Dr. Darius Silingas
(E-Mail: darius.silingas@nomagic.com)
is a Principal Consultant at No Magic, the vendor of award-winning UML tool MagicDraw®. Since 1998, he has participated in numerous international software development projects as a developer, analyst, architect, and consultant. He has run over 100 consultancy and training sessions on software modelling for software professionals in 18 countries, besides he has published a number of papers on modelling and model-driven development and spoken at conferences like OOP, JAX, Code Generation, Architecture and Design World, and EuroSTAR.

Using Simple Architecture Models to Coordinate Scrum Teams

Informal and ambiguous architecture documentation can lead to inefficient communication in software projects and cause delayed sprint targets. Architectural models specified in UML can help to improve that situation if kept simple and created with the stakeholders' concerns in mind.

Scrum and Architecture: Friends or Foes?

Scrum teams do work planning around epics, features, and user stories. The overall system architecture is typically not well captured and maintained, as the Scrum practitioners usually focus on coding and verbal communications inside a team. Any architectural discussions typically end up with hand-drawn informal pictures and boxes and lines in presentation slides. They are imprecise, ambiguous, and difficult to maintain, so typically Scrum practitioners end up with multiple versions of diagrams – all of them being out of date and unreliable. If you need precise information, you have to locate the "one who knows" and ask questions. In large projects, where multiple distributed Scrum teams work on a long-term assignment such as product line development, this quickly becomes a huge problem: the teams are overloaded with communication, they put a lot of effort on fixing small things that fall within the boundaries of one team responsibility, and the overall architecture view is lost. This in turn ends up in an unhealthy working environment, with constant delays of sprint targets and developers unwilling to "waste" any more time on communication. In such a context, it is necessary to create and maintain an architecture model, which serves as a contract between Scrum teams, helps to minimize communications load, and enables seeing the big picture and making strategic cross-team decisions, such as reuse of common architectural infrastructure or ensuring proper interoperability of features developed by different teams.

What Is Important to Model: Addressing Concerns of Stakeholders

Unified Modelling Language (UML®), which is the *de facto* standard in software industry, provides vast capabilities for modelling various aspects of a software system. It is very important to carefully choose a small set of modelling artefacts that are essential for improving architecture-related communications. For that

Stakeholder	Major Concern
Product Manager	Specify functionality of the product and what value it provides to a specific product stakeholder – end user, developer, administrator, etc.
Architect	Distribute functionality of the product into software components based on common architectural framework so that ease of development, reuse, portability, performance, and other qualities are maximized.
Developer	Write code that is compliant to a defined architecture and get good guidance on interfaces of external components to be used.
Tester	Design and execute test cases that represent realistic product usage scenarios, and address problematic issues such as feature interaction or potential conflicts in complex configuration settings.
Architecture Board	Define architecture evolution strategy; select architectural frameworks and technologies for addressing current issues, such as performance or feature portability between different operating systems.

Table 1: Software development stakeholders and their major concerns

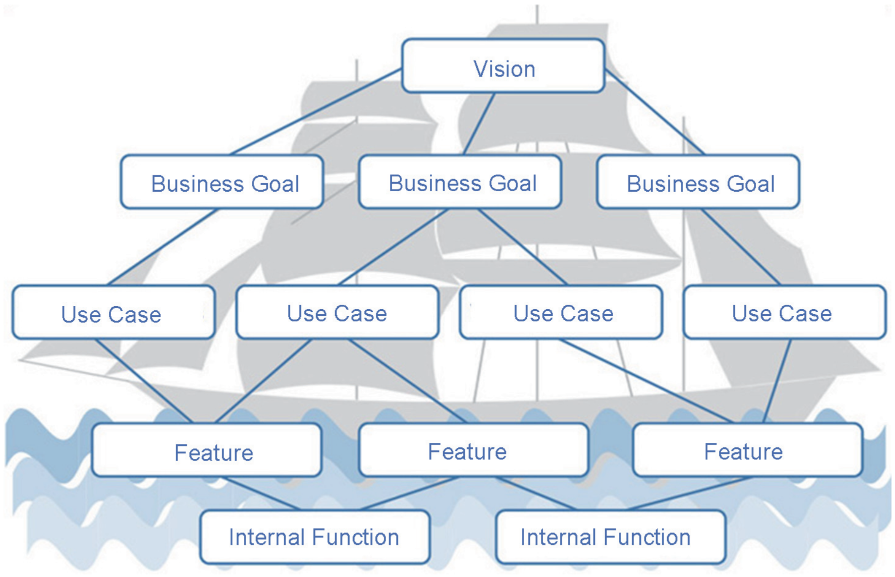

Figure 1: Use cases and features in a requirements ship model.

purpose, you need to understand the concerns of particular stakeholders such as a product manager, architectural board, architect, developer, or tester, and address them in modelling.

Functional Architecture: Use Cases Are Important

Although a Scrum product manager would specify epics, features, and user stories, it is important to understand how they are linked to use cases. In theory, a use case concept is closer to a user story concept, but in practice it is very often true that product managers start defining features that do not provide value for the end user "per se", but rather provide specific functions in one or more use cases. This is nicely shown in the requirements ship model introduced by Alistair Cockburn – the features are under the water, so they are not clearly visible to the stakeholders other than the developers who implement them.

We recommend that the use cases should be clearly identified (typically there are not so many of them) and mapped to the features – this helps to better understand the feature context, prioritize them properly, and define test cases for exploring features in a realistic use case–based context. Also, use cases provide a big picture of value that is provided by a system to the actors. This enables evolving a product in a direction of introducing new functionality, providing different value rather than gold plating existing features.

Architecture Strategy: Common Infrastructure Is Important

In large organizations, it is important to establish architecture strategy, which should be followed by all products/projects that are developed. One of the issues with distributed Scrum teams is that they focus on their sprint targets, but not so much on cross-team architecture. In cases of multiple homogenous systems, e.g. product lines, commonalities are not extracted and reused. This creates multiple problems – redundant implementations, inconsistent solutions for the same use case, products with different user and developer experience, poor portability, etc.

In order to solve these issues, it is necessary to clearly define infrastructure components such as frameworks or utility libraries that can be reused by multiple applications. Such a framework can also serve as a means for raising abstraction level and solving application portability issues. It is useful to capture such infrastructure in a model and maintain it, so that every architect or developer can refer to it for specific infrastructure details – this way, it helps to improve architecture communications.

Logical Architecture: Components, Dependencies, and Interfaces Are Important

When the common architecture infrastructure is established, the architects working on specific applications should focus on specifying the dependencies to the other components, and on the interfaces that this application provides for external usages. This information serves as a contract between development teams working on interacting applications and helps to minimize communications load.

An internal component structure is also important, but it is discussed only inside a development team where verbal communication is efficient enough – so this can be left out from the model and discussed on a source code basis. Keeping it simple and trying to fit into a "just enough" philosophy, the external component specification is much more important to specify precisely and maintain in the model.

What is Important to Model: Understanding the Architecture Views

The views to the architecture model will be dependent on specific stakeholder concerns. While the product manager will be mostly interested in use cases that are also very important to the tester, the architecture board will be willing to see what the components are and how they reuse the infrastructure. The model can be very helpful for keeping these views in sync by capturing their relationships and enforcing some

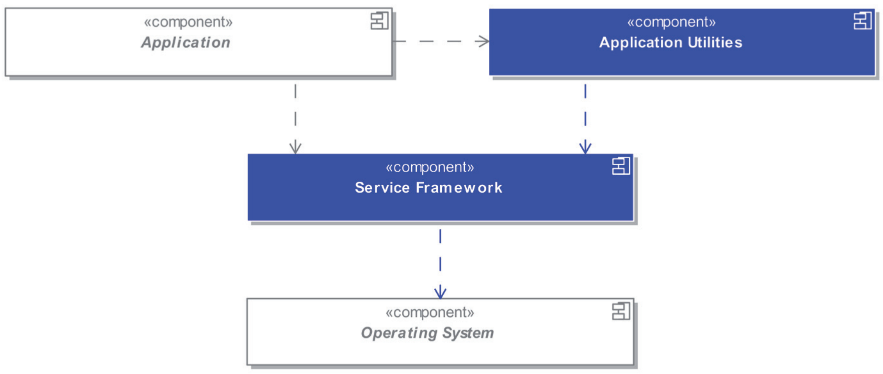

Figure 2: An example of Service Framework *and* Application *Utilities components as a common shared infrastructure for various applications and operating systems*

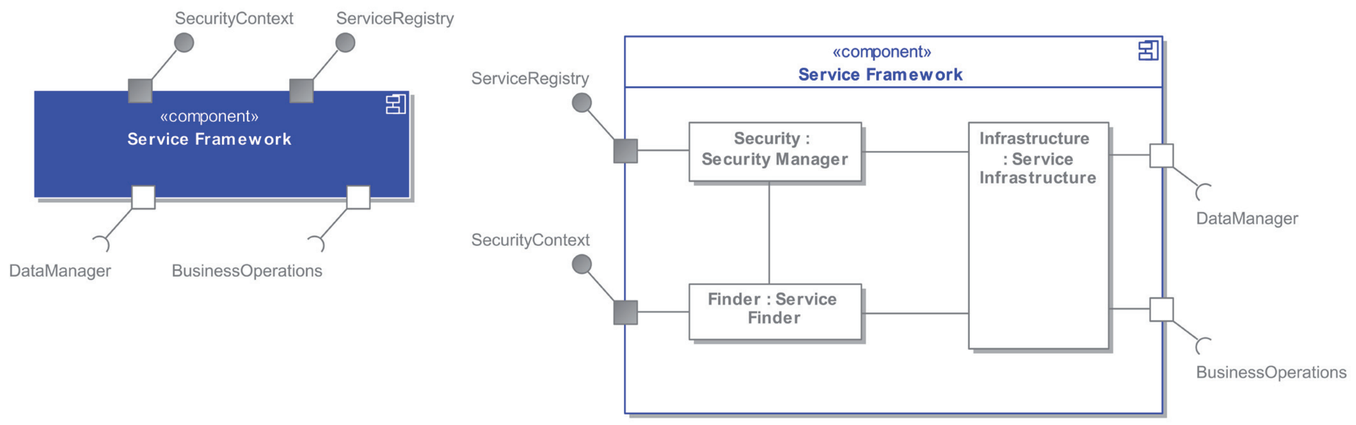

Figure 3: External vs. internal component specification

rules, such as tracing each use case to one or more components or avoiding circular dependencies between components.

A nice feature of a state-of-the-art UML modelling tool like MagicDraw® (download your trial copy at www.magicdraw.com) is to run reports on the model that provide different views based on report templates. This helps to maintain one model as a single consistent source of architecture with projections addressing concerns of different stakeholders, which is not possible with textual descriptions and informal drawings.

Conclusion

In large Scrum projects, modelling is an important activity that helps to synchronize the work of multiple teams, and improve communications and quality of developed software. Scrum practices (or their application in practice) are not sufficient for capturing and communicating architecture in a proper way. Therefore, a small set of modelling artefacts – such as actors, use cases, components with interfaces, and architecturally significant interactions – needs to be captured, documented, and maintained in UML models.

This will not take you extra time – these architecture aspects are discussed anyway, but their informal representations in drawings and texts are very difficult to reuse and maintain and do not help to improve architecture communications, which a proper modelling in UML will do. Also, you do not have to do it all – start with the stakeholders who suffer most from architecture-related problems and address the artefacts that they care about in "just enough" detail.

Feel free to contact us at training@nomagic.com if you want a more detailed presentation of how simple architecture models can help you to coordinate development teams in Scrum or other environments, or get advice for your specific situation. ■

Stakeholder	Use Cases	Infrastructure	Components	Ports and Interfaces	Internal Structure
Product Manager	+	-	-	-	-
Architect	+	+	+	+	-
Developer	-	+	+	+	+
Tester	+	-	+	-	-
Architecture Board	-	+	+	-	-

Table 2: Software development stakeholders and their interest in architecture artefacts