The Journal of Systems and Software 81 (2008) 2413–2416

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Controversy Corner

What do software architects really do?

Philippe Kruchten [*]

Electrical and Computer Engineering, University of British Columbia, 2332 Main Mall, Vancouver, BC, Canada V6T 1Z4

ARTICLE INFO

Article history:
Received 24 June 2008
Received in revised form 19 August 2008
Accepted 19 August 2008
Available online 28 August 2008

Keywords:
Software architecture
Software architect
Antipattern
Time-management

ABSTRACT

To be successful, a software architect—or a software architecture team, collectively—must strike a delicate balance between an external focus—both outwards: Listening to customers, users, watching technology, developing a long-term vision, and inwards: driving the development teams—and an internal, reflective focus: spending time to make the right design choices, validating them, and documenting them. Teams that stray too far away from this metastable equilibrium fall into some traps that we describe as antipatterns of software architecture teams.

1. Introduction

"—Mr. Beck, what is software architecture?" asked a participant at an OOPSLA workshop in Vancouver in the fall of 1992. "—Software architecture?" replied Kent, now famous for being the father of XP (eXtreme Programming, not the O.S.), "well, it is what software architects do." (Chuckles in the audience.) "—So then, what is an architect?" "—Hmm, 'software architect' it's a new pompous title that programmers demand to have on their business cards to justify their sumptuous emoluments."

In the following four years I was going to lead a rather large team of software architects and I often ask myself that very question: "what do architects really do?" and was also asked this by my management and my customers. Since then I have seen many architecture teams in many countries, in companies of all sizes and various domains, and I have witnessed a wide range of good, not-so-good, and really bad answers to this question.

2. Architects design the architecture, no?

The first obvious answer that comes to mind is:

Software architects should design, develop, nurture, and maintain the architecture of the software-intensive systems they are involved with.

It is not a simple, satisfactory answer, since there is no universally accepted definition of what software architecture is. And this would bring us back to the original question.

So a software development organization, based on its context—domain, culture, assets, staff expertise, etc.—must come up with some delimitation of what constitutes software architecture, and what is beyond software architecture. And this definition, this "thin line in the sand" that separates architectural decisions from all other design decisions, including the detailed ones captured in the code, must be made visible to all parties involved. And it may have to be revisited, redefined, and adjusted as an architecture emerges, and as the team expands and the organizational expertise grows.

Let us assume for now that the architects' responsibilities are the part of both the design and the design decisions that have long-lasting impact on some of the major quality attributes of a software-intensive system: cost, evolution, performance, decomposability, safety, security, etc, and still able to support the functionality expected by its end user.

Then this is what software architects should be focused on, this is what software architects should do: make design choices, validate them, and capture them in various architecture related artifacts.

3. Architectural antipatterns

But things are not so simple. There are several "antipatterns" that will make a software architect or software architecture team fail miserably if they were only to *design the architecture*. An antipattern—a concept introduced by Koenig (1995)—documents a common mistake made during software development. While several authors have looked into architecture antipatterns (Brown et al., 1998; Mowbray, 2001), they mostly focus on antipatterns in the architectural *design*, and not so much in the organization and the process, unlike for example Coplien and Harrison (2005) or Ambler et al. (2005).

[*] Tel.: +1 604 8275654; fax: +1 604 822 5949.
E-mail address: pbk@ece.ubc.ca

Here are a few common antipatterns:

3.1. Antipattern: creating a perfect architecture, for the wrong system

A software architect who is not communicating regularly with the customer, the end users, or their representatives (e.g., the product manager) is likely to miss the target, particularly as the target is moving, or rather, as the target is only gradually understood. Ambler and his colleagues call it "Goldplating."

3.2. Antipattern: creating a perfect architecture, but too hard to implement

A software architect who does not understand the (maybe limited) skills, capability and experience of the implementation team(s) that will continue and finish the work will create enormous levels of stress and frustration, and likely not deliver a quality product in time. The architectural effort has turned into a computer science research project. In Ambler et al. it is called "Strive for Perfection"; related is Coplien's pattern "Architect's Implement": Involving the architects in implementing the architecture would mitigate this antipattern.

3.3. Antipattern: architects in their ivory tower

Worse is the architecture team that lives isolated in some other part of the organization—another floor, another building, another country—and who comes up after some months with a complete architecture, out of the blue. To their complete surprise, they will experience rejection: an apparent misfit on both fronts—functional and implementation. This is especially the case if the developers (the nonarchitects) had a few months to make some progress and they have in some ways made some architectural decisions, under some other name. See Ambler's "Ivory Tower" pattern. A special case of this antipattern is the "Architecture watch", an architecture group that only scouts technologies and provides recommendations to other groups, but is not making design decisions and is not accountable for anything, as I have witnessed in two large telecommunication companies. Ambler has two antipatterns that are similar: "30,000 ft. and Climbing", and "Real-world Disconnect".

Finally, there is another issue that cannot be completely ignored; it has to do with *who* has been chosen to be the architects. It is very likely that you have appointed this role some of your most talented staff—good at manipulating abstractions, wide experience of a range of systems and technologies, good communication skills, good domain knowledge, etc.—and you may want to use some of these skills for other tasks than just building architectural views. You want them to speak to the new prospective customers, to show off the organization's technical expertise, to help this or that team that experiences a difficult technical issue. You want them to review the architecture of another project, to take part of a due diligence process to acquire a company, to present papers a conference to strut your stuff, or to merely extinguish some nasty fire. But if you are not careful, this leads to another antipattern:

3.4. Antipattern: the absent architects

No or little architecture design progress is made: the architects are always away doing fascinating things or fighting fires. It is very easy to slip in this mode, especially after some initial good progress and early successes, which brought some fame on the architects.

4. Roles and responsibilities of an architect (or an architecture team)

The roles and responsibilities of an architect can be usefully be captured in some kind of a team "charter" or "mission", that must be adjusted to each organization or project. The list below is derived from the charter of a large team that I led in the mid-1990's (Kruchten, 1999).

1. Defining the architecture of the system. All the usual technical activities associated with design. Understanding requirements, qualities, extracting architecturally-significant requirements, making choices, synthesizing a solution, exploring alternatives, validating them, etc. For certain challenging prototyping activities, the architects may have to use the services of software developers and testers.
2. Maintaining the architectural integrity of the system. Through regular reviews, writing guidelines, etc. and presenting the architecture to various parties, at different levels of abstraction and technical depth.
3. Assessing technical risks.
4. Working out risk mitigation strategies/approaches.
5. Participating in project planning.
6. Proposing order and content of development iterations. For many effort estimation aspects, or for the partition of work across multiples team, managers need the assistance of architects.
7. Consulting with design, implementation, and integration teams. Because of their technical expertise, architects are drawn into problem-solving and fire-fighting activities that are beyond solving strictly architectural issues.
8. Assisting product marketing and future product definitions. The architects have insights into what is feasible, doable, or science fiction and their presence in a product definition or marketing team may be very effective.

As you see, beyond item #1, many activities involve some other party: project management for example, and are not merely focused around the architecture, the design, the architectural prototype. These activities map well onto Bredemeyer's *Architect Competency Framework* (2002) and its five main categories: technology, consulting, strategy, organizational politics and leadership.

We need also to keep in mind that the good architects should bring a good mix between domain knowledge, software development expertise, and communication skills.

Once we identify (and possibly refine) the long list of what we expect the architects to be doing, the next question is: how do we keep a good balance between all these activities"? How do we avoid the temptation to always, day after day, week after week, solve the most urgent problem, or the most interesting problem, or extinguish the latest fire? (The squeaky wheel syndrome.) Or, conversely, it may bring forward the question: do we have the right people with the right expertise in our current software architecture team?

5. Allocating time

To avoid falling in any of the traps or antipatterns mentioned above, and to help maintain a delicate balance between all the forces that an architect is submitted to, I came up in the mid-1990s with a simple time-management practice, summarized in the figure below, extracted from (Kruchten, 2004) (see Fig. 1).

It assumes that you are collecting timesheets, to account of where the architects spend their productive time. This is something done by many organization, though often with task

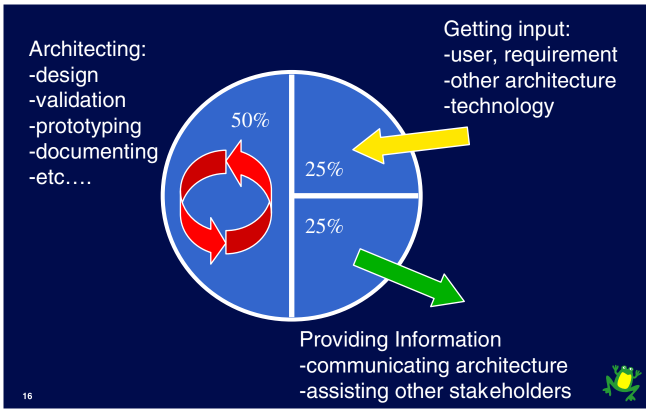

Fig. 1. What do architect really do?

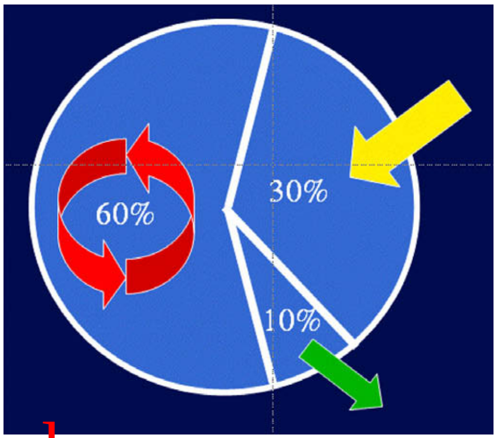

Fig. 2. The [60:30:10] antipattern – goldplating.

categories not quite adapted to what architects really do. In general, both globally across the whole architecture team (if you have more than one architect) and on average over the lifecycle, my recommendation is that the architects should allocate their time in a 50:25:25 (internal:inwards:outwards) ratio as follows:

- *Internal focus: 50%*About 50% of their time focused on architecting *per se*: architectural design, prototyping, evaluating, documenting, etc.
- *External focus:*About 50% of their time interacting with other stakeholders. This in turn has two facets:
  - *Inwards: 25%*25% getting input from the outside world: listening to customers, users, product manager, and other stakeholders (developers, distributors, customer support, etc.). Learning about technologies, other systems' architecture, and architectural practices.
  - *Outwards: 25%*25% providing information or help to other stakeholders or organizations: communicating the architecture: project management, product definition.

This corresponds roughly to the *architectus reloadus* and *architectus aryzus* roles described by Martin Fowler (2003).

The numbers come from my experience in managing a l0-person architecture team in 1992–1995. This apparently crude, off the cuff, partitioning of time has drawn lots of comments, feedback, and push-backs from my colleagues and customers since then, but in the end, unless your situation is really very special, I have not been convinced by any substantive evidence to change the numbers [50, 25, 25] over the last 10 years. (But as is often the case in software engineering, I do not have a scientific proof of my little theory, just anecdotal evidence.).

6. Antipatterns revisited

Let us revisit some of our antipatterns, by simply contrasting the three ratios [internal:inwards:outwards].

6.1. [60:30:10] Goldplating (see Fig. 2)

This software architecture team is not engaged enough with its users, particularly the developers. They are probably doing a good technical job, as they are getting plenty of input, but if they do not regularly provide value to their immediate environment, their input will be too late and be ignored. They have to consistently provide value to the team.

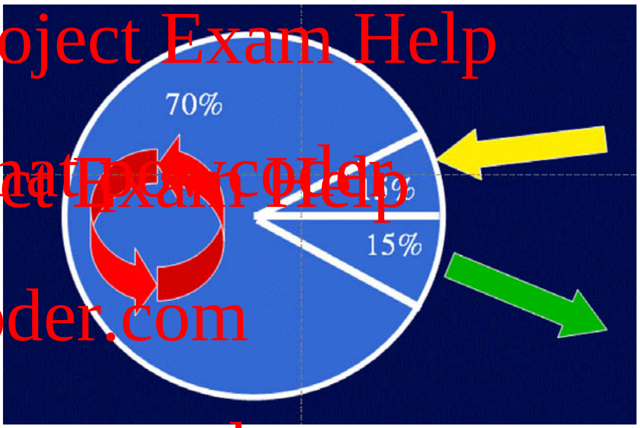

Fig. 3. The [70:15:15] antipattern – ivory tower.

6.2. [70:15:15] Ivory tower (see Fig. 3)

This is a software architecture team that has isolated itself; it is doing far too much navel gazing. They may enjoy themselves, but they are simply not engaged enough with external stakeholders; they are not getting enough input from the users and developers, and they are not providing enough value to their software development organization: such as advocating the architecture, providing assistance to other teams. Even if they do a good job technically, they will rapidly fall off the radar screen, and will be seen as not bringing value.

6.3. [30:40:30] Absent architect (see Fig. 4)

This is a software architecture team that is spending far too much time traveling the world. Unless this is a very mature system that requires very little architectural work (in which case, maybe the team is overstaffed?), they will run into architectural difficulties.

6.4. [25:25:50] Just consultants (see Fig. 5)

This is a software architecture team that is acting more as an internal consulting shop; or their travel and conference budget is simply too large. If their focus is helping internally, maybe this should be made explicit; if their focus is helping externally, maybe they should review their cost-effectiveness?

This is certainly a case where you may start questioning the architecture team's composition and also some of its activities.

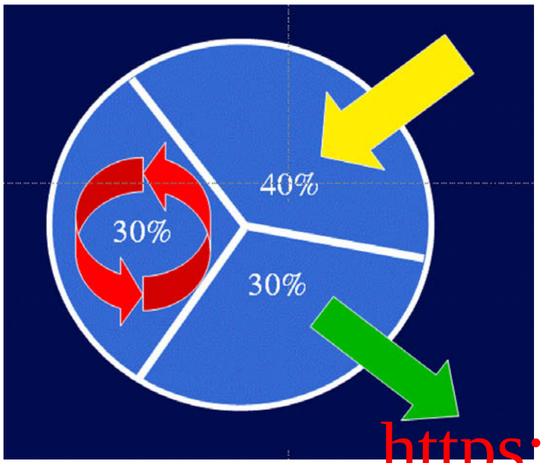

Fig. 4. The [30:40:30] antipattern – absent architect.

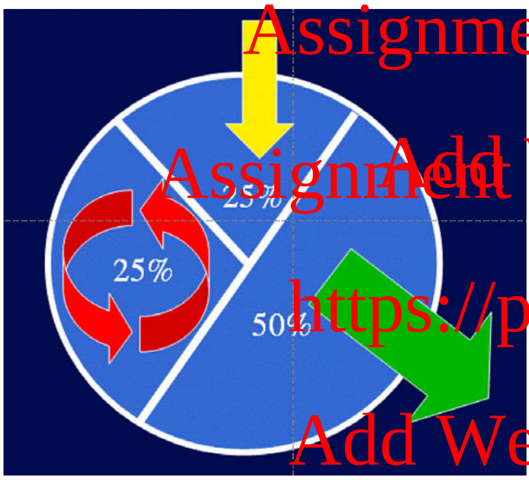

Fig. 5. The [25:25:50] antipattern – just consultants.

Are they doing the job of the product definition team, or should they be simply integrated in one of the development team?

7. Variations

Over time the ratios will fluctuate, but not dramatically. Anything approaching one the antipatterns above starts to be suspicious and indicative of some underlying pending issue, organization imbalance, or misplaced focus. Yes, the ratio will fluctuate over time, and from individual to individual. There will be more internal focus in the elaboration phase of the first development cycle of a rather novel system. There will be more outward focus during construction and transition phases, to assist the development teams. While the important point is the overall ratio for a whole team, and various individual will have different time usage patterns, I would also worry if an individual architect would never go outside his office, never see a user, or on the opposite spend all of his time outside.

8. Pragmatics

This is not rocket science to implement. If you are in a large company that has a time reporting system, then have ‘them’ create the 3 categories above, or map existing ones into these three bucket (if meaningful). Except for absences (e.g., holidays, etc.), have all architect activities without exception fall in one of the three buckets: internal, inwards, and outwards? Reporting accuracy down to the minute is useless—the day or the half-day is often enough.

At first, architects will be somewhat puzzled as to which bucket something goes, so define simple guidelines:

- Who were you working with? Other architects, customers, analysts?
- Who benefited the most? Us, the architects? Or another party? Were you primarily listening and learning, or were you informing, presenting, preaching, convincing? Were you acting as a consultant to another organization?
- Express “blends” in triplets. Blends are activities that are comprised of a combination of internal, inwards and outwards tasks.

For example:

– Defined API for authentication services: [100, 0, 0] × 2 days.
– Explained the architecture to a potential vendor: [0, 10, 90] × 4 h.
– Had a workshop with the database team: [10, 50, 40] × 8 h.
– Attended a software engineering conference: [0, 100, 0] × three days.
– Planned iteration 4 with PMO: [0, 0, 100] × 2 h.

9. Conclusion

To keep an architecture team well-focused and balanced in all its expectations, I have suggested tracking the productive time spent by architects by sorting it in three categories: internal (architecture design), external (both inwards and outwards communication) and keeping them over time roughly in the ratio [50:25:25]. Major deviations should attract the attention of the architect or project managers and possibly some examination of the current focus. I will be happy to hear from the readers their own experience, and any evidence that either refutes of corroborates these results.

Many thanks to the editors and reviewers of JSS for their encouragement and pointy criticisms, as well to readers of previous version, notably the SEI crew.

References

Ambler, S.W., Nalbone, J., Vizdos, M.J., 2005. The Enterprise Unified Process – Extending the Rational Unified Process. Prentice Hall, Upper Saddle River, NJ.
Bredemeyer, D., 2002. Architecture Competency Framework. <http://www.bredemeyer.com/pdf_files/ArchitectCompetencyFramework.PDF>.
Brown, W.J., Malveau, R.C., McCormick III, H.W.S., Mowbray, T.J., 1998. AntiPatterns: Refactoring Software Architectures, and Projects in Crisis. John Wiley & Sons.
Coplien, J.O., Harrison, N.B., 2005. Organizational Patterns of Agile Software Development. Prentice-Hall, Upper Saddle River, NJ.
Fowler, M., 2003. Who needs an architect? IEEE Software 20 (4), 2–4.
Koenig, A., 1995. Patterns and Antipatterns. Journal of Object-Oriented Programming 8 (1), 46–48.
Kruchten, P., 1999. The software architect, and the software architecture team. In: Donohue, P. (Ed.), Software Architecture. Kluwer Academic Publishers, Boston, pp. 565–583.
Kruchten, P., 2004. Training Material for the Course ‘Software Architecture and Iterative Development – Principles and Practice. Kruchten Engineering Services Ltd. (<http://www.kruchten.com/site/courses.html>).
Mowbray, T.J., 2001. AntiPatterns in software architecture. In: 23rd International Conference on Software Engineering (ICSE’01), ACM, Toronto, p. 998c.

Philippe Kruchten is professor of software engineering in the department of Electrical and Computer Engineering of the University of British Columbia which he joined in 2004. He has been a software architect most of his career first at Alcatel and then at Rational Software, where he also led the development of the Rational Unified Process® (RUP)®. He has a mechanical engineering diploma and a doctorate degree in information systems from French institutions.