



Just Enough Anticipation

Architect Your Time Dimension

Eltjo Poort

A bit of planning is indispensable to anticipate events that will affect your software's risk profile. Traditional architectural planning emphasizes the spatial dimension (for example, components and connectors) but neglects the time dimension. Consulting IT architect [Eltjo Poort](#) makes the case for an explicit representation of architectural evolution along the time axis and reports on the experiences with architecture roadmapping in his practitioner community. —*Cesare Pautasso and Olaf Zimmermann*



ARCHITECTS IN THE digital world, such as software, enterprise, or solution architects, derive their role name from architects in the civil-construction world. The metaphor works on more than one level: architects in both worlds are responsible for conceptual (and structural) integrity and are the content leaders with overview over design and realization, making key design decisions and drawing blueprints.

However, the digital world differs considerably from the civil-construction world in at least one aspect. IT-based solutions such as application software, IT infrastructure, and IT services change far more frequently than buildings do. After all, bits and bytes are easier to change than brick and mortar, right?

The definition of architecture in the ISO/IEC 42010:2011 standard explicitly mentions evolution.¹ Change is also the central theme in modern software development practices such as agile de-

velopment and DevOps. Nevertheless, the digital-architecture disciplines seem to be lagging behind a bit in this development, perhaps hindered by the metaphor that gave them their name. My recent experiences indicate that evolution and change should be given their proper place in the digital-architecture world. So, time should become a first-class concept for architects of software, infrastructure, services, enterprises, and so on.

Issues with Time-Agnostic Architectures

Many software-intensive systems are part of a complex application landscape. They form systems of systems or software ecosystems with myriad interdependencies between commercial and custom-made software, hardware platforms, and organizational entities, all with their own evolution cycles. Such landscapes now occur in all industries: my experi-

DEALING WITH DEPENDENCIES

Popular architectural styles such as service-oriented architecture (SOA) and its microservices¹ variant reduce the pain of ever-changing dependencies by decoupling subsystems or components. SOA decouples applications in an IT landscape by hiding their internal behavior behind service interfaces. Using the right tools and protocols, such architectural styles achieve independent deployability at the technical level.

However, these architectural styles can provide only limited relief regarding logical dependencies. After all, a service's consumers can never use a feature that hasn't yet been implemented in that service. If they need that feature, they must wait, and the service might gracefully fail in the meantime. In other words, logical dependencies are related to the arrow of time.

Reference

1. S. Newman, *Building Microservices: Designing Fine-Grained Systems*, O'Reilly, 2015.

management discipline, by addressing time-triggered risks. This practice also increases the documentation's practical value in cases in which the future state turns out to be a moving target. When the world keeps changing, documentation that acknowledges change stays more relevant than documentation that doesn't.

Architecting Time: An Evolution Viewpoint

According to ISO/IEC 42010:2011, architecture documentation consists of views that represent the architecture from certain viewpoints. These viewpoints aim to demonstrate to stakeholders how the architecture addresses a particular set of their concerns. Philippe Kruchten's 4+1 view model gives five good examples of viewpoints that do this for common stakeholder concerns.³ How about adding an evolution viewpoint that shows how the architecture addresses the impact of changes in the solution's environment?

Nick Rozanski and Eoin Woods introduced an *evolution perspective* in which the architecture explicitly considers change.⁴ One activity related to this perspective is to characterize a system's evolution needs. To do this, an evolution viewpoint first identifies future events that will have an architectural impact on the solution and then shows how the architecture anticipates them. Considering architecture as a risk- and cost-management discipline,² we're interested in the events' architectural impact in economic terms: risk, cost, and business value. Translating the events' direct technical impact into those terms helps us communicate about the events with business stakeholders⁵ and helps us select the most important events if we can't deal with all of them.

ence ranges from banks and insurance companies, to transport and logistics, to safety, justice, and other public-sector domains. In these landscapes, a time-agnostic architecture is a perishable good: its best-before date is often only weeks in the future.

I've observed issues such as these:

- architecture documents that are perpetually "almost finished" (delaying the projects dependent on them) or are already obsolete when they're issued;
- development based on architectural decisions that have already been revoked (to address changing circumstances); and
- difficulty planning ahead, caused by lack of insight into architectural constructs' time dependencies.

One way to address such issues is to design the solution's evolution into the architecture. At CGI, we started developing this practice in situations with many logical dependencies,

both inside and outside our solution scope. (For more on dealing with dependencies, see the related sidebar.) As part of our *risk- and cost-driven architecture* (RCDA) approach, we created architecture documentation that not only describes the current situation and expected future situation but also identifies and deals with architecturally significant events on the way from the current situation to the future situation. (RCDA is a set of architecture practices and principles CGI uses to design IT-based solutions for clients in a lean, agile manner.²) These events can be changes in the logical dependencies, such as a feature becoming available on a service, or an external system changing its behavior. But, as the following examples show, they can also be other occurrences that affect the solution's risk, cost, or business value.

Explicitly anticipating such events not only helps address the issues just identified but also is instrumental in fulfilling architecture's role as a risk

TABLE 1

Some future events with architectural impact.

Event	When expected	Impact type	Impact
A competitor releases its next-generation product.	4th quarter 2017	Risk + business value	If we don't match our competition's new features, our own product will be harder to sell, and we'll lose revenue.
Microsoft discontinues Windows XP support.	Apr. 2014	Risk	Vulnerabilities are no longer patched. This implies a security risk—for example, the risk of intrusion and data leaks.
Our Corilla license contract expires.	May 2017	Cost	We could reduce costs by switching to an open source alternative.
A new version of Java EE (Enterprise Edition) application servers (for IBM WebSphere and JBoss) is released.	Nov. 2015	Cost	We could reduce maintenance costs by using new features announced for the next version of Java EE.
The project to build system Y finishes.	1st quarter 2017	Risk + business value	System Y (which is interdependent with ours) will require interface features that our solution currently doesn't support. We must build these features, or our solution will lose its business value.

Table 1 shows some typical events and their architectural impact. As you can see, most of them are associated with risks. This is inherent in the definition of a “future event with architectural impact” in two ways. First, in a view of architecture as a risk- and cost-management discipline, an item’s architectural significance is closely related to its risk (and cost) impact.² Second, any point in time at which a solution’s cost or value significantly changes has a deadline-like nature, and every deadline brings a risk of being too late. In fact, a project’s risk register is often a good source to search for such time-triggered events that need to be anticipated in the architecture.

The second step in characterizing the system’s evolution needs is to identify backlog items for the solution’s evolution, which will potentially end up on the solution roadmap. At this stage, it’s important to understand the dependencies between the solution architecture and the architecturally significant events. The backlog items should be linked

to the components, modules, functions, nodes, and other architectural elements they touch in the design. On the basis of the dependencies between architectural elements, backlog items, and events, the architect can engage in economic reasoning about the roadmap with relevant stakeholders such as product owners, project managers, and product managers.

For an example of an architecture roadmap visualizing these improvement items, including their dependencies on each other and on the significant events, see the sidebar “An Example of Architecture Roadmapping.” The economic reasoning is possible because we previously identified future events’ impact on the solution’s risk, cost, and business value. For example, the team might decide to take on some technical debt to make a release deadline in time for new reporting regulations coming into effect, if their analysis shows that the potential drop in the product’s value (if the deadline is missed) is greater than the interest incurred by the technical debt. (Technical

debt⁶ is a metaphor Ward Cunningham developed. The increased cost of modifications due to work left undone in a system, such as refactoring and upgrading, is compared to the interest paid on a loan. Doing that work is equivalent to paying back the loan.) For more details on this economic reasoning, see “Enabling Agility through Architecture,”⁷ which describes these activities as steps to achieve “informed anticipation” in software architecture.

Like other viewpoints, the evolution viewpoint can be a chapter in an architectural description document or a stand-alone artifact prepared specifically for interested stakeholders. Because this viewpoint aims to deal with change and aims to work in volatile environments, a more dynamic medium such as a project or product wiki might also be suitable.

Anyone with concerns related to change and planning is a stakeholder of the evolution viewpoint—specifically project, program, or product managers; product owners; and architects, designers, or develop-

AN EXAMPLE OF ARCHITECTURE ROADMAPMING

Figure A shows the architecture roadmap for a fictitious scenario; the roadmap uses Philippe Kruchten's color-coding scheme for backlog items.¹ In this scenario, our competitor has announced it will release a next-generation version of its platform BuyYourTripsHere.com in the fourth quarter of 2017. Our team's product manager has identified a crucial feature (*F* in Figure A) that our product, AdventuresBeyondBelief.com, must have in place before the competitor's next-generation release: the ability to check whether a hotel has free Wi-Fi. If we don't have that feature in place in time, we expect a 25 percent market share loss—a potential drop of \$250,000 in business value.

F is a functional feature but drives architecture roadmapping because it depends on an architectural improvement

(*A* in Figure A): our hotel communication platform must be upgraded to the latest version. Also, the free-Wi-Fi information must be exposed on the back office's integration hub (for example, an enterprise service bus), but we can't achieve this in time.

The team decides to perform *A* and *F* in time to beat the competitor and to create a temporary solution bypassing the integration hub. This temporary solution means that the team will accrue technical debt, but we've estimated that the debt's interest will be much lower than \$250,000. Refactoring (*T* in Figure A) to replace the temporary solution with a proper connection over the integration hub is planned for the subsequent release.

Reference

1. P. Kruchten, R.L. Nord, and I. Ozkaya, "Technical Debt: From Metaphor to Theory and Practice," *IEEE Software*, vol. 29, no. 6, 2012, pp. 18–21.

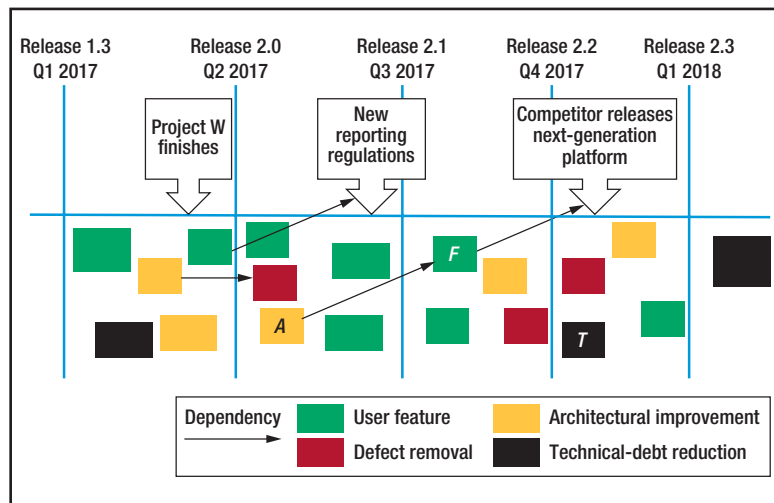


FIGURE A. An architecture roadmap. *F* is a crucial feature (a website's ability to check whether a hotel has free Wi-Fi), *A* is a change necessary before *F* can be implemented (an upgrade to a hotel communication platform), and *T* is the payment of technical debt through refactoring.

ers working on other solutions in the same interdependent system of systems. The evolution viewpoint helps the managers and product owners plan ahead. By acknowledging future events in the time dimension, it helps fellow workers who depend on your architecture, by telling them what aspects of the architecture will change (for example, the integration hub or interfaces) and when. The viewpoint

provides important input to project management tools such as risk registers and Gantt charts, and to product owners populating and prioritizing sprint backlogs in agile projects.

Experiences in Architecture Roadmapping

The approach we just described has been applied in parts of CGI since 2012. This roadmapping aims to find

the right balance between overanticipation and underanticipation in implementing architectural constructs. Architectural constructs are under-the-hood parts of a solution. They include things such as abstraction layers, firewalls, or caching mechanisms, which typically aren't visible to users but are needed to achieve quality attributes such as modifiability, security, or performance. Ove-

ranticipation typically manifests itself in architectural constructs that over time turn out to be less valuable than the trouble of creating them was worth. (This phenomenon is called YAGNI [you aren't gonna need it] in agile circles.⁸) Underanticipation often occurs when architectural constructs are implemented too late, causing the solution to accrue technical debt and making it increasingly difficult to add features in an evolutionary manner. Naturally, the time dimension is crucial to achieving the right amount of anticipation.


Our architects have been using architecture roadmapping in many contexts and domains, for time spans between one and six years ahead. Their anticipation documents are often quite informal, never called an evolution viewpoint but rather a “roadmap,” “decision support,” or a “strategy document.” Their architectures typically take into account these timed events:

- project or process milestones, such as delivery and approval deadlines, and deadlines in dependent projects;
- product version or infrastructure upgrades;
- business changes due to various reasons—for example, changes in external agreements (such as Key Performance Indicators), mergers or takeovers, or legislative or policy changes; and
- changes in the availability of resources, including the availability of expertise.

These anticipated events generally affect a combination of risk, cost, and business value. For example, a delivery deadline typically has impact in terms of the cost of delay and risk of losing customers. The introduction

of a product version upgrade could add business value by supporting new features. However, it could also represent a risk if the product's supplier discontinues support for a previous version (or changes the product's backward-compatibility policy).

Our architects reported significant benefits from making the time dimension part of their architecture in this way. The benefits mentioned most were improved (more realistic) stakeholder expectations and better prioritization of required architectural improvements. This is because architecture roadmapping helps architects articulate evolution scenarios' business impact. It also helps them discuss the timing of architectural improvements on the basis of that business impact, rather than on the basis of generic (and sometimes dogmatic) “rules” such as YAGNI or “Do not optimize prematurely.” Some of our architects also stressed the importance of stakeholder communication to identify anticipated events.

Documenting the time dimension part of your architecture might look like extra work. However, anticipation should be a large part of your job as an architect, anyway. If you communicate your anticipation as an evolution viewpoint or architecture roadmap, your architecture description will stay valid longer. And, you'll have a ready answer when stakeholders ask how you've addressed their change and planning concerns. 

References

1. ISO/IEC 42010:2011, *Systems and Software Engineering—Architecture Description*, ISO, 24 Nov. 2011; www.iso-architecture.org/42010.
2. E.R. Poort, “Driving Agile Architect- ing with Cost and Risk,” *IEEE Software*, vol. 31, no. 5, 2014, pp. 20–23.
3. P. Kruchten, “The 4+1 View Model of Architecture,” *IEEE Software*, vol. 12, no. 6, 1995, pp. 42–50.
4. N. Rozanski and E. Woods, *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*, 2nd ed., Addison-Wesley, 2011.
5. J. Schulenklopper and E. Rommes, “Why They Just Don't Get It: Communicating about Architecture with Business Stakeholders,” *IEEE Software*, vol. 33, no. 3, 2016, pp. 13–19.
6. P. Kruchten, R.L. Nord, and I. Ozkaya, “Technical Debt: From Metaphor to Theory and Practice,” *IEEE Software*, vol. 29, no. 6, 2012, pp. 18–21.
7. N. Brown, R.L. Nord, and I. Ozkaya, “Enabling Agility through Architecture,” *CrossTalk*, Nov./Dec. 2010, pp. 12–17.
8. M. Fowler, “Yagni,” blog, 26 May 2015; <http://martinfowler.com/bliki/Yagni.html>.

ELTJO POORT is a solution architect at CGI. Contact him at eltjo.poort@cgi.com.



Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.