Software Systems Architecture

Second Edition

Working With Stakeholders Using Viewpoints and Perspectives

NICK ROZANSKI · EOIN WOODS

Software Systems Architecture

Second Edition

mitigate your most important risks first. If you don't use risks to prioritize your work, there is always a danger that you'll do the most interesting or the easiest parts of the architecture work first, when in fact these may be the least important, depending on the system.

EXAMPLE If you're a security expert, it seems sensible to use that knowledge and experience to thoroughly analyze and model the system's security early on. After all, this is likely to be useful, you know how to do it, and it allows you to get something significant out of the way immediately.

While on the face of it this seems to be a sensible approach, when you think about it objectively, it's clear that this isn't necessarily the best way to work. If you prioritize your work according to your preferences, rather than thinking explicitly about risk, human nature means that you're likely to choose things that you're interested in or relatively low-risk areas that allow you to deliver "quick wins" to show progress. In actual fact, the highest risks your system is facing could be in different areas such as performance or something relatively obscure and awkward such as regulatory compliance.

Understanding and prioritizing your risks (in terms of impact and likelihood) will allow you to focus your architecture work in the areas where it is most important. We discuss this approach further in Chapter 13.

Choose Descriptive Names

The names of elements in a model can have a significant impact on its effectiveness for communication. Your stakeholders' understanding of a model will be colored by the names of the model elements because of assumptions they make based on particular words. It is also important to choose good names because names tend to be very "sticky": Once a name has been understood and discussed, it becomes part of the common language for a project and so is very difficult to change, even if it isn't a very good name.

When initially creating a model, it is easy to give elements misleading or ambiguous names because you are still trying to understand the role and responsibilities of each element. This makes it important to keep revisiting names as you develop the model to ensure that the element names you finally choose are accurate and meaningful, helping readers of the model to easily grasp its fundamental structure and the role of each element within it.

Define Your Terms

A particular problem with graphical modeling notations is the tendency to draw the diagram that represents the structure of the model and then consider the model complete. Of course, the model isn't complete because none of the symbols on the diagram have really been defined, and the model is wide open to misinterpretation. This problem isn't limited just to models created with graphical notations; it is quite common to encounter quantitative models (e.g., spreadsheet-based performance models) that are very difficult to interpret due to missing definitions of the various elements and relationships captured in the model.

As you develop a model, be sure to spend enough time carefully defining all of its elements so that their meanings, roles, and mappings to the real world are all clear and not open to different interpretations.

Aim for Simplicity

The simpler a model is, the easier it will be to use and the more likely it is that its audience will find it effective. However, if a model is too simple, it will also fail because it no longer represents the essential features that interest the audience. You must aim for a balance somewhere between simplifying a model so far that it is no longer a valid and effective description and overcomplicating it to the point that the model is difficult to use and maintain.

Most models start out being simple and well structured, but as more detail is added and more special cases are considered, their complexity often increases significantly. Increasing a model's complexity quickly reduces its effectiveness for communication and analysis.

As your model develops and becomes more detailed and complex, continually review it yourself and ask others to do the same in order to assess its effectiveness. If a model becomes too complex to use easily, consider replacing it with a number of simpler, related models that contain the same information but in a more accessible form.

Use a Defined Notation

Nearly all models use some form of notation to represent their content, whether it be a graphical notation, a symbolic or mathematical notation, or even program code—you have many choices. Most notations can be used in different ways, yet none of them will match your exact needs, so you will often have to extend them. The result of this is that when your stakeholders read a new model, it can be difficult to be sure what notation it is written in and what the notation means.

Without a clearly defined notation, the readers of your model must rely on their intuition and your explanation and commentary to interpret the model. The difficulty of interpreting the notation can easily become a barrier to understanding.

For every model you create, be sure to define the notation you use carefully, so that stakeholders have no doubts about its meaning and can focus on the content of the model, rather than struggling with its representation. Even with informal models like sketches, define your notation so that the sketch can be interpreted even when you are not available to explain it.

Beware of Implied Semantics

Diagrams provide rich semantics that you can use to impart many shades of meaning to the reader. For example, a layered architecture is usually represented in a diagram with the layers positioned vertically above one another, with the most important layer at the top. We are all familiar with such diagrams, and it is not usually necessary to explain what this notation means.

This is a valuable time-saver, but it also means that any diagram that looks like a set of layers may be interpreted as one, whether the architecture is layered or not. If this is not your intention, this can lead to confusion or worse. In such a case, you should either note somewhere that vertical position does not imply layering, or even better, reorganize your diagram so that your readers do not get the wrong impression.

There are many types of implied semantics that may require you to redraw your diagrams.

- Vertical positioning of similarly sized objects on the page can be misconstrued as implying layering or some other hierarchy.
- Left-to-right positioning may be interpreted as a flow of control or a sequence of events.
- Putting one element inside another may imply ownership or containment.
- The sizes of diagram elements can suggest their relative importance.
- Coloring of diagram elements or text may suggest importance (or otherwise). Similarly colored elements may be assumed to be related in some way.
- Using certain icons for your elements, such as a stick person or a graphic of a computer or a disk, may mislead your readers about the nature of those elements.

These diagrammatic conventions can be very useful, but you should ensure that you use them with care to impart the meaning that you intend.

Validate Models

Models are an approximation of reality. This is what makes them valuable, allowing us to focus on important details to understand, communicate, or analyze, but it is also a potential weakness because you can never be sure whether the approximations you made have rendered the model invalid.

This aspect of models means that it is important that you continually validate your structural models for consistency and practicality and your analytical models for correspondence to the real world. You can do this via expert review, technical prototyping, and checks of your model against the real world. The important point is to validate your models often enough and thoroughly enough to be confident that they will be useful in their intended roles.

Keep Models Alive

Things change during a software development project: Requirements come and go, new constraints emerge, and priorities change. This potential for continual change is something you have to deal with in order to deliver effective systems that meet the real needs of your stakeholders.

The need to absorb changes means that you cannot expect a model developed at the start of a project and left unchanged to still reflect reality by the time the system is delivered. The challenge you face is that if your models stop reflecting reality, they soon stop being used and "die."

In order to avoid the premature demise of your models, it is important to regularly update them so they will continue to be relevant. Although you do not want the maintenance of models to become a major burden that slows the project, you need to get the balance right by investing enough time and effort to keep them current. Scheduling a small amount of routine model maintenance activity into your weekly plans can be a useful aid for achieving this.

Keeping models up-to-date is also much easier if you keep them as simple as possible, by focusing on those core concepts that really need to be modeled to aid analysis and communication. We have often found that the main architectural structures of a system do not change very quickly when compared to the details of the system built upon them. These key structures are also the things you need to model in order to understand the system (it is very difficult to re-create these models from the code later). Keeping such models current is much less difficult than maintaining comprehensive models of every aspect of the system.

MODELING WITH AGILE TEAMS

With the emergence of agile development as a mainstream approach, it has become a common situation for software architects to be working with agile teams. Common wisdom is that architects and agile teams mix like oil and water

because of concerns on the part of the agile team about big up-front design and unnecessary documentation. However, our experience is that working in and with good agile teams isn't difficult, and there's a lot to be gained from it, but it is necessary to be flexible with respect to the approach that the team wants to use.

We've found that working with an agile team involves an intelligent application of many of the points that we've already made in this chapter, in particular:

- *Work iteratively*; rather than trying to produce complete models in one go, deliver incrementally developed and refined ones.
- *Share information via simple tools* rather than assuming that everyone will be happy to use a complicated modeling tool that works well for you (agile developers probably won't).
- *Ensure that there are customers for all of your models* and you know what they'll use them for (even if you're the customer); otherwise you're not modeling with a definite purpose.
- *Create models that are good enough* rather than aiming for perfection, which is both unattainable and probably less useful given the time it will take (although make sure that they are good enough!).
- *Focus on architectural concerns that solve problems* that the team is having or will have to clearly differentiate the architecture work from the core development work (unless, of course, the team is clearly struggling, in which case you'll need to step into the detailed design too).
- *Create executable deliverables* such as prototypes or executable models to help validate ideas, communicate with the development team, and bring your work alive for them.

To help place modeling in context for agile teams, consider introducing them to the ideas of agile modeling (AM), which is a set of values, principles, and practices aimed at ensuring the effectiveness of modeling activities, with particular reference to the values of the agile approach. AM embraces and extends many of the points we've already made in this chapter, being based on the recognition that models are only ever an approximation of reality and as such must always be developed with an explicit aim to be useful in a particular situation.

CHECKLIST

For each model you have produced, ask yourself the following questions.

- Does the model have a clear purpose and audience?
- Is the model going to be understood by its audience (business and technical stakeholders, as appropriate)?

- Is the model complete enough to be useful?
- Is the model as simple as possible while still being detailed enough for its purpose and audience?
- Have you clearly defined the notation(s) used in the model?
- Is the model well formed; that is, does it conform to the rules of the modeling language you are using?
- Do model elements have meaningful names and definitions?
- Is the model internally consistent and consistent with other models?
- Does the model have a level of abstraction appropriate to the problem to be solved and the expertise of the stakeholders?
- Does the model have the right level of detail? Is it sufficiently high-level to bring out the key features of the architecture? Does it present enough detail for a specialist audience?
- Have you provided a definition of the terminology and conventions used in the model?
- Does your model have appropriate scope? Are the boundaries clear?
- Is the model accompanied by an appropriate level of supporting documentation?
- For quantitative models, does the model have sufficient rigor (mathematical basis) and an appropriate degree of complexity?

SUMMARY

The most important parts of any AD—and sometimes the only things that are actually produced—are its models. Models are a way to represent the salient features of the system and to communicate these to stakeholders. A good model can make all the difference when helping stakeholders understand your architecture. The AD consists of a collection of views, and each view consists of a collection of models (plus other elements such as principles, standards, and glossaries).

There are three broad classes of models, two formal and one informal. The two classes of formal models are *qualitative models* (which illustrate the key structural or behavioral elements of the system) and *quantitative models* (which make statements about measurable aspects of the system). Both are useful, although architects typically focus on qualitative models because there often isn't enough detailed information available to do any reliable quantitative analysis. We refer to the informal models as *sketches*, and they are used primarily for communication with less technical stakeholders.

A model is only an approximation of reality, and the architect must always be aware of its simplifications and approximations (and make stakeholders aware of these also).