Configuring a Use Case Process

(Managing Requirements, Part 2)

Author's Note*: This article was written back in 1998. It describes an Agile process before the term Agile was coined. The use cases we used correspond to what we would call stories today. The detailed requirements were taken from existing source documents and translated directly to test cases. Later in the project we actually started using use cases to glue and trace the test cases back to business requirements.*

I am a strong proponent of use cases, yet as I pointed out in my last column,[1] I find that use cases are often misused. In this column I would like to address the **configuration** of the use case process, which I find to be equally misunderstood and misused.

An international corporation I am working with is developing a distributed, multi-currency, and multi-lingual, core financial system. In many ways the project exemplifies the best current practices for OO software development.

During the first six weeks we developed a comprehensive domain model. This required an extensive interview process with the company's CPAs, auditors, treasurers, controllers, and other financial personnel. The domain model is stored in a commercial CASE tool as an integrated set of seventeen UML class diagrams. To validate the domain model, a video was made of a walkthrough of the diagrams. The video was then distributed internationally to several hundred of the company's key financial personnel. The blown up versions (required for the video) of the domain model class diagrams were affixed to the walls in the project development area, and are frequently referenced and occasionally updated by the development team.

The overall software architecture (see figure 1) is built on fundamental OO principals and reuses a standard database architecture that is described in detail in Peter Heinckiens new book.[2] The database architecture maps from relational tables to objects in such a way that the business model is entirely separated from the database technology. This allows easy migration to other database technologies. A professional integrated Java development environment is being used to create the actual software, and commercial class libraries play an important role in reducing development effort. Experts in the field are consulted for important design decisions, and a technical issues resolution database is maintained. It details the pros and cons of the alternatives debated as well as the rationale for the design selected.

Seventeen system increments are defined in the project plan. As each increment is finished, it is tested on multiple platforms and with multiple commercial database management systems. A corporate steering committee was set up to oversee and facilitate the project. Beta test sites have been selected, and there is an excellent working relationship between the development team members and the financial personnel that will ultimately use the software system. When ambiguities in domain knowledge or application requirements arise, a financial expert is available to dialogue with the team within an hour or two. This is ideal. The financial personnel have a vested interest in the project and usually see any problem through to its resolution.

We are well into the development of this system, yet so far we have only defined eighteen use cases, and none of these have been elaborated into complete interaction dialogs or put into a standard UML detail use case template.

This lack of a complete set of detailed use cases is heretical to many of my colleagues. I agree that in most cases much more effort should be devoted to use case development, but not all projects require the same use case process.

---

[1] Timothy D. Korson. The Misuse of Use Cases. OBJECT MAGAZINE, May 1998, pp. 18-19. http://www.software-architects.com/publications/index.html

[2] Peter M. Heinckiens. BUILDING SCALABLE DATABASE APPLICATIONS, Addison-Wesley, MA, 1998.

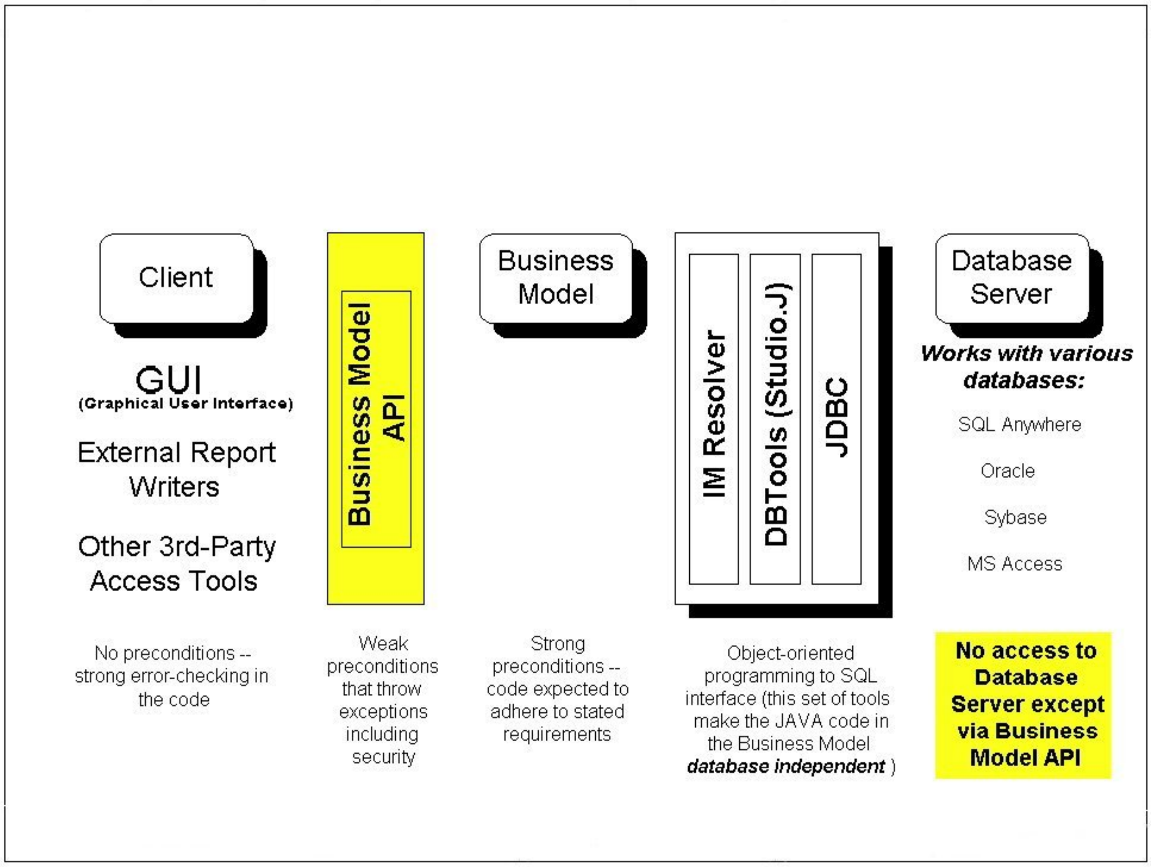
Client
GUI
(Graphical User Interface)
External Report Writers
Other 3rd-Party Access Tools
No preconditions -- strong error-checking in the code
Business Model API
Weak preconditions that throw exceptions including security
Business Model
Strong preconditions -- code expected to adhere to stated requirements
IM Resolver
DBTools (Studio.J)
JDBC
Object-oriented programming to SQL interface (this set of tools make the JAVA code in the Business Model database independent)
Database Server
Works with various databases:
SQL Anywhere
Oracle
Sybase
MS Access
No access to Database Server except via Business Model API

Figure 1

A Use Case Process Should be Configured for Each Specific Project

A full-blown use case process requires a complete set of hierarchically organized use cases, developed in sufficient detail so that the system test cases can be directly derived from the most detailed level of use cases. These use cases should be completely elaborated with alternative paths, preconditions, etc. A use case model should be developed that show the extends and uses relationships. For each use case there should be a sequence diagram that traces the use case to the implementation.

Large, multi-site, multi-team, projects often need a full-blown use case process, but not all projects fall into that category. There are several factors that affect the configuration of a use case process.

Multi-team. The larger the number of teams on a project, the greater the need for detail and formality in the use case process. Often use cases cross team boundaries. Well-written, detailed use cases can document certain aspects of team interactions and help avoid misunderstandings. At some of our consulting sites we have augmented the use case template and corresponding sequence diagram to explicitly show team boundaries. *The financial system described in the opening paragraphs of this column is being developed by a single team, with all team members located in a dedicated project work area*.

Multi-site. Multi-site projects compound the problems of multi-team development and make it even more necessary for all project documentation--including use cases--to be formal and detailed.

Specialized Teams. I favor cradle-to-grave integrated teams, but work with many companies that have separate teams for requirements analysis, design and development, and testing. Obviously when the requirements team is separate from the development team, the need for documentation detail is increased. *The financial system team is tightly integrated. The same individuals who are writing the Java code interviewed the accountants, created the domain model, and will test the system*.

Team Member Domain Knowledge. When I was a professor at Clemson University, I spent my summers as a visiting scientist at the Software Engineering Institute. I will always remember one particular speaker at the SEI monthly seminar series. He worked with radar signal processing, and the thesis of his presentation was that it is easier to teach a radar specialist how to program than it is to teach a programmer about radar technology. He gave several relevant and amusing anecdotes to illustrate his point. Most of his observations are difficult to refute--which is why I am so adamant that development teams participate in the domain analysis. The less a team knows about the domain, the more details must be spelled out in the use case model. *Many of the team members in the financial system example have degrees in accounting and have extensive financial backgrounds*.

Integration with the System Test Process. If the use case model is tightly integrated with the system test process, the cost of developing detailed use cases is repaid when the system test cases are developed. If a separate independent test team (which would have to develop test cases anyway) is involved in developing the use cases early in the project lifecycle, then many benefits accrue. In this case, the independent test team can be involved in domain model validation and other QA activities throughout the project lifecycle. *For the financial system there is no independent test team*.

Extent to Which the Requirements Already Exist in Some Other Form. If you are starting a large multi-site project for which there are few existing written requirements, I would recommend that the requirements gathering process include the development of a complete detailed set of use cases. If the project is a reengineering of an existing system, or if for some other reason the project already has an existing set of requirements, the team must decide what percentage of the requirements to rewrite as a hierarchically structured set of use cases. Many factors enter into this decision. I recommend always creating at least the top two to three levels of domain use cases to help drive validation of the domain model. When requirements already exist in some form, most teams choose to develop only the next few levels of use cases, resulting in 50 to 200 use cases. This is reasonable. In the presence of an existing detailed requirements document, the incremental value of each additional level of use cases decreases dramatically. An exception to this is when the use case process is tightly integrated with the system test processes, and the independent test team derive their test cases directly from the detailed use cases. *Business requirements for the financial system already exist in several large manuals, as well as in several legacy systems*.

Heavy-Weight Use Case Process	<	>	**Light-Weight Use Case Process**
Multiple sites			Single site
Multiple specialized teams			Single integrated development team
Development team has limited domain knowledge			Development team has extensive domain knowledge
Integrated testing process			No external testing group
Few preexisting written requirements			Extensive preexisting written requirements
Stable specifications			Rapidly changing interface specifications
Limited access to clients			Ready access to knowledgeable clients

Figure 2

Stability of the Requirements and Availability of the Clients. Uses cases must be kept accurate and updated as requirements change. If the requirements are fairly stable, the task of maintenance is manageable. However, if the requirements are subject to rapid change, the cost of updating thousands of use cases may be prohibitively high. In some instances, when the clients are readily available for input and rapid feedback, the team may elect to forgo written detail interface specifications in favor of a continuous client review process of actual project increments. *Our interface specifications are not well defined, but our clients are readily available for input*.

Many other factors, including the nature of the application, level of automation of the testing process, availability of resources, time to market, corporate culture, etc., should influence the configuration of a use case process. The fundamental point I wish to make is that one size does not fit all, yet most project managers seem to think that if they are going to use objects, there is one, and only one, right way to utilize use cases.

My experience leads me to believe in a use case spectrum (see figure 2). Your project is likely somewhere in the middle of the spectrum. Configure your use case process intelligently and then manage it carefully.