

## “Lunching” and Institutionalizing

Paul Clements

In 1999, a small group of us visited a company that manufactures software-intensive aircraft subsystems. We were interested in the software architecture(s) that they used for their systems, and how architecture played a role in their overall culture and development process. We interviewed over 20 people in six sessions over two days, including developers, senior designers, functional area managers, quality assurance/configuration management/tool groups, and systems designers. Although architecture was our quarry for this trip, we were delighted to discover a strong product line culture along the way.

The first interview session was with a group of senior designers with the title of "project engineer." Project engineers are the technical authorities for projects. They drew for us a detailed picture of the common architecture for their group of products, which we will give the collective (and, to protect confidentiality, fictitious) name of "Aircraft Systems," and which we were to learn was indeed a product line of systems at this company. The project engineers told us that projects<sup>1</sup> begin with attempts to reuse, with as little change as possible, the architectural components from previous projects. In fact, the process starts with the software requirements specification (SRS). The Aircraft Systems group maintains a "common" (what we would call a "core") SRS; system-specific SRSs are written as variations of this. They also told us that the project engineers meet weekly to work out common requirements that they can pass on to the groups who maintain the major subsystems and associated software.

We asked what prevented a developer from picking up a component from a previous project and changing it to meet the exigencies of the product he or she was working on at the time. The answer, they said, was that this was strongly discouraged. The functional area managers would not allow it, and the offending engineer would "have to eat by himself in the lunchroom." Beneath the levity of this comment, a picture of strong product line culture was beginning to emerge.

Subsequent interview groups confirmed this. We learned that their development environment (in this case, Rational's APEX) plays a critical role in the sustainment of the product line. For one thing, every project is open to every other project, so that people can straightforwardly search for components to reuse. Each component is stored with its history of usage, its own requirements, its test cases, and its Ada spec, so when retrieved for reuse its entire heritage and artifact accompaniment come with it. APEX reveals what components depend on what other components, and users are able to compare versions to quickly discern differences. And APEX automatically sends email to the functional area manager when a changed subsystem under that manager's purview is checked in. We were learning that the functional area managers are the guardians of the product line.

---

<sup>1</sup> By "project" we mean here the organizational unit or team that builds a product.

At this point, we did not know who these mystical functional area managers were, or what role they played. But they were clearly important to the product line, and commanded a lot of respect from the engineers and developers. We asked to interview them—they were not originally on our schedule because we did not know they existed—and they revealed the whole product line picture.

About four and a half years ago, this company faced a crisis: three major projects, which were expected to be won in a staggered fashion, came in simultaneously. These projects were to supply systems for three different kinds of United States military aircraft. They realized that they needed to exploit the commonality in all of the systems, and the product line was born. There was apparently no leader from above who ordered the architects into a locked closet and forcefully heeled the organization over to a new tack. Rather, the functional area managers simply began to meet (over lunch, we are told) to work out procedures for exploiting the commonality that they knew existed in previous projects and that would be required in the new ones. Projects apportioned the work and shared each other's components.

Today, the functional area managers control the design and evolution of the subsystems. They are like a core asset group, but unlike other core asset groups, this one has no implementation duties. Rather, the functional area managers are like senior designers who (a) approve or reject design changes requested by individual projects; (b) keep track of changes that are needed by more than one project; and (c) find projects that have the money to make the approved changes. (Sometimes changes are made using independent research and development money as well.) So this "core asset group" has the vision and the change authority, but changes are made by projects. Each of these managers has a team of people that "work" for them and are assigned to individual projects as needed. Each functional area team represents a collection of expertise about a particular functional area (subsystem). Thus, the organization is a matrix in which a developer works for both a functional area manager and (while assigned to a project) a project manager. See Figure 1.

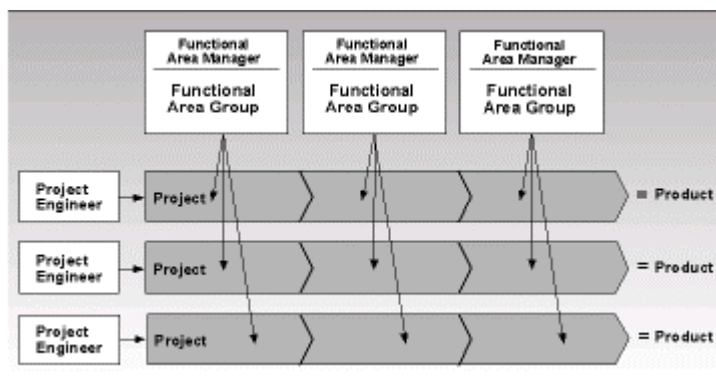


Figure 1. Organization by Functional Area Group

Improvements are continually made to the subsystems (which are their primary software core assets). The functional area managers are always looking for ways to improve the architecture, and then looking for projects that can use (and hence pay for the making of) those improvements. Examples include (a) moving away from direct subroutine invocation as the collaboration mechanism, and toward registration;

and (b) simplifying the pattern of dependencies (as manifested by Ada “with” statements) among the subsystems, which at the time of our visit was almost incomprehensibly complex.

Some of the conditions that work in this company’s favor include

- a relatively small number of projects so far. The number of versions of any subsystem that a reuser has to browse through is no more than 15-20, and 6 is a much more likely number. Thus, they don't yet have a runaway version-control or component-search problem.
- a domain in which the software architecture naturally mirrors the hardware/system architecture. Most software that they build matches up straightforwardly to electronic boxes that plug into the aircraft, boxes that the company has long manufactured. Other software that doesn't correspond precisely to a box falls out naturally and is treated separately.
- deep domain expertise, which is not a surprise.

Also not surprisingly, their architecture supports the product line view by providing changeability and portability qualities. It is strongly layered, and enforces strict visibility rules among the layers. Subsystems that are present in every product (such as a bus manager that handles putting messages on the communication bus and pulling them off for other subsystems) are distinguished from subsystems that may or may not be present in a particular product. Subsystems have internal structures of their own; the part that displays information is often changed, but the state-machine or controlling part seldom does, and these are well-separated parts of each subsystem. There is an input/output part of each subsystem that communicates with other subsystems via the bus. Overall, the architecture features minimal use of global data, maintains stable subsystem interfaces, and mandates consistent and disciplined interprocess communication protocols. Platform and environmental dependencies, scheduling policies, user interface, and message formats are all isolated (encapsulated) in separate software components.

This company clearly has a strong culture of product lines and architecture, and this emerged consistently. Everyone who drew the architecture drew the same picture. Everyone who told how the product line works told the same story. But the culture is an oral one. Nowhere was the product line operation codified in writing. Except for the SRSs, people rather than documents seemed to be the authoritative sources for much of the information necessary in development. But the oral culture has advantages. There is a strong and formal mentoring process in place, and it seems to be quite effective. There is also open communication. Developers are free to talk to their project engineers and advocate changes; they either convince them to make the change or are convinced by them that the change is not needed. The project engineers will not hesitate to lobby the functional area managers for changes. Communication flows freely.

And what does this company have to show for its product line efforts?

- They can port to a new processor in about eight days, and have already ported their software to a PowerPC.
- Reuse numbers as high as 94% were cited, and reuse in the 80% range seemed merely nominal for the groups to whom we spoke, earning only a modest shrug of the shoulders.

- Their tool/environment person (whose tools help collect metrics) estimated that what they used to do in 18-38 months they can now do in 8-16 months.

So necessity was once again the mother of invention, a product line scenario we have seen more than once. But under the calm guidance of the functional area managers, who saw what needed to be done and worked quickly and efficiently among themselves to do it, the transition to the product line paradigm was accomplished with a minimum of trauma. But besides starting the product line, the functional area managers have been its guardians, have kept it alive and nurtured it, have protected its conceptual integrity, and have instilled a strong sense of product line culture in the entire Aircraft Systems organization. They are its champions.

Not bad for an idea that started over lunch.

## About the Author

Dr. Paul Clements is a senior member of the technical staff at Carnegie Mellon University's Software Engineering Institute, where he has worked for 8 years leading or co-leading projects in software product line engineering and software architecture documentation and analysis.

Clements is the co-author of three practitioner-oriented books about software architecture: *Software Architecture in Practice* (1998, second edition due in late 2002), *Evaluating Software Architectures: Methods and Case Studies* (2001), and *Documenting Software Architectures: View and Beyond* (2002). He also co-wrote *Software Product Lines: Practices and Patterns* (2001), and was co-author and editor of *Constructing Superior Software* (1999). In addition, Clements has also authored dozens of papers in software engineering reflecting his long-standing interest in the design and specification of challenging software systems.

He received a B.S. in mathematical sciences in 1977 and an M.S. in computer science in 1980, both from the University of North Carolina at Chapel Hill. He received a Ph.D. in computer sciences from the University of Texas at Austin in 1994.

---

The views expressed in this article are the author's only and do not represent directly or imply any official position or view of the Software Engineering Institute or Carnegie Mellon University. This article is intended to stimulate further discussion about this topic.

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.