



# Between the Waterfall Wasteland and the Agile Outback

Eltjo Poort

## From the Editor

Agilists and architects too often talk past each other. In this issue's "The Pragmatic Designer," guest columnist Eltjo Poort helps to bridge the divide by identifying five architecture responsibilities. This enables teams to introspect about how well they are handling each, and encourages them to avoid the extremes. —George Fairbanks

**IN THE PAST** decade, we have seen a significant shift in attitudes toward the architect's role. Many organizations now prefer to allocate the responsibility for major design decisions not to an architect but to teams without a named architect. Sometimes the roles on such teams are identified by such terms as *pathfinder*, *master builder*, *ninja developer*, or *steward*. Some of these roles appear to be merely euphemisms to avoid the (in some circles, dreaded) A-word, but others represent a genuinely different way of looking at architectural responsibilities. In this article, I will share some insights about this shift. These have been gained by a small group of instructors teaching more than 1,400 architects in dozens of organizations across the globe.

Research<sup>1</sup> shows that applying architecture practices improves the

quality of software and the ability to control the risk and cost of delivering it. If organizations want to reap those benefits without a named architect, they need a way to think about the maturity of the architecture function on an organizational level. We deconstructed the role of the architect into a set of responsibilities, and then we reconstructed those into a model that helps organizations assess how well they are crafting architecture and where they can improve—with or without named architects. We fine-tuned this maturity model by applying it in practice for a year. The result presented here might help you recognize the weak spots on your team and find ways to improve.

The model consists of two things: a set of five responsibilities that make up the architecture function (Figure 1) and a way to assess how well organizations fulfill each of them. I will briefly discuss both.

## Five Architecture Responsibilities

The perception of architecture in the field of software engineering has gone through a number of changes since it was first used in that context. Here's a rough sketch of how five distinct architecture responsibilities emerged over the years. In the 1990s, architecture was viewed<sup>2</sup> as a set of structures (components and connectors) that represents an abstraction of a system being delivered—an abstraction needed to deal with the growing complexity of typical software systems. The main architectural activities were<sup>3</sup>

- architectural analysis, with the aim of understanding context
- architectural synthesis, resulting in architecture models
- architectural evaluation, aimed at validating the architecture.

In the early 2000s, a second perception emerged, with a focus on a new responsibility: architects needed to make important decisions<sup>4</sup> to create the right models of their solutions (*right* meaning that they fulfill their stakeholders' needs). If *abstraction* and *structures* describe what the architect creates, the *decision making* refers to how they create it.

Around 2010, partly under pressure of the agile movement's focus on business value, a third perception emerged: the *why* was added to the *what* and the *how* of architecture. This view shed light on the business goal of architecture: to improve organizations' control over risk and cost<sup>5</sup>—not only during design, but extending the architects' responsibility to the delivery domain.

So, we end up with five architectural responsibilities: understanding context, making decisions, modeling, validating, and delivery. Between these five responsibilities are many dependencies. Here are just a few:

- Modeling and decision making without understanding context will lead to wrong models and decisions.
- Modeling actually implies decision making (about decompositions, relationships, and so on).
- If there are no models and no decisions, there is nothing to validate.
- Delivery of unvalidated decisions and models may lead to trouble.

So, fulfilling the five responsibilities in isolation is not enough: they should be fulfilled in a coherent way.

### Balanced Architecture

There's no such thing as good architecture in an absolute sense: the best

one can hope for is an architecture that fits the stakeholder needs in its context. The best-fitting architectures result from paying proper attention to all five responsibilities mentioned. This is not easy; due to such factors as cultural pressures, dogmas, and misconceptions, many organizations ignore some of the responsibilities,

resulting in a flawed architecture function. Two extreme examples are the Waterfall Wasteland and the Agile Outback caricatures described later in this article.

Paying proper attention to all five responsibilities, however, does not mean always giving equal attention to each one: depending on the context,

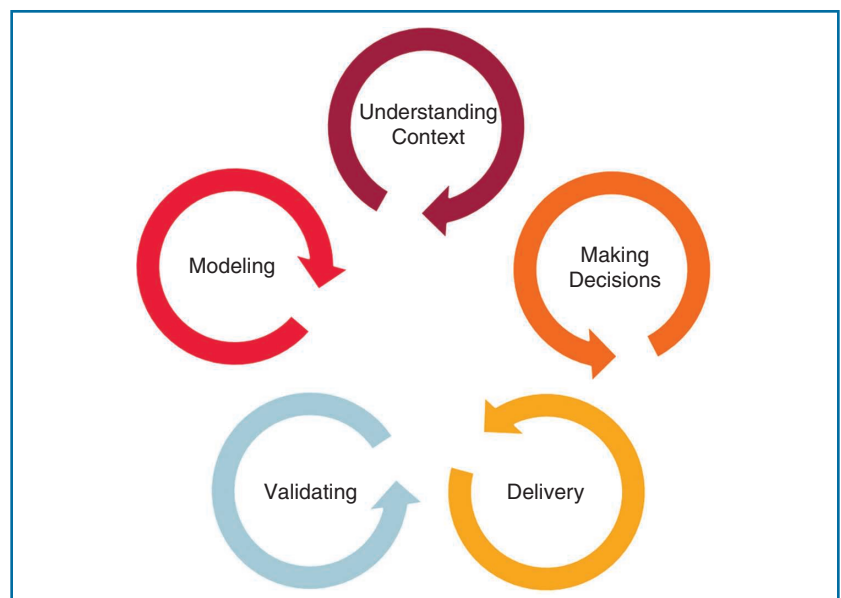


FIGURE 1. The five responsibilities of the architecture function.

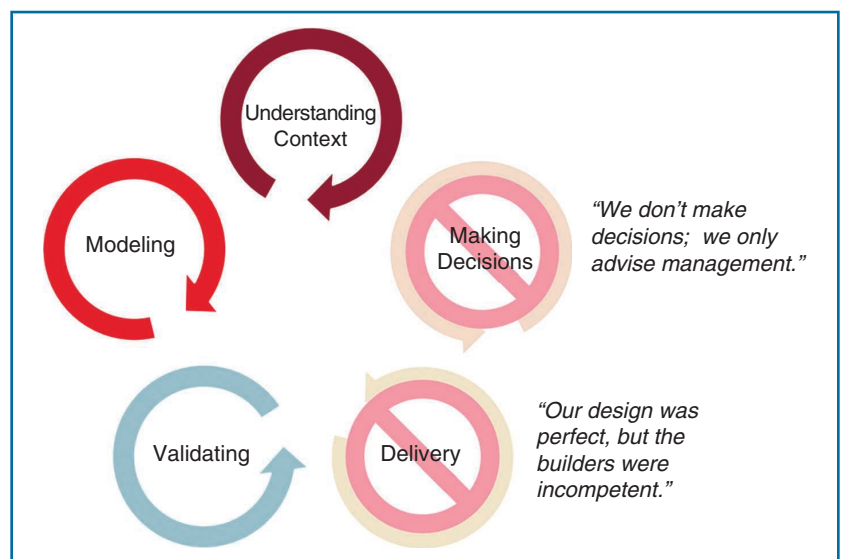


FIGURE 2. The flaws of the Waterfall Wasteland.

modeling may indeed require more attention than decision making, and validation may be more critical in some situations than in others.

When talking to teams, architects, and stakeholders in different organizations, we started to notice some interesting patterns in the way they took up these responsibilities. We created caricatures to highlight the differences between those patterns and called them the *Waterfall Wasteland* and the *Agile Outback*. Note that these are caricatures; they do not exist in real life. They have exaggerated features, may be amusing to some and offensive to others, but can be useful in making a point.

## Caricature One: The Waterfall Wasteland

In the Waterfall Wasteland, the architects are sometimes said to live in an ivory tower. They ignore the decision-making and delivery responsibilities, which they consider to be someone else's. They have a very clear job de-

scription: to create perfect models and validate them against stakeholder needs. If the resulting solution is unsuccessful, it's obviously not their fault. The idea that they would be responsible for decisions or share responsibility for successful delivery is abhorrent to them: it would mean that their success would depend on the capability of others.

Organizations in the Waterfall Wasteland typically have trouble adapting to change: the carefully modeled and validated designs have a limited shelf life and are hard to adapt to new insights gained during delivery. There is a long feedback cycle between architecture and delivery. The (often hefty) architecture documents go out of sync with reality and become ballast and waste (Figure 2).

## Caricature Two: The Agile Outback

In the Agile Outback, teams usually don't have architects. Modeling is avoided since, according to the Agile Manifesto (<http://agilemanifesto.org>), "The best architectures... emerge from self-organizing teams."

This could be (mis)interpreted to mean that modeling is unnecessary or even counterproductive. Teams in the Agile Outback rarely use models to think about or validate designs. Instead, they rely on quick feedback from failures.

Organizations in the Agile Outback produce a lot of direct business value at high velocity in the beginning of a product's lifecycle. However, in our experience, such organizations tend to have problems sustaining that velocity. They often have to revisit decisions and redo work that could have been avoided with a little more forethought. Some architectural decisions are not easy to refactor, and a few hours generating and evaluating alternatives would have been well spent (Figure 3).

## Assessing Agile Architecture Maturity

How can organizations avoid getting stranded in the Agile Outback or the Waterfall Wasteland? How can teams find the right balance and reach the Goldilocks Zone with just enough up front and sufficient adaptive architecture?

As part of our risk- and cost-driven architecture approach, we developed a maturity model based on behavior that we observed in teams—behavior that we found to be a good indicator of how well teams are fulfilling each of the five responsibilities of the architecture function, both individually and collectively. A unique feature of this model is that it allows organizations to identify strengths and weaknesses in their architecture function, without requiring specific roles. Teams can be mature in terms of agile architecture without a named architect or an architecture document—and the model will show that.

Figure 4 shows a form that we have been using to assess teams' agile



FIGURE 3. The flaws of the Agile Outback.

**Table 1. Behavior indicative of maturity.**

Behavior	Description
<b>Understanding context</b>	
Effective stakeholder communication	Business, delivery, and operational stakeholders are actively involved in architectural design. They are easily accessible to explain context, and they frequently help identify key concerns and risks. They are not just asked for approval: there is a continuous feedback cycle in business language between business concerns and architectural design.
Context knowledge managed	Knowledge about architectural context is gathered, validated, and preserved. Specific architectural drivers and the (business) goals behind them are documented and validated.
<b>Making decisions</b>	
Decisions as primary deliverable	Architectural decisions are communicated individually and not only as part of a design document. Stakeholder feedback is gathered on individual decisions.
Prioritized by business impact	Those concerns and decisions that have the highest risk and cost impact on their collective stakeholders are considered first and receive the most attention. This contrasts with setting priorities by checklist, template, or the next deadline.
Justified and documented	Architectural decisions, including the criteria on which they are based and their relation to specific business goals and consequences, are visible to stakeholders. Stakeholders can see which alternatives were rejected and why.
Well-timed decisions	The timing of each architectural decision is a conscious tradeoff between the cost of delaying the decision and the risk of making a wrong decision. This contrasts with making all decisions collectively by management approval of a document or with timing dictated by the next most urgent thing.
Decentral unless...	Architectural decisions are consciously made at the optimal level of decentralization: in decentralized teams, if the benefits of local optimization outweigh the risk and costs of diversity and nonstandardization, or at the central level, if team interests can be conflicting or would cause too much complexity. Responsibility for architectural decisions is shared between those owning the wider context and those owning specific solutions.
<b>Modeling</b>	
Visual model of context	Information-system context is documented and validated (context diagram), showing solution boundary and external dependencies.
Visual models of solution	Appropriate visual models of solutions are created. They show how the architecture addresses relevant stakeholder concerns and are used as the basis for validating, creating, and delivering solutions. The models are curated and maintained throughout the solution's lifespan.
<b>Validation</b>	
Fulfills stakeholder needs?	Architectural designs are validated before implementation by checking that the architecture can support the most risky and costly anticipated requirements. The thoroughness of this validation (from a 1-h whiteboard session to a multisprint architectural prototype) is a conscious tradeoff taking into account business criticality, size, complexity, and volatility.
<b>Delivery</b>	
Architectural runway recognized	The product backlog or project plan contains user features/epics/stories as well as items to realize architectural elements and technical debt reductions ("enablers"). The backlog is fed by a variety of sources, including those representing higher-level architectural concerns.
Architecture debt control	Technology upgrades and work to fix architectural shortcuts are visible on the product backlog or project schedule. Decisions to fix debt are based on economic tradeoffs (as opposed to "only if there's time left after the rest is done").
Just enough anticipation	Business, delivery, and operational stakeholders help identify future events that impact risk, cost, and value of solutions, so that teams are not taken by surprise. Dependency analysis is used to start work on architecture runway and technical debt remediation in time.

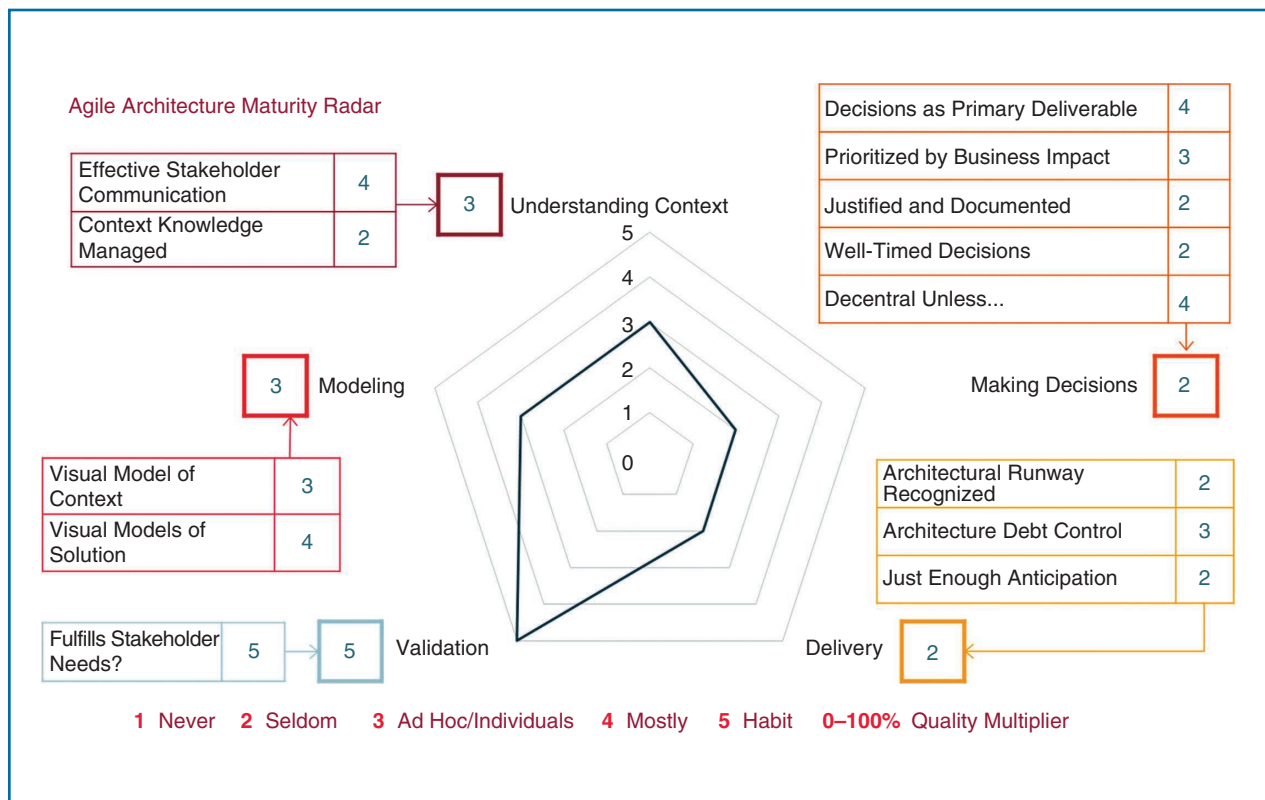


FIGURE 4. The concept of agile architecture maturity radar.

architecture maturity. It shows the behavior indicative of each responsibility's maturity level and a graphical representation of the resulting maturity profile. Table 1 gives a brief description of each type of behavior. The scoring mechanism is quite simple: participants make an assessment of the frequency of the behavior on a scale of 1 (never) to 5 (habit). A quality multiplier between 0% (useless) and 100% (perfect) is used to adjust scores downwards in situations where the frequency of the behavior could give a misleading perception ("We are in the habit of doing this very badly").

### Experiences

So far, we have used this model and its predecessors to assess nine teams in three organizations in the transport

and financial sectors. In these assessments, we conducted 45 interviews overall with architects, team members, and business stakeholders to score the behavior. We used the five responsibilities and associated behaviors as a tool kit to help assessors focus their questions. The assessors were experienced senior architects. They produced expert maturity assessments underpinned by observation. The feedback we received indicates that the model is useful in highlighting where organizations can improve and develop a more balanced way of dealing with architecture. The Waterfall Wasteland and Agile Outback caricatures were found to be both amusing and helpful, especially in two situations: for recognizing the importance of modeling in contexts

with high business criticality and complexity and for highlighting the importance of adaptive design and short architectural feedback loops in volatile environments.

**W**e are now working on our next step, which is to evolve the model from a tool kit for experts to a more objective measurement instrument. One of the approaches we are investigating is to ask teams to assess themselves after explaining the model and showing examples and counterexamples of each type of behavior. We tried this out on workshop attendees, who indicated that it was a useful exercise. An encouraging result was that members of the same team






## ABOUT THE AUTHOR



**ELTJO POORT** leads the architecture practice at CGI in The Netherlands. Contact him at [eltjo.poort@cgi.com](mailto:eltjo.poort@cgi.com).

tended to have very comparable self-assessment scores. So far, this is just anecdotal, and we are working on gathering more evidence. If you are interested in participating, want to share feedback, or try this at home, contact me. 

### References

1. R. Slot, "A method for valuing architecture-based business transformation and measuring the value of solutions architecture," Ph.D. thesis, Faculty of Economics and Business, Univ. of Amsterdam, The Netherlands, 2010.
2. M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*. London: Pearson, 1996.
3. C. Hofmeister, P. Kruchten, R. Nord, and H. Obbink, "Generalizing a model of software architecture design from five industrial approaches," in *Proc. 5th Working IEEE/IFIP Conf. Software Architecture (WICA'05)*, 2005, pp. 77–88.
4. J. Tyree and A. Akerman, "Architecture decisions: Demystifying architecture," *IEEE Softw.*, vol. 22, no. 2, pp. 19–27, 2005.
5. E. R. Poort and H. van Vliet, "RCDA: Architecting as a risk- and cost management discipline," *J. Syst. Softw.*, vol. 85, no. 9, pp. 1995–2013, 2012.

## IEEE Computer Society Has You Covered!

**WORLD-CLASS CONFERENCES** — 200+ globally recognized conferences.

**DIGITAL LIBRARY** — Over 700k articles covering world-class peer-reviewed content.

**CALLS FOR PAPERS** — Write and present your ground-breaking accomplishments.

**EDUCATION** — Strengthen your resume with the IEEE Computer Society Course Catalog.

**ADVANCE YOUR CAREER** — Search new positions in the IEEE Computer Society Jobs Board.

**NETWORK** — Make connections in local Region, Section, and Chapter activities.

Explore all of the member benefits at [www.computer.org](http://www.computer.org) today!

