade, including object factories, dependency injection and modularity. Not long ago this arcane knowledge was known only to the mythical pattern experts. Yet now these are just tricks of the trade, if you will.

The real purpose of an architect is that of a craftsman and artist, as the building architect Christopher Alexander writes in *The Mary Rose Museum*: "Our experience as contractors, engineers and architects during the last 15 years has proved one thing over and over again: The things placed on drawings are inevitably – always – wrong in many particulars. Drawings serve as an important rough sketch of something that will be built, but must be executed with constant attention to room shape, light, wall and ceiling detail, openings – above all to the feelings which arise in each place, in the construction, as it is taking shape. These feelings are too complicated to predict and cannot be predicted. When a building is built from plans that are conceived on the drawing board and then simply built, the result is sterile at best – silly most of the time – and sometimes unthinkably bad. This is something familiar in virtually all large buildings that have been built since 1950. It is inevitable, given the process of construction used to build them. And it is inevitable that this process must lead to unsatisfactory results." [Christopher Alexander, Gary Black and Miyoko Tsutsui, 1995] An architect who builds a house does not and cannot predict the location of each wire and pipe, because they must fit through the stones, bricks or studs in the walls. That, in turn, can change the location of sinks. You don't know these things until the last minute.

While it is true that the architect has many masters, it is equally true that the architect must work very much like Cardinal Richelieu, the Red Eminence, and with a firm hand guide developers to realize a beautiful vision. The architect must be a stickler for detail and an enforcer of high standards, while always having the guts to make critical changes to a design when necessary. An architect has to have guts paired with a good sense of the form of what is to be built.

Architecture is communication

According to an article by Alstair Cockburn [*The end of software engineering and the start of economic-cooperative gaming, 2004]*, "At its core, software development is people inventing and communicating, solving a problem that they do not fully understand, which keeps changing underneath them, creating a solution that they do not fully understand, which also keeps changing underneath them, and expressing their thoughts in artificial and limited languages that they do not fully understand and that also keep changing underneath them, to an interpreter that is unforgiving of error, where every choice has economic consequences, and resources are limited. Software development is a resource-limited, goal-directed, cooperative game, whose moves consist of invention and communication."

Cockburn deliberately does not indicate which people are solving, creating and expressing, or "gaming" if you will. The way I see it, the architect is at the very heart of this activity. In other words, the architect is a gambler and an explorer.

If you are an architect, you must have a vision for the system, and the series of discussions that lead to the architectural structure of the solution, as being created throughout the life of the project. Creating software architecture is not a process we may plan ahead of time, as with building architecture. We may employ our skills and experiences to guide the process with some accuracy. The effectiveness of quickly finding proven solutions from similar situations in the past, or of actually feeling in your gut what the right solution can be, is based in experience. From these cooperative gaming activities, the architecture is the living artifact. It stems from the commitment to a project over the life of the development, and it is carried through by means of different forms of communication.

In another article [*The declaration of interdependence for modern management, 2004]*, Cockburn writes that we should "Focus on the value that is being created and watch the flow of increase in value. Each person or work station contributes a value, and there is a flow involved. In software development, the flow is often around decisions: someone makes a visioning decision, a requirements decision, an architecture / UI / domain model / testing / configuration management decision, and those decisions affect other people in a patterns. That I call the 'flow'." This means that software engineering is not really an engineering practice at all! In an engineering practice, you can copy a previous construction and re-create it over and over again. In creating software, it has been found from 40 years of collective experience that you cannot copy another team's efforts and re-create their exact results. The act of developing software does not work that way! Building software is creative at heart – artistic! Software architecture, being an essential part of this process, is therefore artistic, and it is the glue that holds the artifacts of an artistic process together.

The architect can never sit behind closed doors and expect to succeed. The architect must be out there every day, communicating with stakeholders, in order to perform the job function with excellence. Wait, isn't this the project manager that I am writing about now? No. The day-to-day tasks of the person who is doing the architecture quite often cross over into the realm of the manager. Also, I am a firm believer that each person on any team has to do everything possible to further the project. If that means the lines between roles become fuzzy, then so be it. The exact line is hard to define and is based on the needs of the organization. There are powerful and classical organizational patterns at work here that govern the needs of each organization. It is all but impossible for a good architect, one who is inclined toward interpersonal communication, not to care about the needs of his organization when they are so closely related to the architectural endeavor. Sometimes the architect and the project manager are one and the same person, which means of course that the architect does officially have managerial tasks. Even when the manager is not the architect, the architect still cares about the state of the organization and works continuously to improve the quality of work in the organization by enabling the team(s) to work at full efficiency. The architect constructs the architecture of the solution in a manner that enables developers to use their tools effectively. The architect enforces high standards and makes sure that work on the project is as enjoyable, straightforward and efficient as possible. Thus, it is impossible to be an architect without also at times being a talented psychologist. You have to feel how the winds blow in an organization, especially if the winds are not just gentle breezes. It is a fundamental mistake not to listen to the organization when creating architecture.

Architecture is artful

What is it we want to communicate through software architecture? We want to convey a beautiful vision! What we build is something that users and observers of the system will likely spend some time with. Doesn't this mean it has to be beautiful as well as useful?

Many great architects came before us, including the Athenian Mnesikles who created the Periclean gateway to the Athenian Acropolis, and also the unknown designer of the Roman Coliseum. They did not have computers, as we do today, but they did understand beauty and power.

Wait a minute! We were discussing software architecture, not building architecture! Please humor me; there is a strong connection between the two.

Christopher Alexander wrote in *The Timeless Way of Building* that "There is one timeless way of building. It is a thousand years old, and the same today as it has ever been. The great traditional buildings of the past, the villages and tents and temples in which man feels at home, have always been made by people who were very close to the center of this way. It is not possible to make great buildings, or great towns, beautiful places, places where you feel yourself, places where you feel alive, except by following this way. And, as you will see, this way will lead anyone who looks for it to buildings which are themselves as ancient in their form, as the trees and hills, and as our faces are." [Alexander, 1979]

Alexander hands this task to those who today work with computers. In a presentation he gave at OOPSLA, *The Origins of Pattern Theory the Future of the Theory, And The Generation Of A Living World*, Alexander stated that he had, ". . . . an extraordinary vision of the future, in which computers play a fundamental role in making the world – and above all the built structure of the world – alive, humane, ecologically profound, and with a deep living structure . . . . I hope that all of you, as members of a great profession of the future, will decide to help me, and to help yourselves, by taking part in this enormous world-wide effort. I do think you are capable of it. And I do not think any other professional body has quite the ability, or the natural opportunity for influence, to do this job as it must be done." [Alexander, 1996]

Alexander's vision was all about the beauty and truth of the world, and he passed his vision of a living, breathing and beautiful world to us, the people who work with computers. Those who guide this process are the architects.

There is one very important quality of the art that is architecture, and that is the ability to see the solution ahead of time, to feel it or envision it, if you will, in all its glory and in quite rich detail. Some would mark this down to experience, but there's more to it. This is about having the skill of foresight. Before the design begins, the architect needs to take a step back in his mind and see the design in full function before him. This is done before any cardboard mock-ups or prototypes are built, and the process should be revisited at every major change point during the life of the project. The architect has to become the user and *see* himself using the system. The end user is not the final focus of the construction; the end user has a goal in mind when using the system, and the fulfillment of this goal is the real reason for which this system will be built. The better the architect is at *seeing* this beforehand, the better he or she will be at understanding what needs to be put into motion to accomplish it, and thus what the design needs to be.

Am I not now redefining the role of the architect? The software architect today typically is not involved in many of the activities I describe here. Rather, the architect is concerned with doing the delivery. A more proper name for many architects today is chief building contractor. I understand many people see it this way because our discipline was defined from a foundation in engineering. I feel we have to change this view of the architect! The architect has a good imagination and is good at painting mental pictures. This skill may be taught, but aptitude for this activity is critical. Experience builds it, too. When we have this mental picture in our heads, it is time to begin the actual design process, which does not end until the project ends.

Architecture is agile

Discussing the concept of agile software development seems to be inevitable in this paper. What do I mean by the statement architecture *is* agile? Well, with all due respect, would you like to be involved in a non-agile software development project? I did not think so! We strive to be agile in our development processes; all of our architectural structure is supposed to be agile to the nth degree, with modularity and factories all over. How can an occurrence as central to any development project as its architecture be anything less? Does it matter whether agile architecture or agile practices came first? Of course, agility in structure is associated to some degree with size of organization. Does our organization require distributed development, or are we a small core team producing our product? Do we adapt our core functionality to key customers? These are technical questions, of course. Okay, so what about changing business needs? Responding to competitor evolution? Agility is present in business and practical development both, yet I see so many solutions that are a Gordian knot, where if you lift one little class at the edge of the code base, all of the other classes and parts of the software automatically get pulled up, too. I also see many solutions that are such a complete pain to work with that you almost immediately give up! All this effort for nothing! What a waste. The exact definition of what makes architecture agile I will leave for another article.

Let's instead examine the concept of whether architecture is agile from another perspective, and slant it back toward the architect that we are discussing here. I am a certified Scrum Master who really likes the openness and agility of that process. Scrum as such does not define an architect's role. The process does not define many roles at all – it is agile in this respect. A Scrum team is free to figure out who does what, as long as the members agree that this is the most productive way to reach the sprint goal. Consequently, if there is a team member who enjoys architectural tasks and has useful experience in resolving the tasks quickly and expeditiously, and if the team agrees it is a good way of moving the sprint forward, then the team may turn to that person for resolving architectural tasks. In Scrum, as I said, the architect's role is not defined as such, and if no architect is appointed by the team, who does the architecture? Usually it is the team that makes decisions regarding the design and architecture of a solution. And in cases of scaling, several Scrum teams make decisions about the overall architecture that may span several Scrum teams. The last part is often performed in a congregation known as a Scrum of Scrums meeting. When the architect as a role is not present in a Scrum project, it means that the architectural tasks are completed in collaboration within the team and through discussion among the participants and among the teams in the Scrum of Scrums. So the architect now is not an actual person but rather an entity created inside the project. Perhaps describing the architect as some non-physical entity seems strange, but if we briefly and for argument's sake imagine such an entity, what is its core nature? It is communication!

The architect must be able to adjust the architecture to respond to changing needs, and must retrofit a solution that is applicable for multiple settings. The architecture had better suit

distributed development! I don't see it as too much architecture if you take reasonable measures based on knowledge passed at the beginning of a project. Some things you just have to know you will need as part of your solution. Architecture that is agile does not mean it is necessarily lean or piecemeal; it means being subject to continuous change, and flowing around changes in a fluid manner. The solution can flow, but only if the architect can flow.

Architecture is the creation of a better world

Some of the responsibilities mentioned above are sometimes not handled solely by the architect. Instead, they may be handled by a project manager, an interaction designer or a business analyst. That's all good and well if you have those roles in your project. Any large project would. What I am trying to say is that you, as the architect, must continuously communicate with all of these functions (and certainly several more) to get to the form of your project. Some things you just have to know about your domain in order to be a good architect. One good thing to know is who your friends are in the organization.

The architect has been described in this article as a person with a range of positive qualities. Is this the ode to the architect? No, that is not my goal. Is there really a living individual that can live up to all of these expectations? I believe there are people like this. In fact, I know a few really good candidates. It is not always easy to live up to being a positive, sharing, honest and caring person. Most of us only manage to be as good as we want to be. I don't intend it to be a requirement that the architect always live up to all of the qualities described in this article. You can be a really good architect even if you don't have a generous serving of some of the personality traits described here. What you must realize, however, is that, as an architect, you are the role model every day for many different people. You are the main line of communication that bridges the gap between user and technology. You are the keeper of the flame in your development department because the design you create will inspire good use of proven practices. As an architect, you have to get into the minds and lives of many people and get emotional about what they need. There is not a more communicative or social role than that of the architect! The architect must be part architect, part listener, part organizer, part psychologist, part visionary, part humanitarian, part friend, part leader, part stickler for detail, and always willing to make a critical change in the design while not being a pushover.

Everything that I've written above has to matter to you as an architect! You can't be a great architect if you don't honestly and truly care.

About the author

Magnus Mårtensson lives in Sweden where he has a master's degree in computer science from Lund University with minors in psychology and pedagogy. He has a burning passion for technology and for conveying his message to others. As a senior consultant with Dotway AB, Magnus works as an architect, coach and technical specialist. Some of his foci include Web technology, Web services, distributed applications and agile processes. He writes and blogs (Techie.notepad http://blog.noop.se/) often about project processes, software architecture, design principles and coding.