The Software Architect
—and the Software Architecture Team

Philippe Kruchten
Rational Software, 650 West 41st Avenue # 638, Vancouver, B.C., V5Z 2M9 Canada
pbk@rational.com

Key words: Architecture, architect, team, process

Abstract: Much has been written recently about software architecture, how to represent it, and where design fits in the software development process. In this article I will focus on the people who drive this effort: the architect or a team of architects—the software architecture team. Who are they, what special skills do they have, how do they organise themselves, and where do they fit in the project or the organisation?

1. AN ARCHITECT OR AN ARCHITECTURE TEAM

In his wonderful book *The Mythical Man-Month*, Fred Brooks wrote that a challenging project must have one architect and only one. But more recently, he agreed that "Conceptual integrity is the vital property of a software product. It can be achieved by a single builder, or a pair. But that is too slow for big products, which are built by *teams*."[1] Others concur: "The greatest architectures are the product of a single mind or, at least, of a very small, carefully structured team."[2] More precisely: "Every project should have exactly one identifiable architect, although for larger projects, the principal architect should be backed up by an architecture team of modest size."[3]

[1] Keynote address, ICSE-17, Seattle, April 1995
[2] Rechtin 1991, p. 22
[3] Booch 1996, p. 196

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35563-4_35

P. Donohoe (ed.), *Software Architecture*

We speak about a *software architecture team*, and assume that the lone architect is just a simpler case. We speak of a *team*, not just a working group or a committee; a team in the sense defined by Katzenbach and Smith in *The Wisdom of Teams*: "a small number of people with complementary skills who are committed to a common purpose, performance goals, and approach for which they hold themselves mutually accountable."[4]

2. SKILLS OF THE ARCHITECTS

Software architects must collectively have a certain number of skills: experience (both in software development and in the application domain), good communication skills, sense of leadership, they are proactive, and goal-oriented.

2.1 A broad range of experience

Software architects must have accumulated significant experience in *software development*, especially if they are to tackle ambitious projects, but at the same time, they must be (or should become) knowledgeable in the *problem domain*. The two kinds of expertise must be well balanced. Ambitious software architecture projects will not succeed without *both*.

If the architects have a good understanding of the problem domain, such as telephony, air-traffic control, or computer-aided manufacturing, but only limited experience with software development and software architecture, they will not be able to rapidly develop an architecture that can be communicated to the various development groups. Even if they do not develop the code themselves, they must master the software design method (e.g., OOD), the programming language(s), understand the development environment, the development process, because they will have to interact daily with the software designers, programmers, and database engineers. They have to understand them and be understood. Their design decisions must be acceptable by software engineers. They must make some decisions very quickly, based on experience and "gut feelings" rather than pure, thorough analysis.

> In one case the resentment against an architecture team was growing. When we were called to help, we discovered that they were excellent people with a lot of experience in their field, doing a very good job of analysis, building a very solid, object-oriented model of their domain, but carefully avoiding making any software design decisions. All questions

[4] Katzenbach 1994, p. 45

about the "how" were brushed aside as "mere implementation details" that should not pollute their architectural description. Further investigation showed that they were in fact afraid of making any technical choices because of their lack of experience with similar systems. They were also under psychological pressure from technical leaders of the various development teams who they thought were much more qualified to speak about the software itself. Therefore they had shifted their focus toward analysis, even though the rest of the software development organisation was still holding them accountable for the major architectural decisions.

When architects have a good grasp of the software development aspect but a poor knowledge of the domain, they will develop nice solutions for the wrong problems, reduce the real problems to problems they know how to solve, or impose solutions that suit software engineers but are for a user community that works, behaves and thinks completely differently. For example, air-traffic controllers are not software developers; they have another view of the usefulness of menus and windows rather than the views shared by most software engineers. Imposing Macintosh-like desktop metaphors because it proved to be good for software types may prove to be a mistake in this specific case.

If you agree that software architecture, like building architecture, is concerned with more than the nuts and bolts of the software, such as how the software is used in its context—sociological and economical i.e., looking outwards, and not merely inwards—then it becomes clearer why a software architecture team must be versed in both software development and the application domain. Architects need to anticipate changes, changes in the environment in which the system under development will be deployed, which will in turn trigger requests for change or evolution. You can only do this if you are looking at that context, that domain, not just looking at the software itself. Architects need to develop a long-term vision for the project: where do we want to be with this software in two years, five years, and ten years from now?

Software architects are curious, they keep their ears and eyes open, read technical publications, and try to constantly sharpen their skills, extending and broadening the scope of their knowledge. They develop their creative skills by looking at other fields, other domains, other disciplines, from which they can derive more analogies.

Achieving this balance of expertise in a software architecture team is hard. It is not enough to bring together a few people that are very good at software development and a few people that are good at the problem domain; they must have enough knowledge, language, and vision in common so they can communicate and produce something.

In another case, when we were creating a team to develop an architecture for an air-traffic control (ATC) system, we identified some talented software designers, and some talented ATC specialists. This was just the beginning. All the software people were sent for hands-on training on air-traffic control, going to ATC classes, then spending days sitting next to controllers in a live Area Control Centre, trying to understand the essence of their activity. Similarly, the ATC specialists were sent to courses such as Object-Oriented Design, Programming in Ada, to reach the point where there was enough common vocabulary for them to efficiently work together and leverage each other's skills.

This approach does not always work without resistance: "Why should I learn about programming, since I will never program?", "Why should I waste my time going through air-traffic controller training? I am a software designer..."

The real difficulty is when there is only one architect: that one person must therefore be knowledgeable in both software development and the domain. Finding such people on the market is not very easy. The few we know of who are like that developed their unique combination of skills in a given organisation or company.

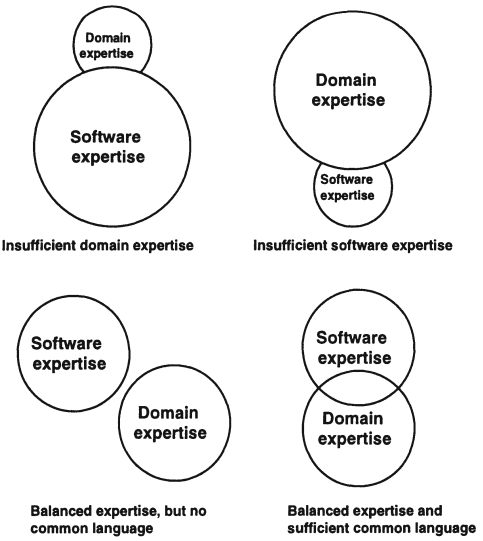

Figure 1. Looking for the balanced team

When the scope of the project is large, the problem of finding people with balanced expertise and a common language gets worse. Within the software development domain, you may need to gather enough expertise from various "specialities" such as data engineering, operating systems, networking and security expertise. Within the application domain, there may be also specialisation: in telecommunications, for example, there is voice communication, call handling, voice messaging, versus data communication and packet switching. It is not possible to find people that are experts in all specialities, but collectively, the architecture team must be reasonably aware of these specialities so they can bring in and interact with experts whenever necessary.

The issue of a common language is important. By language, we mean as well as a common spoken language, they must have a common way of representing the architecture, and a common programming language. The choice of a language is a choice of a *model*, complete with its opportunities for creativity, its internal assumptions, and its constraints. Languages, among people who speak them, provide a rapid transfer of knowledge, imply consent with the underlying connectives, and agreement on stated conclusions.[5]

The *wider* the experience the better. People who have been working with the same kind of architecture for 20 years have 20 years of experience but are likely to reproduce that same architecture for a new project. A person with only 12 years of experience with three or four different kinds of architecture brings more experience to the table. Consider getting help from external consultants: the very nature of their job ensures a wealth of experience, perhaps including work with your direct competitors.

In all cases, software architects must understand software sufficiently well to be reasonable programmers. An architect unable to express or sketch a concept in a programming language is as suspicious as a building architect who does not know how to use a T-square, a French curve, or a lettering pen; he is putting the project at risk by having a wider gap to bridge with other developers. Often I write code simply to understand what I design.

2.2 Communication skills

Communication issues grow exponentially with the size of the development organisation. Successful architects or architecture teams rapidly become a centre of technical communication in a project, and they spend a significant amount of their time interacting with one or more stakeholders: explaining the architecture to other software developers, to

[5] Rechtin 1991, p. 80

system engineers, to customers, users, prospects, marketing people, and managers. They must have therefore good written and verbal communication skills. Therefore, they must persuade, understand, dig out the real issues, convince the sceptical, and sell the architecture. As Jonathan Losk says: "Don't ever stop talking about the system."[6]

> On two different, large command and control systems, we noticed that the lead software architect was dedicating more than *half of his time* just communicating what architecture was and why it was important.

Software architects must also *listen*, listen to the project worries, to recurring difficulties with certain tools, procedures, and design choices, while they are constantly adjusting, correcting or merely explaining software architecture, again and again. In many cases, they have to *negotiate*, finding compromises that can be accepted by several stakeholders.

Unfortunately, there isn't a strong correlation between good technical skills and good communication skills.

2.3 Leadership

Architects must have some leadership skills—technical leadership, that is. This technical leadership must be based on their knowledge and their achievement, not simply on some administrative decision. We do not mean that they are the project leaders or managers, but they will lead the software development in many ways: by establishing the structure in which all the software development will be hosted, by establishing the main design rules, by ensuring that the design principles are followed, by injecting new ideas, new solutions, and new techniques into the project to improve its productivity or the quality of the product, by coaching and mentoring newcomers or more junior people.

2.4 Proactive, goal-oriented and committed

A software architecture team is not a committee, meeting every so often to share ideas or discuss issues. It is not a review board, nor a think tank for top management composed of selected staff technologists. It consists of a small number of people fully committed to a very specific goal: designing an architecture. A committee which meets two hours a week cannot design an architecture. Software architects work at this *full time*. Only in some rare cases can a member of the team split his or her time between more than one activity. In particular, we think that the function of a software architect is

[6] Jonathan Losk, cited in Rechtin 1991, p. 292

rarely compatible with that of project manager, except for very small projects (eight people or less). We will address this point in another article.

Architects must be able to sustain a high degree of uncertainty and ambiguity. Their work often consists of a long succession of suboptimal choices, made in relative obscurity, i.e., without the luxury of examining all alternatives and all ramifications of the choices. Many people with scientific training— and this is aggravated by inexperience—cannot tolerate it for very long and will tend to defer the decision-making to others.

3. THE ROLE AND PURPOSE OF THE ARCHITECTURE TEAM

The architecture team share a common goal, or small set of goals. For the team to remain focused and efficient, the goals must be clearly defined, both to the architecture team and to their environment. This imposes the need to define (for a given project) what software architecture is, what are its boundaries, and in particular, what are the responsibilities and extent of authority of the architecture team, how decisions are being delegated, how to avoid "turf conflicts," and who is accountable.

One of the best ways to establish this, especially in environments where the concept of software architecture is new, is to create and publish a charter for the architecture team. Section 3.1 is a template we have successfully used on several projects, with small variations.[7]

3.1 The charter of a software architecture team

The software architecture team is responsible for evolving and maintaining the vision of the "*name your project*" software architecture.

The main activities of the software architecture team are:

- Defining the architecture of the software
- Maintaining the architectural integrity of the software
- Assessing technical risks related to the software design
- Proposing the order and contents of the successive iterations and assisting in their planning
- Consulting services to various design, implementation, and integration teams
- Assisting marketing in future product definition

[7] This text was pinned on the wall near my office at Hughes Aircraft of Canada during most of my time as the lead software architect for the Canadian Automated Air Traffic System.

The main deliverables from the software architecture team are documents: a software architecture document, some elements of software design documents, design and programming guidelines, iteration contents, meeting and review minutes and design audits of the running system.

3.1.1 Defining the architecture

The architecture of the software is the general framework in which all software design is performed. The architecture defines the major design elements, the way they are organised, structured, the way they interact, and the way they are to be used.

To ensure that the architecture will meet the needs of the various parts of the software and the external requirements, the architecture team works in close relationship with the various "domain" development teams, and the system architecture team. The architects' view is one of breadth, whereas the domain designers' is that of depth.

3.1.2 Maintaining architectural integrity

The architecture team is responsible for the development and maintenance of design and programming guidelines. The architecture team is involved in the organisation of design and code reviews to ensure that those guidelines are being followed, or to make them evolve as necessary. It plays a major role in the organisation of end-of-iteration "post-mortem" reviews.

All changes to major interfaces and all explicit violations of a design or programming rule must be approved by the architecture team. The architecture team is the final arbiter in matters of software aesthetics.

Finally, the software architecture team is involved in "change control board" decisions to resolve problems that have an impact on the software architecture or some critical interface.

3.1.3 Assessing technical risks

The architecture teams maintain a list of perceived technical software-related risks. The team may propose exploratory studies or prototypes to investigate the feasibility of a technical solution before inserting it in the architecture.

3.1.4 Proposing contents of iterations

The architecture team proposes the technical contents and the order of successive iterations by selecting a certain number of scenarios and a certain

number of common mechanisms (services) to be studied and implemented. This technical proposal is completed and refined by the various development teams based on available personnel or customer requirements in terms of deliverables, availability of tools and COTS products, or needs of other projects. The architecture team then helps the various development teams with the transition from the architectural-level design to the more detailed design.

3.1.5 Consulting services

Because of their thorough understanding of the entire system, members of the software architecture team can provide assistance to various development teams as a floating resource for a given study, or, when needed, to help keep the project on schedule, or as "coaches" because of their specific skills or knowledge.

3.1.6 Product definition

In the context of a line-of-business of *Widgets*, the software architecture team provides some assistance in the definition of future products. It can help the marketing team with the prospective customer's requirement analysis, and during the study of the impact of a new product, shelter the development teams from too much disruption. Although it is not their main objective, the software architects play a major role in the project as facilitator or arbiter between the various product teams because of their various functions.

The software architecture team is accountable to the project manager. Its work is reviewed by the project technical staff, and a selection of senior software designers from the other product design teams. Project management can also evaluate the architects' contributions to the product by auditing their input in the final running system.

4. A TEAM AMONG OTHER TEAMS

Where do you hook an architecture team in your "organisation chart"? A software architecture team is a team of software designers and developers which should be organised no differently than any other software development team. It just happens that they are focused on different levels of abstraction or granularity, and may, on the average, be more experienced than others. But it would be a mistake to separate them, either in terms of the

reporting structure or geographically, from the other software development groups they are supposed to interact with on a daily basis.

Being on the software architecture team is not a honorific position, nor a sinecure, it is not a staff position, nor a research job. Its schedule is tied to that of the other teams, which are its main customers. It reports to the same project manager.

One way to picture this is to consider the software architecture team as playing a symmetrical role to that of an integration and test team—the architecture team precedes the development teams, scouting the terrain, drafting the design, while the integration team follows, collecting debris and the wounded. In some circumstances we have called this the "engine, box cars, and caboose" model. The software architecture team is the engine pulling the train, the box-cars are the 'softcrafters' who are very good in one specific domain; and the integration team is the caboose, getting the pieces of software and making sure they can be integrated in a continuous manner. Note again that software architecture is not project management.

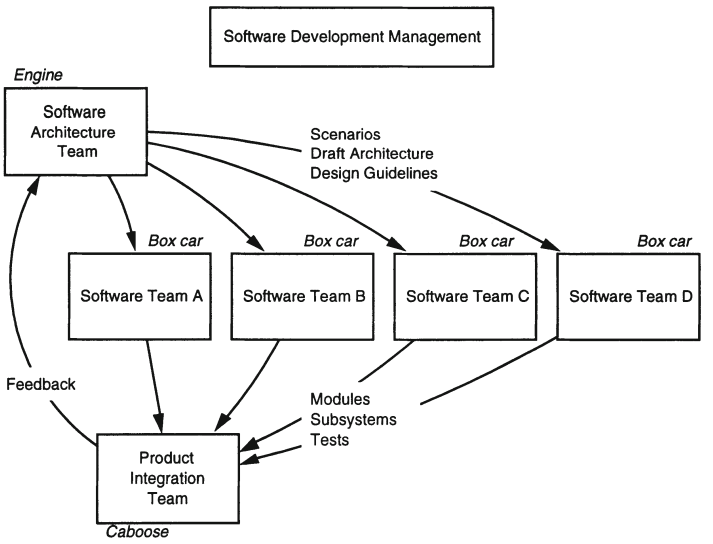

Figure 2. The engine and caboose model

This does not preclude a large company or organisation from having some R&D activity related to software architecture, nor some staff-level working group that overlooks the overall practice of software architecture

across projects, but we are describing the software architecture team of a given project.

Some large organisations have set up two levels of software architecture activity:

- one group at the corporate level, whose purpose is to capture and diffuse the best practice in that area, or to oversee large-scale architectural reuse, and
- architecture teams closely associated to actual projects,

with some circulation of individuals from one group to another (cf. Fig. 3)

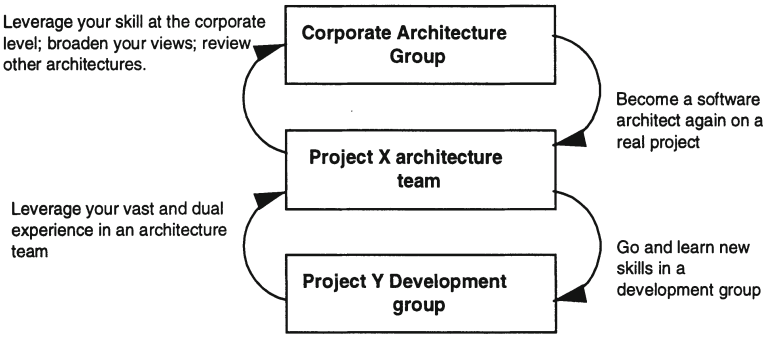

Figure 3. Recycling software architects

4.1 Size of the team

There is no simple absolute answer to this question. However, we will share a rule of thumb we have used which seems to correspond to the few data points we know from successful architecture teams that have been in place for years.

For a new, large, unprecedented project, one software developer out of 10 is on the architecture team during the inception and elaboration phase when architectural design is the preponderant activity. Then this can be reduced to one out of 12 or 15 as the project moves into the construction phase or during evolution cycles.

What happens to the disappearing architects? It is likely that the overall software development organisation will grow during the construction phase, hence the ratio is reduced. Also some of the software architects of the initial team can become technical leads for the various development groups. This evolution is beneficial from several aspects: since they have been part of the initial architecture team, they play a positive role in communicating the architecture and its principles to various parts of the project, or conversely, they bring new issues and difficulties to the attention of their former team-

mates, with whom they had developed "high bandwidth communication links." In a large project, they play the role of "remote sensors" for the core architecture team, thereby contributing to maintaining the system's integrity. As a result, the evolution of the architecture through the latter phases of development can be smoother, based less on adversarial or conflicting relations between an architecture team and relatively foreign development teams.

Continuity is a key aspect. Eb Rechtin pleads for the architect to remain in charge until the project is delivered to the customer, whereas management may be tempted to "recycle" an architecture team to work on a new project as soon as the architecture of the current project is deemed "complete", i.e., at or soon after the end of the elaboration phase. The right balance is probably half way: keep enough architects on the project to guarantee the architectural integrity and to make any changes and improvements to drive it to a successful conclusion. This is where a team of architects offers more flexibility than a single architect.

4.2 System architecture and software architecture

In organisations that develop and integrate systems (composed of hardware and software, sometimes with all kinds of other devices), usually there is a strong *system* architecture function. Is software architecture just part of it? We have found that software issues are sufficiently distinct from system issues, and that the skill set of a software architect is significantly different from the other specialities present in the system architecture group to warrant the creation of a well-defined separate group to deal with software within the system architecture group.

However, software is more often the central issue, as hardware becomes more and more of a commodity, and the system architecture and software architecture functions tend to merge into a single entity.

5. TRAPS AND PITFALLS

Even when all of the pre-conditions are met, software architecture teams still fail. Over a large range of projects, Rational consultants have seen and analysed some of the reasons for these failures. We addressed some of them indirectly already such as

- Inexperience.
- Lack of domain experience
- Lack of software development experience
- Architecture team acts as a committee

Other reasons why the architecture fails to meet the needs of the software developers are:

- Undefined authority, undefined responsibilities
- Architecture team works in an "ivory tower"
- Not focused on design
- Imbalance in the team composition
- Procrastination

Let us examine some of these traps and how to avoid them.

5.1 Lack of authority

Another story to illustrate the importance of authority.

> A complex telecommunication system started without much of a software architecture. After some time, an architecture team of talented people was created. But the group leaders of the development organisation—the "barons" as the architects called them—took this innovation as a serious challenge to their position and authority. Therefore they ignored whatever was coming out of the software architecture effort, protected what had been their "turf" for a couple of years now and blocked most of the communication between developers. Things did not progress well, the architects became tired and disillusioned and management—unfamiliar with the concept of a software architecture team—did not provide much support. The architects then left the company one after another. The "barons" had won, but the project was now two years late, and still without much of an architecture.

Defining the exact extent of the architects' authority is even more important when a consultant or an external organisation is fulfilling this role.

5.2 Ivory tower

We met the software architecture team of the large multinational company in various public events, and liked their views on software architecture. A few months later, we were called to help one of their divisions and we referred to the company's architecture group. The division management had never heard of the group.

There are other simpler ways of developing the ivory tower syndrome:

- Put the team in another building
- Present software architecture as some kind of sinecure for ageing, or weary designers

The best way to avoid this is to communicate, communicate, communicate. The architecture team, especially when recently created, must

make its activity visible by publishing a partial draft of a software architecture document, designing notes, and inviting other people to contribute or review. Every week they should be interacting with the rest of the development organisation.

5.3 Imbalance

An unbalanced software architecture team may have difficulties producing a complete and balanced architecture. A lack of understanding of the domain may lead to an architecture that solves computer science problems only; the same is true if the speciality or main field of interest or experience of the architecture team dominates.

> On a project that had no clearly defined software architecture team, we were trying to find out what the architecture of the system was in order to assess it. Interviewing various groups, we got four totally different 'architectures', each group claiming that it was in its charter to define the architecture (cf. Fig. 4).

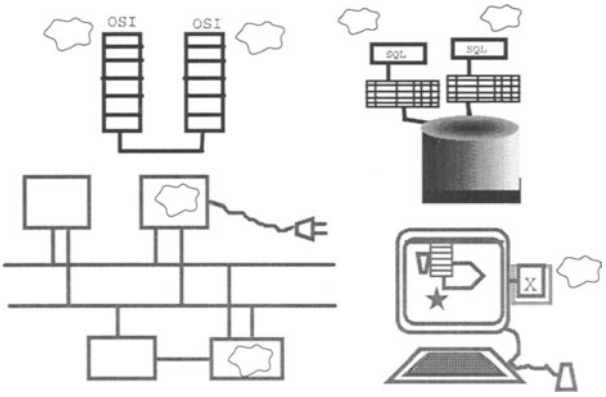

Figure 4. Four specialities, four architectures

1. The telecommunications specialists told us that the main characteristic of the system was its distribution over a vast wide-area network, and that the telecommunication aspect was driving everything. They described OSI protocol stacks, and mentioned some 'code' to be developed as application services elements, hooked at OSI layer six and above.
2. The data engineering leader described the system as one huge database (with impressive E-R diagrams to support this) described the commercial database at the core of the system, and how everything could be managed

by the database, networking, computer-human interface, with only a few algorithms that would have to be programmed in some other language than SQL.

3. The Computer-Human Interface group had done some in-depth study of ergonomic issues, and claimed that the single biggest differentiator between the system under development and its predecessor was in the Graphical User Interface. They spoke about X servers and clients, revolutionary widgets and gadgets, as well as new peripherals.
4. Finally, the systems engineers had a view entirely centred on the computers and the networks. Software was only a strange ingredient you sprinkle on top of the boxes, once you have defined them.

It was hard to believe that they spoke about the same system. Moreover, they were not very concerned about the lack of architecture. Each group had enough to satisfy its concerns in terms of architecture as they defined it.

Morale: find people who have broad experience. Learn about the fields that are under-represented. Eventually bring in specialists to consult with the architecture team.

5.4 Confusing a tool and the architecture

This story also illustrates another danger, which is to acquire a tool or a component that plays a major role in the architecture, and then being led to believe that the architectural design is done—that the tool has defined the architecture.

Vendors of certain major products, especially databases or GUI, would like you to believe that they provide a complete architecture so that once you have bought their product your architectural design is done, that everything will gravitate around their product.

5.5 Procrastination

> Attendre d'en savoir assez pour agir en toute lumière, c'est se condamner à l'inaction. *Waiting to know enough to act in full light is to condemn oneself to permanent inaction.* Jean Rostand, French biologist

Procrastination is the worse trap of all; it is insidious, and the best teams can easily fall in it. We have already written that the practice of software architecture is a long and rapid succession of suboptimal tactical decisions, mostly made in partial light. There is a great tendency (especially with people who have scientific training) to want to analyse more, find more options, go further down the paths in order to make the "right" choice, the optimal choice. This slowly kills the project. When decisions are not made, other teams cannot make any progress, or their progress is in jeopardy. They

will sit on their hands, waiting for the architects to decide or lose confidence in the architects and make their own decisions.

> A project was selecting a tool set to build one of its major components—its graphical user interface (GUI). Several candidate tools were studied: each had its advantages and disadvantages, none was a perfect match. The team delayed the choice, hoping for a better product to appear or for one of the three products to make some significant progress. Meanwhile, the development organisation in charge of the Computer-Human Interface could not make any progress as too much of its work was held back by the lack of tools. Then some hardware choices were not made, because of the lack of understanding of the consequences on the software. It came to a point where the whole project started to suffer significantly because a choice was not being made. The differences between the various GUI products were minimal. Drawing the winner out of a hat would have been better than waiting. But this was not rational. More requests for proposal were issued, more evaluation copies bought, more studies commissioned to establish more criteria for the choice and so on. The sad conclusion of this story is that the final choice was made outside of this team, on a purely political basis, with no consideration whatsoever for any of the technical arguments.

We have found that, in many respects, it is better to make a decision now, in the dark, explaining clearly that it was made with little knowledge of the consequences, rather than suspend a whole project for weeks. Then it is up to the architects watch in the following weeks and months to see if the decision brings more trouble or solutions. If it really becomes a problem, then do not hesitate for one minute: change it. Do not compromise or tergiversate, be very decisive. But do not leave a known evil for another that you do not know, yet. You have put the development back on track. Explore your alternatives, including the cost of changing tracks.

> Le courage consiste à savoir choisir le moindre mal, si affreux qu'il soit encore. *Courage is knowing how to choose the lesser evil, as awful as it still is.* Stendhal

Do not become too focused on the technical optimality of the solution; there is rarely an optimum when taking into account all factors, cost and schedule included. The relative advantages of this or that solution are often minor, and are not enough to justify important delays.

This ability to rapidly make tactical decisions and live with the associated anxiety is one of the elements that distinguishes the software architect from other software developers. It takes a while to get used to it. Some people never do it, and will always hide behind an architect that has more courage

than themselves. *The life of a software architect is a long and sometimes painful succession of suboptimal decisions taken partly in the dark.*

Finally, it is very hard to eradicate the idea that software development is not a linear process: it proceeds by trying out ideas to validate them. Therefore, it is OK to start coding things before extensively studying every detail on paper. It is also better to make a choice now (even though it may be the wrong one) and discover early in the project that it is wrong, than to wait forever for the ideal, complete, perfect answer to fall from the sky.

Architects can act as the "conscience" of the products. When management appears to be procrastinating on key issues affecting the product, the architects need to prod them to get the decisions made early. Architects also need to tell management the bad news, including negative results and failed prototypes, early enough so that adjustments can be made.

6. THE PERSONALITY OF THE ARCHITECT

Is there a certain psychological profile that suits the role of the software architect? Maybe. Based on original ideas of the Swiss psychologist Carl Jung, American psychologists Isabel Myers and David Kersey developed, in the 50's and the 60's, a classification of personalities which has had some success in corporations throughout North America, most notably under the label "Myers-Briggs Type Indicator." They developed psychological tests to classify individuals according to four major traits:

1. Extroversion (E) versus Introversion (I)
2. Sensation (S) versus Intuition (N)
3. Thinking (T) versus Feeling (F)
4. Perceiving (P) versus Judging (J).

Although these four characteristics are not binary, but rather a scale—one can be extroverted to some degree—one may classify individuals in 16 "bins" of irregular size, labelling each bin with the letters indicated above: ESTP, ISFJ, etc. The type ESTP would therefore describe a person whose personality leans towards Extroversion, preferring Sensation over Intuition, relying on Thinking more than on Feeling and using Perceiving rather than Judging. Over three decades psychologists studied common characteristics of each of the 16 groups of individuals, notably how they fit in their working environment. They refined the model to introduce "mixed types," taking into account the traits where the test does not clearly lean towards a letter or another, marking this with the letter X, such as EXTP for someone who would be in between ESTP and ENTP.

All this preamble is to tell you that it seems that good software architects are found among the INTJ or INTP types. Not much surprise about the 'I':

most people in scientific or technological fields are introverted. The NT part is the Promethean temperament: the 12% of the population who loves intelligence, power over nature, competence, skills, and their work. NT types like to be liked for their ideas. When in a leadership position, they are visionary leaders. They do not like routine. They are the vectors of change.

David Keirsey nicknames the INTP the "Architect", the "Abstractionist". Abstract design is their forte and coherence is the primary issue. They are curious, rational, and theoretical. The world exists to be understood. They are the logicians, the philosophers of systems. They exhibit a great precision in thought and language. They easily detect contradictions and flaws. But they can also become obsessed with analysis or the gathering of more data.

Keirsey nicknames the INTJ the "Scientist." INTJs are the most self-confident of all 16 types, with a great awareness of their own power. Authority or slogans have little impact on them, unless it makes sense. They can easily make decisions, bringing issues to closure. Unlike the INTPs, they need only to have a vague, intuitive impression of the unexpressed logic of a system to continue surely on their way. They rapidly discard theories that cannot be made to work. They are better at generalising, classifying and demonstrating than INTPs. They are less likely to procrastinate.

Although we do have some very limited evidence that successful architecture teams are primarily composed of INTJ and INTP, other types are useful to achieve a good balance as the team grows: for example an ISTJ—the highly dependable "trustee"—would keep track of things in a large project. While some extroverted types could improve communication. Thus, we would satisfy one of Katzenbach and Smith's axioms for a team: a blend of technical and functional skills, problem-solving and decision making skills, and interpersonal skills. The bad news is that INTJs and INTPs represent only 2% of the general population. After selecting people based on their expertise, you may not have much latitude left, unless you are in a big company with deep pockets.

7. SUMMARY

- Designate a software architect, or assemble a small team of software architects who share a common goal or vision of the product.
- The software architecture team must be experienced in both the problem domain and software development.
- Software architects should be fully dedicated to their task; in particular, the role of software architect is usually not compatible with that of project manager.
- The architect(s) and the project manager are joined at the hip.

- Establish a charter of the software architecture team which clearly defines its role and responsibilities, and establishes its authority.
- Do not isolate the software architecture team—it is a software development group among other software development groups.
- Common pitfalls for a software architecture team include: lack of experience, undefined authority or isolation, an unbalanced mix of technical skills, lack of focus on the design, and procrastination.

REFERENCES AND FURTHER READING

Grady Booch, *Object Solutions*, Addison-Wesley, Menlo Park, CA, 1996.

Frederick P. Brooks, Jr., *The Mythical Man-Month—Essays on Software Engineering*, 2nd edition, Addison-Wesley, Reading MA, 1995.

Carl Jung, *Psychological Types*, Harcourt Brace, New York, 1923.

Jon R. Katzenbach and Douglas K. Smith, *The Wisdom of Teams*, Harper Business, New York NY, 1993. They give good examples from business cases of good, great, and not-so-good teams. Then they extract from their examples the underlying principles of what makes teams tick and become "high performance organisations".

David Keirsey and Marilyn Bates, *Please Understand Me—Character and Temperament Types*, Prometheus Nemesis Book Co., Del Mar, CA, 1984. A very practical and easy-to-read explanation of the Myers-Briggs classification, and its use in everyday life, at work and elsewhere.

John A. Mills, "A Pragmatic View of the System Architect," *Comm. ACM*, 28 (7), July 1985, pp. 708-717. In this very lively paper, Mills describes the roles of the system architect: "A whole-system designer, fire-fighter, mediator, and jack-of-all-trades, the system architect brings unity and continuity to a development project—offsetting the inevitable compartmentalisation of modern modular designs." He depicts three slightly different organisations each with a system architect or a team of system architects. He also justifies the necessity of a central architectural function when a project reaches a certain critical mass, so that the number of one-to-one communication links between the various development groups is reduced.

Isabel Myers, *Manual: The Myers-Briggs Type Indicator*, Consulting Psychologists Press, Palo Alto, CA, 1962. She describes the 16 types and the associated assessment procedure.

Eberhardt Rechtin, *Systems Architecting: Creating and Building Complex Systems*, Prentice-Hall, Englewood Cliffs NJ, 1991. Chapter 14 (p.289-293) describes the profile of a system architect, as does the following paper.

Eberhardt Rechtin, "The Systems Architect: Specialty, Role and Responsibility," *Proceedings of NCOSE*, 1994.

Mary Shaw & David Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall, 1996. Chapter 9 speaks about the Education of Software Architects and describes the course being taught at CMU.