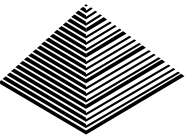
15



CelsiusTech

A Case Study in Product Line Development

with Lisa Brownsword

We trained hard, but it seemed that every time we were beginning to form up into teams, we would be reorganized. I was to learn later in life that we tend to meet any new situation by reorganizing; and a wonderful method it can be for creating the illusion of progress while producing confusion, inefficiency, and demoralization.
— Petronius Arbiter, 210 B.C.

This chapter relates the experience of CelsiusTech AB, a Swedish naval defense contractor that successfully adopted a product line approach to building complex software-intensive systems. Called Ship System 2000 (SS2000), their product line consists of shipboard command-and-control systems for Scandinavian, Middle Eastern, and South Pacific navies.

This case study illustrates the entire Architecture Business Cycle (ABC), but especially shows how a product line architecture led CelsiusTech to new business opportunities. Figure 15.1 shows the roles of the ABC stakeholders in the CelsiusTech experience.

Note: Lisa Brownsword is a member of the technical staff at the Software Engineering Institute, Carnegie Mellon University.

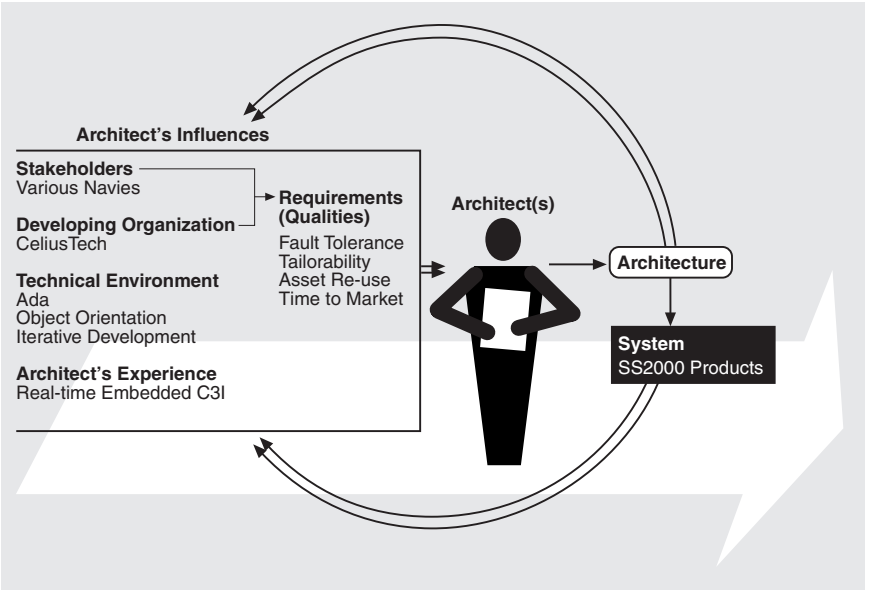


FIGURE 15.1 The ABC as applied to CelsiusTech

15.1 Relationship to the Architecture Business Cycle

CelsiusTech has long been known as a leading supplier of command-and-control systems within Sweden's largest, and one of Europe's leading, defense industry groups, which also includes Bofors, Kockums, FFV Aerotech, and Telub. At the time they developed the systems that are the subject of this chapter, CelsiusTech was composed of three companies: CelsiusTech Systems (advanced software systems), CelsiusTech Electronics (defense electronics), and CelsiusTech IT (information technology systems). It employed approximately 2,000 people and had annual sales of 300 million U.S. dollars. Their main site is near Stockholm, with subsidiaries located in Singapore, New Zealand, and Australia.

This study focuses on CelsiusTech Systems (CelsiusTech for short), whose focus includes command, control, and communication (C3) systems, fire control systems,[1] and electronic warfare systems for navy, army, and air force applications. The organization has undergone several changes in ownership and name since 1985 (see Figure 15.2). Originally Philips Elektronikindustrier AB, the division was sold to Bofors Electronics AB in 1989 and reorganized into NobelTech AB

[1] The term *fire control* refers to firing a gun at a moving target, from a platform that is itself moving with 6 degrees of freedom and flexing as well.

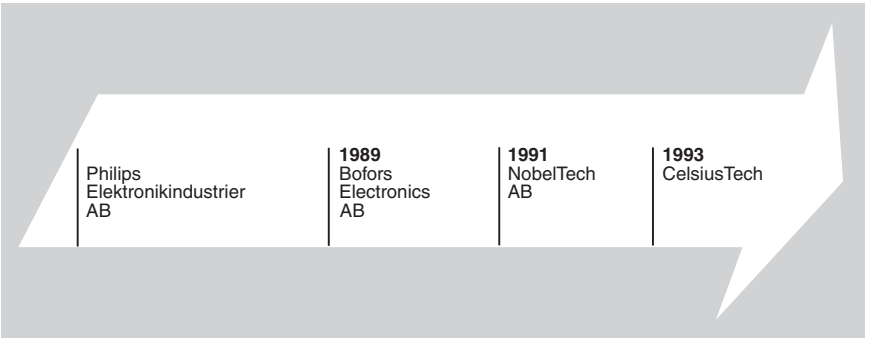


FIGURE 15.2 CelsiusTech Systems' corporate evolution

in 1991. It was purchased by CelsiusTech in 1993. Although senior management changed with each transaction, most of the mid- and lower-level management and the technical staff remained, thus providing continuity and stability.

THE SHIP SYSTEM 2000 NAVAL PRODUCT LINE

CelsiusTech's naval product line, known as Ship System 2000 (internally as Mk3), provides an integrated system that unifies all weapons, command-and-control, and communication systems on a warship. Typical system configurations include 1 million to 1.5 million lines of Ada code distributed on a local area network (LAN) with 30 to 70 microprocessors.

A wide variety of naval systems, both surface and submarine, have been or are being built from the same product line. These include the weapons, command-and-control, and communications portions of the following:

- Swedish Göteborg class coastal corvettes (KKV) (380 tons)
- Danish SF300 class multi-role patrol vessels (300 tons)
- Finnish Rauma class fast attack craft (FAC) (200 tons)
- Australian/New Zealand ANZAC frigates (3,225 tons)
- Danish Thetis class ocean patrol vessels (2,700 tons)
- Swedish Gotland class A19 submarines (1,330 tons)
- Pakistani Type 21 class frigates
- Republic of Oman patrol vessels
- Danish Niels Juel class corvettes

The Naval Systems division has sold more than 50 of its Mk3 naval systems to seven countries.

Figure 15.3 shows a Royal Swedish Navy multi-role corvette of the Göteborg class during a visit to Stockholm harbor. On top is the C/X-band antenna of the surveillance and target indication radar. Forward and aft of this, on top of the

FIGURE 15.3 Swedish multi-role Corvette of the Göteborg class featuring a CelsiusTech command-and-control system. Photo from Studio FJK; reproduced with permission.

superstructure, are the two fully equipped fire control radar and optronic directors from CelsiusTech.

Systems built from the product line vary greatly in size, function, and armaments. Each country requires its own operator displays on different hardware and in different presentation languages. Sensors and weapons systems, and their interfaces

to the software, also vary. Submarines have different requirements than surface vessels. Computers in the product line include 68020, 68040, RS/6000, and DEC Alpha platforms. Operating systems include OS2000 (a CelsiusTech product), IBM's AIX, POSIX, Digital's Ultrix, and others. The SS2000 product line supports this range of possible systems through a single architecture, a single core asset base, and a single organization.

ECONOMICS OF PRODUCT LINES: AN OVERVIEW OF CELSIUSTECH'S RESULTS

In this section we discuss CelsiusTech's results in building complex software-intense systems.

Shrinking Schedules. Figure 15.4 shows the status and schedules for later systems under development from the CelsiusTech product line. Ships A and B were contracted for at the same time and, as we will see, caused CelsiusTech to move to a product line approach. System A is the basis of the product line. Customer project A ran almost nine years, although functional releases were running on the designated ship by late 1989. System B, the second of the two original projects, required approximately seven years to complete and is similar to the previous non-product-line Mk2.5 system. It was built in parallel with system A, validating the product line. While neither system individually showed greater

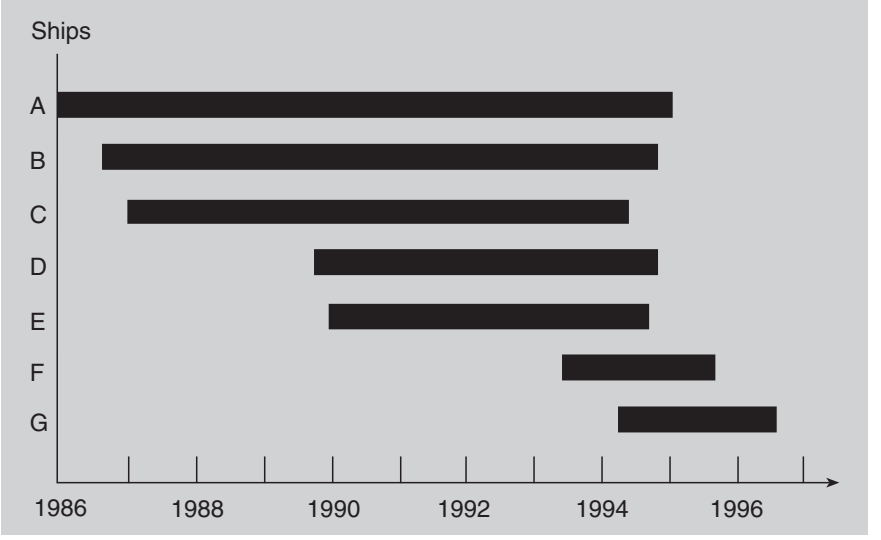


FIGURE 15.4 Product schedules

productivity, CelsiusTech was able to build both (and the product line) with roughly the same number of staff as for a single project.

Systems C and D were started after much of the product line existed, with a correspondingly shortened completion time. Systems E and F show a dramatic schedule reduction because they are fully leveraging the product line assets. CelsiusTech reports that the last three ship systems were all *predictably* on schedule.

Code Re-use. While the production schedules show time to market for a product, they do not indicate how well the systems use a common asset base. Figure 15.5 shows the degree of commonality across the CelsiusTech naval systems. On average, 70% to 80% consist of elements used verbatim (i.e., checked out of a configuration control library and inserted without code modification).

Using Core Assets to Expand the Business Area. CelsiusTech has expanded its business into a related area that takes advantage of the architecture and other core assets that were originally developed for naval uses. STRIC, a new air defense system for the Swedish Air Force, embraces the abstraction that a ground platform is a ship whose location does not change very often and whose pitch and roll are constantly zero. Because of the flexibility (amenability to change) of the SS2000 architecture and product line, CelsiusTech was able to quickly build STRIC, lifting 40% of its elements directly from the SS2000 asset base. (See the sidebar Mastering Abstraction at CelsiusTech.) This demonstrates one of the feedback links in the ABC: The existence of the SS2000 product line and its architecture enabled new business opportunities.

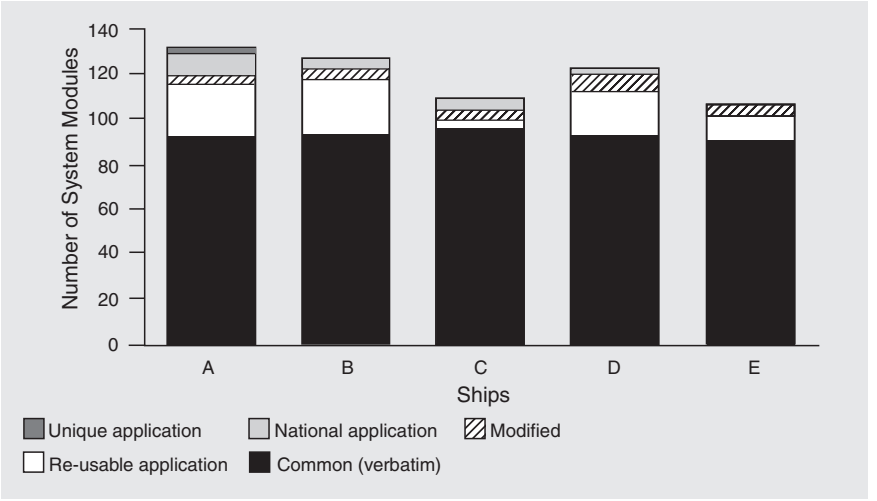


FIGURE 15.5 Commonality across CelsiusTech naval systems

Mastering Abstraction at CelsiusTech

Studying software product lines holds two particular fascinations for me. The first is discovering what all successful product line organizations have in common. Early in our study of product lines, the list of what I thought was required to be successful was fairly long. As we discovered and analyzed more examples, each brought with it a new dimension of experience and each seemed to whittle away at my list of organizational must-haves.

One common aspect still remains, however, and that is a product line mindset. A successful product line organization considers its business to be the care, nurturing, and growth of its software product line, *singular*, particularly its core asset base. This is in stark contrast to immature or unsuccessful product line organizations that see their enterprise as churning out a set of products, *plural*, that have some things in common.

The distinction is subtle but palpable. A product-based organization talks about its products and subjugates long-term product line goals to satisfy short-term product deadlines. Such an organization will, for example, reward workers for heroic measures to get a product out the door, even if it means performing a late night clone-and-own on a core asset. In contrast, a product-line-based organization talks about the product line and its health almost as if individual members of the family are coincidental byproducts. This singular mindset helps a product-line-based organization make strategic moves quickly and nimbly and as a unified whole.

The second fascination for me is what a successful product line organization can do, at the enterprise level, with this powerful capability. With an innate understanding of the product line's scope—that is, an articulated definition of what systems are within the product line's capability to build and what systems are not—an enterprise can make a conscious decision to "drive" its capability around the neighborhood and pick up business in nearby underutilized markets.

At CelsiusTech, both of these points were made eloquently by a cartoon I saw on a developer's bulletin board during our visit to gather information for this chapter. I wish I had asked for a photocopy of it, but it looked something like this:

At about the time of our visit, CelsiusTech had announced a new project to build a product line of air defense systems—that is, ground-based anti-aircraft guns. The company estimated that on the day they made the announcement, fully 40% of the new product family was in place because it was based on Ship System 2000.

The developer's cartoon made the point that an air defense system is just a simplified ship on land that does not pitch or roll much and stays stationary most of the time. It told me that the CelsiusTech staff had a firm grasp of the concept of abstraction, but it also told me that they had the product line mindset. This cartoon was not about the production of an air defense system but a celebration of what their beloved product line was about to become. It succinctly depicted the company's enterprise-level foray into a whole new business area via its product line capability. That cartoon was a magnificent exhibition of one organization's product line sophistication.

— PCC

WHAT MOTIVATED CELSIUSTECH?

To understand why CelsiusTech made the decision to develop a product line and what actions were required, it is important to know where it began. Prior to 1986, the company developed more than 100 systems, in 25 configurations, ranging in size from 30,000 to 700,000 source lines of code (SLOC) in the fire control domain.

From 1975 to 1980, CelsiusTech shifted its technology base from analog to 16-bit digital, creating the so-called Mk2 systems. These tended to be small, real-time, and embedded. The company progressively expanded both system functionality and expertise with real-time applications in the course of building and delivering 15 systems.

From 1980 to 1985, customer requirements were shifting toward the integration of fire control and weapons with command and control, thus increasing the size and complexity of delivered systems. The Mk2 architecture was expanded to provide for multiple autonomous processing nodes on point-to-point links, resulting in Mk2.5. Mk2.5 systems were substantially larger, in both delivered code (up to 700,000 SLOC) and number of developers (300 engineer-years over 7 years).

Conventional development approaches were used for Mk2.5. These had served the company well on the smaller Mk2 systems, but difficulties in predictable and timely integration, cost overruns, and schedule slippage resulted. Such experiences were painful, but they were important lessons for CelsiusTech. The company gained useful experience in the elementary distribution of real-time processes onto autonomous links and in the use of a high-level, real-time programming language (in this case, the Pascal-like RTL/2). Figure 15.6 shows the systems built by CelsiusTech prior to 1985.

In 1985, a defining event for CelsiusTech (then Philips) occurred. The company was awarded two major contracts simultaneously—one for the Swedish Navy

	1970–1980: Mk2 Systems	1980–1985: Mk2.5 Systems
Kind of System	Real-time embedded fire control; Assembly language and RTL/2	Real-time embedded C3; RTL/2
Size	30–100K SLOC	700K SLOC; 300 engineer-years over 7 years
Platforms	Analog and 16-bit digital systems	Multi-processors, minicomputers, point-to-point links

FIGURE 15.6 Systems built by CelsiusTech prior to 1985

and one for the Danish Navy. Requirements for both ships indicated the need for systems larger and more sophisticated than the Mk2.5s, which had suffered from schedule and budget difficulties. Having to build two even larger systems, let alone in parallel, presented management and senior technical staff with a severe dilemma. Clearly, the development technologies and practices applied on the Mk2.5 system would not be sufficient to produce the new systems with any reasonable certainty of schedule, cost, and required functionality. Staffing requirements alone would have been prohibitive.

This situation provided the genesis of a new business strategy that recognized the potential *business* opportunity of selling and building a series, or family, of related systems rather than some number of specific systems. Thus began the SS2000 product line. Another business driver was the recognition of a 20- to 30-year lifespan for naval systems. During that time, changes in threat requirements and in technology advances would have to be addressed. The more flexible and extensible the product line, the greater the business opportunities. These business drivers or requirements forged the technical strategy.

The technical strategy would need to provide a flexible and robust set of building blocks to populate the product line from which new systems could be assembled with relative ease. As new system requirements arose, new building blocks could be added to the product line to sustain its business viability.

In defining the technical strategy, an assessment of the Mk2.5 technology infrastructure indicated serious limitations. A strategic decision was made to create a new-generation system (the Mk3) that would include new hardware and software and a new supporting development approach. This would serve as the infrastructure for new systems development for the next decade or two.

EVERYTHING WAS NEW

CelsiusTech's decision to convert its business strategy to a product line approach coincided with a time of high technology flux. This meant that, to implement the technical strategy for the SS2000 product line, virtually all aspects of the hardware, the software, and development support would have to change. Thus, the hardware shifted from VAX/VMS minicomputers to Motorola 68000-series microcomputers. Whereas the Mk2.5 systems consisted of a small number of processors on point-to-point links, the SS2000 products have a large number of highly distributed processors with fault-tolerant requirements. The software life-cycle approach shifted from RTL/2-based structured analysis/design and waterfall development to Ada83 with more object-based and iterative development processes. Development support migrated from custom, locally created and maintained development tools to a large, commercially supplied environment. The major technical differences are summarized in Figure 15.7.

ANALYSIS OF THE BUSINESS CONTEXT

The CelsiusTech experience reveals several factors that played an important role in the establishment of the SS2000 product line, some of which were advantages, some inhibitors.

Ownership Changes. While it is routine to buy, sell, and restructure companies, the impact on an organization attempting to adopt significantly different

Prior to 1986 Previous Technical Infrastructure	1986 New Technical Infrastructure
• Minicomputers	• Microcomputers
• Few processors on point-to-point links	• Many processors on commercial LAN
• No fault tolerance	• Fault tolerant, redundant
• RTL/2	• Ada83
• Waterfall life cycle, early attempts at incremental development	• Prototyping, iterative, incremental development
• Structured analysis and design	• Domain analysis, object-based analysis and design
• Locally developed support tools	• Rational development environment

FIGURE 15.7 Changing technical infrastructures

business and technical strategies can be devastating. Typically, management changes associated with company ownership transactions are sufficient to stop any transition or improvement efforts under way (as observed by Petronius Arbiter over two millennia ago). That this did not happen at CelsiusTech can be attributed either to strong and far-sighted top management or to top management's preoccupation with the other issues. Since CelsiusTech changed hands several times during this period, the latter explanation is more likely. It is clear that middle management had a strong commitment to a product line and were allowed to proceed unfettered by top management, who might otherwise have been hesitant to approve the necessary upfront investments. Normally a reorganization disrupts the entire organization. In the CelsiusTech case, the effects of the reorganizations and changes of ownership were buffered by middle management.

Necessity. The award of two major naval contracts in 1986, ostensibly a reason for celebration, was regarded as a crisis by CelsiusTech. Management immediately realized that they had neither the technical means nor the personnel resources to simultaneously pursue two large development efforts, each pioneering new technologies and application areas. Since all CelsiusTech contracts are fixed price, large-scale failure meant large-scale disaster. Indeed, less challenging systems had been over budget, past schedule, hard to integrate, and impossible to predict.

CelsiusTech was driven to the product line approach by circumstances; they were compelled to attempt it because their viability was clearly at stake. The fact that this period was also one of major technological change made it much more difficult to separate the costs associated with product line changes from those associated with adopting new technology.

Technology Changes. In 1986, all the chosen technologies were immature, with limited use in large industrial settings. Big, real-time, distributed systems making extensive use of Ada tasks and generics were envisioned but at the time were unprecedented. Moreover, object-based development for Ada was still a theoretical discussion. From 1986 to 1989, then, CelsiusTech was coping with the following:

- The maturation of technologies, such as Ada and object technology
- The maturation of supporting technology, such as networking and distribution
- The maturation of infrastructure technology, such as development environments and tools to assist in the automation of development processes
- The learning curve of the company, both technical and managerial, in the use of new technologies and processes inherent in the product line approach
- The learning curve of customers, who did not fully understand the contractual, technical, and business approaches of product lines
- The management of similar requirements across several customers

These maturation issues significantly increased the time required to create the product line. An organization making the same development paradigm shift

today would be in a much better position, with microcomputers, networks, portable operating systems, open systems standards, object-based development methods, Ada (or other programming languages appropriate to the domain and platforms), performance engineering, distributed systems technology, real-time operating systems, real-time analysis tools, large-project support environments, and large-project process assistants. These technologies are all either mature or at least usable and readily available. Also, much more is known about building and fielding software product lines (see Chapter 14). CelsiusTech estimates that up to one-third of its initial technology investment was spent building assets that can now be purchased commercially.

CELSIUSTECH'S ORGANIZATIONAL STRUCTURE

CelsiusTech's organizational structure and practices did not remain constant over the ten-year period covered here, but migrated through several distinct structures. The kind of knowledge and skills required of the staff also changed.

Project Organization Prior to 1986. The naval command-and-control system (Mk2.5) development was headed by a project manager who used the services of various functional areas, such as weapons or C3, to develop major segments of system capability. Figure 15.8 shows the organizational structure for the Mk2.5 project. Each functional area (command-and-control, tracking, etc.) was led by a project manager who had direct authority for staff resources and for all system development activities through release to system integration.

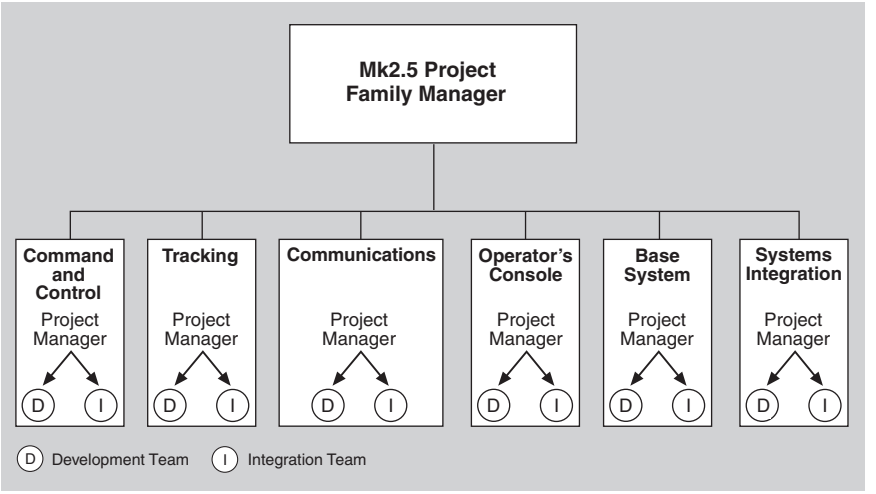


FIGURE 15.8 Mk2.5 project organization, 1980–1985

CelsiusTech found that this compartmentalized arrangement fostered a mode of development characterized by the following:

- Assignment of major system segments to their respective functional areas as part of system analysis
- Requirements and interfaces allocated and described in documents, with limited communication across functional area boundaries, resulting in individual interpretation of requirements and interfaces throughout design, implementation, and test
- Interface incompatibilities typically not discovered until system integration, resulting in time wasted in assigning responsibility and protracted, difficult integration and installation
- Functional area managers with little understanding of areas other than their own
- Functional area managers with limited incentives to work as a team to resolve program-level issues

SS2000 Organization Late 1986 to 1991. With the introduction of the SS2000 product line in late 1986 came a number of organizational changes from the Mk2.5 project organization. Figure 15.9 shows the organizational structure CelsiusTech used from late 1986 until 1991. A general program manager designated to lead

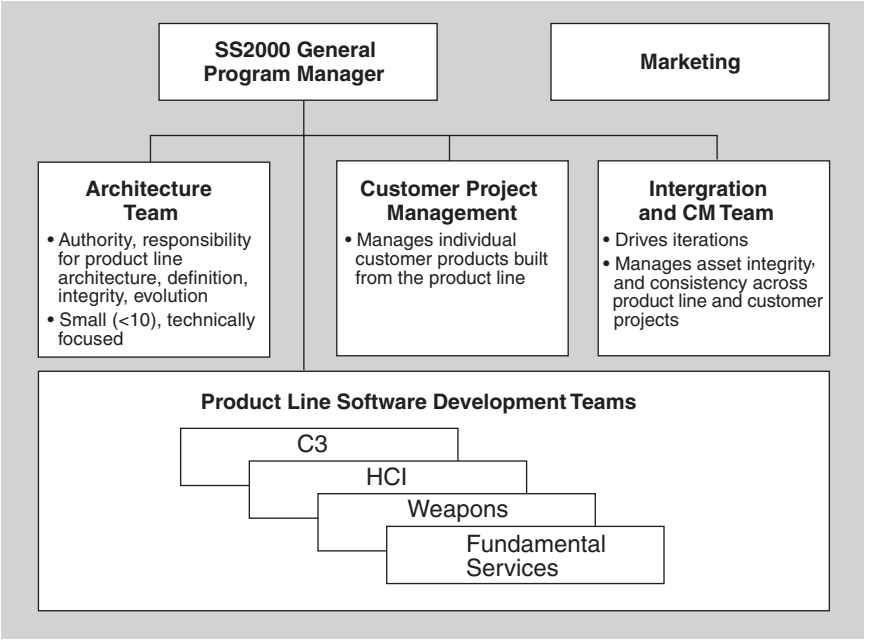


FIGURE 15.9 SS2000 organization, 1987–1991

the program was responsible for both creation of the product line and delivery of customer systems built from it. CelsiusTech sought to remedy the problems associated with the compartmentalized structure of the past by creating a strong management team focused on product line development as a company asset rather than on "empire building." To this end, functional area managers now reported directly to the general program manager. Developers were assigned to functional areas—weapons, C3, or human–computer interface (HCI), common services (used by the functional areas), and the interface to the various hardware and operating systems (called the Base System).

A small, technically focused architecture team with total ownership and control was created, reporting directly to the general program manager. CelsiusTech determined that the success of a product line hinged on a stable yet flexible architecture, one that was visible throughout the organization and vested with authority from the highest levels of management. In this way, the company reorganized itself to take advantage of the ABC: Architecture had to be at the heart of their new approach, and the architecture in turn changed important aspects of the organization.

The coordinated definition and management of multiple releases was central to the creation of a product line. To better support their release management, CelsiusTech combined software system integration and configuration management into a new group, reporting directly to the general program manager. Both the architecture team and the integration–configuration management group were novel approaches for CelsiusTech and were instrumental in the creation of the SS2000 product line.

The architecture team was responsible for the initial development and continued ownership and control of the product line architecture. This ensured design consistency and design interpretation across *all* functional areas. Specifically, the architecture team had responsibility and authority for the following:

- Creation of product line concepts and principles
- Identification of layers and their exported interfaces
- Interface definition, integrity, and controlled evolution
- Allocation of system functions to layers
- Identification of common mechanisms or services
- Definition, prototyping, and enforcement of common mechanisms such as error handling and interprocess communication protocols
- Communication to the project staff of the product line concepts and principles

The first iteration of the architecture was produced in two weeks by two senior engineers with extensive domain experience. It remains the framework for the existing product line, containing organizing concepts, layer definition, identification of approximately 125 system functions (out of the current 200 or so) and their allocation to specified layers, and the principal distribution and communication mechanisms. After completion of the first iteration, the architecture team took on the lead designers from each of the functional areas. The full team, now

comprising ten senior engineers, continued to expand and refine the architecture. This was in sharp contrast to the past, when functional area leaders had autonomy for the design and interfaces for their respective areas.

The combined integration and configuration management team was responsible for the following:

- Development of test strategies, plans, and test cases beyond unit test
- Coordination of all test runs
- Development of incremental build schedules (in conjunction with the architecture team)
- Integration and release of valid subsystems
- Configuration management of development and release libraries
- Creation of the software delivery medium

SS2000 Organization 1992 to 1998. From 1992 to 1994, CelsiusTech's emphasis increasingly shifted from the *development* of the architecture and product line elements to the *composition* of new customer systems from the product line. This trend increased the size and responsibilities of the customer project management group. CelsiusTech modified its organizational structure to assign the development staff to one of the following:

- *Component projects that develop, integrate, and manage product line elements.* The production was distributed across component project areas consisting of the functional areas (weapons, C3, and HCI), common services, and the operating system and network elements. Component project managers were rotated regularly, providing middle management with a broader understanding of the product line. The elements were provided to the customer projects.
- *Customer projects responsible for all financial, scheduling and planning, and requirements analysis through system integration/test/delivery.* Each customer system built from the product line was assigned a project manager responsible for all interactions and negotiations with the customer.

As CelsiusTech completed the basic product line and gained further experience using it, it looked for more efficient ways to produce systems and evolve the product line to take advantage of newer technology and changing customer needs. This was a feedback effect of the ABC, where the architecture caused the organization to continually reinvent itself, resulting in the organizational structure shown in Figure 15.10.

Each major application domain (naval and air defense) became its own business unit with its own manager. Each business unit had a marketing group, a proposal group, a customer projects group, and a systems definition group. The business unit was responsible for its software elements and its customer project managers. Each unit's operations were guided by a Marketing Plan, a Product Plan, and a Technical–Architecture Plan. The marketing group was responsible for the Marketing Plan that assessed the opportunities and value of each market

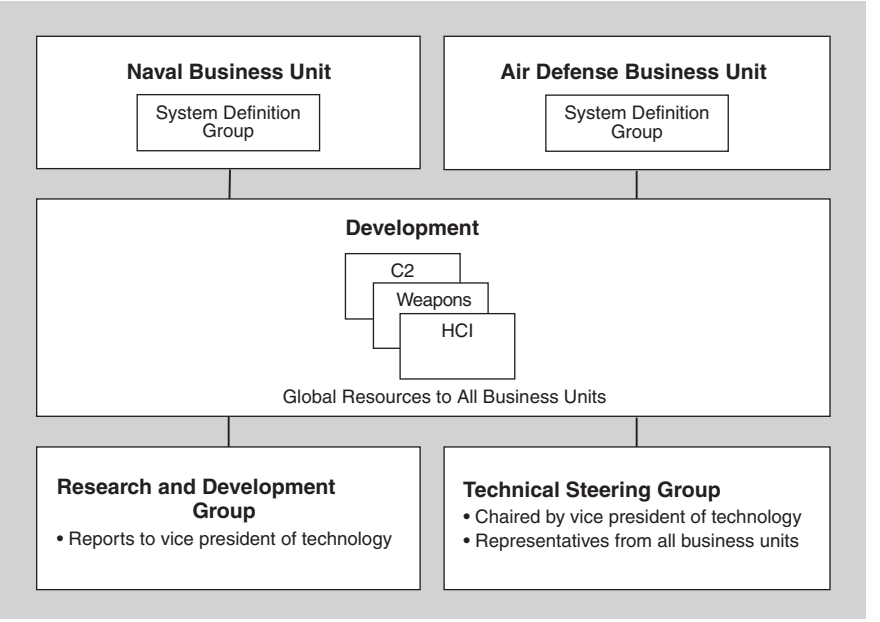


FIGURE 15.10 SS2000 organization, 1992–1998

segment. The Product Plan described the products the business unit sold and was owned by the proposal group. The Product Plan implemented the Marketing Plan. The system definition group was responsible for the Technical–Architecture Plan for their business unit. In turn the Technical–Architecture Plan implemented the Product Plan, outlining the evolution of the business unit's architecture. New project proposals took into account the business unit's Product and Technical–Architecture Plans. This approach kept the projects aligned with the product line.

Modules were supplied by the Development Group. Any customer-specific tailoring or development was managed from the business unit customer project using Development Group resources. The business unit's Systems Definition Group was responsible for the architecture. It owned and controlled the evolution of the architecture and major interfaces and mechanisms. For the Naval Business Unit, the Systems Definition Group was small (typically six members), consisting of senior engineers with extensive knowledge of the naval product line. It was responsible for overall arbitration of customer requirements and their impact on the product line.

The Naval Business Unit sponsored an SS2000 Product Line Users Group to serve as a forum for shared customer experiences with the product line and to provide direction for its evolution. The Users Group included representatives from all SS2000 customers.

The Development Group provided developer resources to all business units. Integration, configuration management, and quality assurance were also Development Group resources, matrixed to the business units as required. To further optimize creation of new systems from the product line, a Basic SS2000 Configuration Project was formed to create a basic, preintegrated core configuration of approximately 500K SLOC, complete with documentation and test cases that would become the nucleus of a new customer system.

The Technical Steering Group (TSG) was responsible for identifying, evaluating, and piloting potential new technology beneficial to any of CelsiusTech's business areas. It was headed by the vice president of technology and staffed by senior technical personnel from the naval and air defense business units, the Development Group, and the R&D Group. The TSG ensured that each Systems Definition Group created and evolved its architecture and technology plan.

Staffing Late 1986 to 1991. As shown in Figure 15.11, the project staffing levels ranged from an initial 20 to 30 to a peak of more than 200, with an average of 150. During the early stages of the program, while product line concepts and architecture were being defined, CelsiusTech found the staff levels too high. There was confusion among developers because concepts and approaches were in a state of flux.

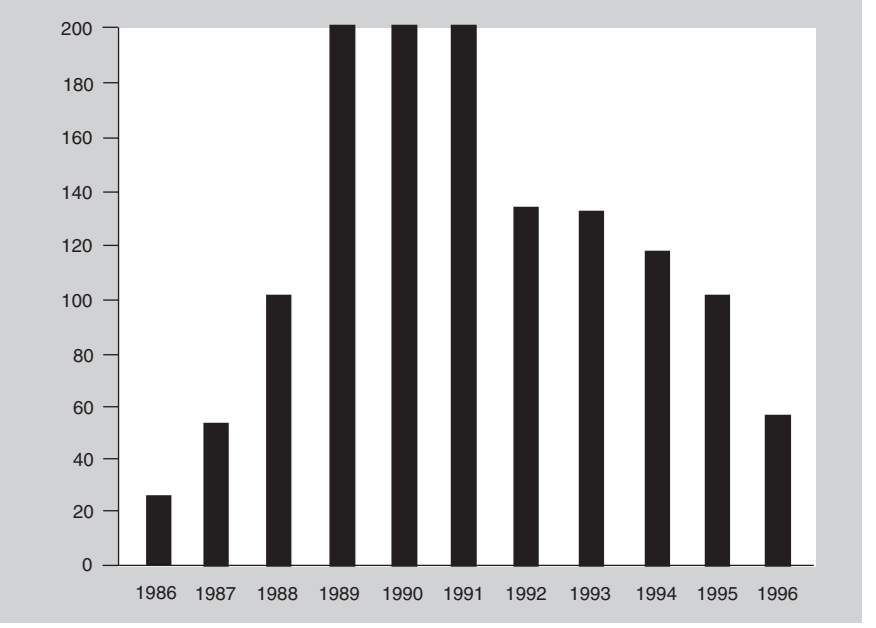


FIGURE 15.11 Approximate software staff profiles

The architecture team was responsible for creating the framework for the product line. Team members needed solid domain and customer knowledge combined with engineering skills and an ability to find relevant common mechanisms or product line elements. Communication and teaming skills were also mandatory. Developers needed to understand the framework, the building codes, and how their respective modules should fit. During the product line's formative period, the development staff required skills in the use of Ada, object-based design, and their software development environment, including the target testbed. In addition, they had to have broad knowledge of product line concepts, SS2000 architecture and mechanisms, creation of re-usable modules, incremental integration, and at least one functional area domain.

With much of the necessary technology immature, the management team (and senior technical staff) was operating largely on faith in the achievement of a shared ultimate capability. A key focus of their responsibilities included "selling" the business need and the desired future state to their teams.

Organizations that attempt to install immature technology encounter resistance as inevitable problems arise. Key to sustaining the early phases of such installations is strong, solutions-oriented management. At CelsiusTech, the general program manager focused on finding solutions rather than finding fault with the various immature technologies, their suppliers, or the development team. Managed experimentation was encouraged, not penalized, and technical innovations were supported. The general program manager thus became a role model for other managers.

During the formative years of the product line, managers were required to have extensive knowledge of product line concepts and business motivation. In addition, they needed strong skills in planning, communication, and innovative problem solving.

Management also had to cope with the inevitable discontent and resistance associated with a new business paradigm and its attendant technology. Substantial effort was made to help personnel understand the new business strategy and rationale. People who did not subscribe to or could not grasp the product line approach either left the company or found assignments on maintenance or other projects. This caused a loss of domain knowledge that took time to regain.

Staffing 1992 to 1998. By the end of 1991, four customer systems were under way, and a sufficient number of re-usable modules not only existed but had been delivered as part of the original two systems. The core of the product line was maturing rapidly so that, rather than all new modules, systems were now routinely composed from existing modules. Designers were needed less and were reassigned to other projects within the company. Howerer, with the increase in parallel customer projects, more integrators were needed, although the average of three to five per customer system remained steady. Because of the increasing number of projects during this period, the number of management staff did not decrease.

From 1994 to 1998, the staffing profile continued to change. As the product line and its use matured, CelsiusTech used fewer designers, developers, and integrators for the two latest customer systems in that period. Ever fewer designers were needed, potentially moving between business units. The downward trend was most notable in integration, given that CelsiusTech budgeted for an integration staff of one or two per system. Continuing system composition optimizations, such as the Basic SS2000 Configuration project, were expected to further reduce development-related staff levels. With the continued increase in parallel customer projects, the number of management staff remained constant.

With greater emphasis on the *composition* of systems from the product line, developers needed stronger domain and SS2000 knowledge than during product line creation. The use of Ada, object technology, and their development environment had become more routine. The integration group's focus turned to the integration and release management of many parallel systems. Increasing emphasis was placed on re-using test plans and data sets across customer systems.

The architecture team needed to maintain a solid knowledge of the product line and factor in growing current and approaching customer needs. Communication with customer project managers (for negotiation of multiple customer needs) and developers (desiring to optimize major interfaces and mechanisms) continued to be extremely important. Engineering skill to balance new needs yet preserve overall architectural integrity was vital for team members as they continually evolved the architecture and its major interfaces and mechanisms. The architecture team was involved in technical evaluations, prototype development of new interfaces (both for the external user and for application developers), and assessing the impact of the new technologies on the product line.

Less emphasis on technology maturation and training was required of management as more of the product line became available. With a larger set of customer systems existing, coordination of changing customer requirements across multiple customers emerged as a major management priority. Requirements negotiation involved not only customers but also other customer project managers and the product line architecture team. Customer project managers required increasing skill in negotiation and greater knowledge of the existing and anticipated future directions of the product line.

15.2 Requirements and Qualities

For new products to be derived from an organizational repository, they must be structured so that they can share modules. As we discussed in Chapter 14, this means that there must be a standard set of modules, with agreements about individual module's responsibility, behavior, performance, interface, locality of function, communication and coordination mechanisms, and other properties. This

familywide structure, the modules it comprises, and the properties about each that are constant across all members of the product line constitute the *product line architecture*.

As we have seen throughout this book, the primary purpose of an architecture is to acheive a system that meets its behavioral and quality requirements. The architecture for each SS2000 product line member was no exception. The most important of these requirements were:

- *Performance*. Command-and-control systems must respond in real time to continuously arriving sensor inputs and be able to control weapons under tight deadlines.
- *Modifiability*. The architecture needs to be robust with respect to computing platforms, operating systems, addition or replacement of sensor and weapon systems, human–computer interface requirements, communication protocols, and the like.
- *Safety, reliability, and availability*. The system must be available when needed, provide the correct data and commands to weapon systems, and fire only under the correct conditions.
- *Testability*. Each system must be integrable and testable so that errors (if any) are quickly found, isolated, and corrected.

Besides these single-system requirements, the SS2000 architecture carried the additional burden of application to an entire class of systems. Thus its requirements included the ability to replace one module with another tailored to a particular system without disrupting the rest of the architecture.

OPERATING ENVIRONMENT AND PHYSICAL ARCHITECTURE

The requirements of modern shipboard systems influence design solutions in profound ways. Sensors and weapons systems are deployed all over the ship; crew members interact with them via a multitude of separately housed workstations. The HCI must be highly tuned to facilitate rapid information flow and command acceptance and must be tailored to the operational mission of the vessel and the cultural idiosyncrasies of its crew. The likelihood of component failure dictates a fault-tolerant design.

Figure 15.12 is a physical view of a typical system. A redundant LAN is the communications backbone, connecting from 30 to 70 different, cooperating processors. Nodes on the LAN can total around 30. A *node* is the end of a communication run and may correspond to a crew station, a weapons platform, or a sensor suite, all located in various parts of the ship and widely dispersed. It may host up to six processors. The LAN is a dual Ethernet. Device-interface modules send and receive data to and from the system's peripherals, primarily sensors, and the weapons systems being controlled.

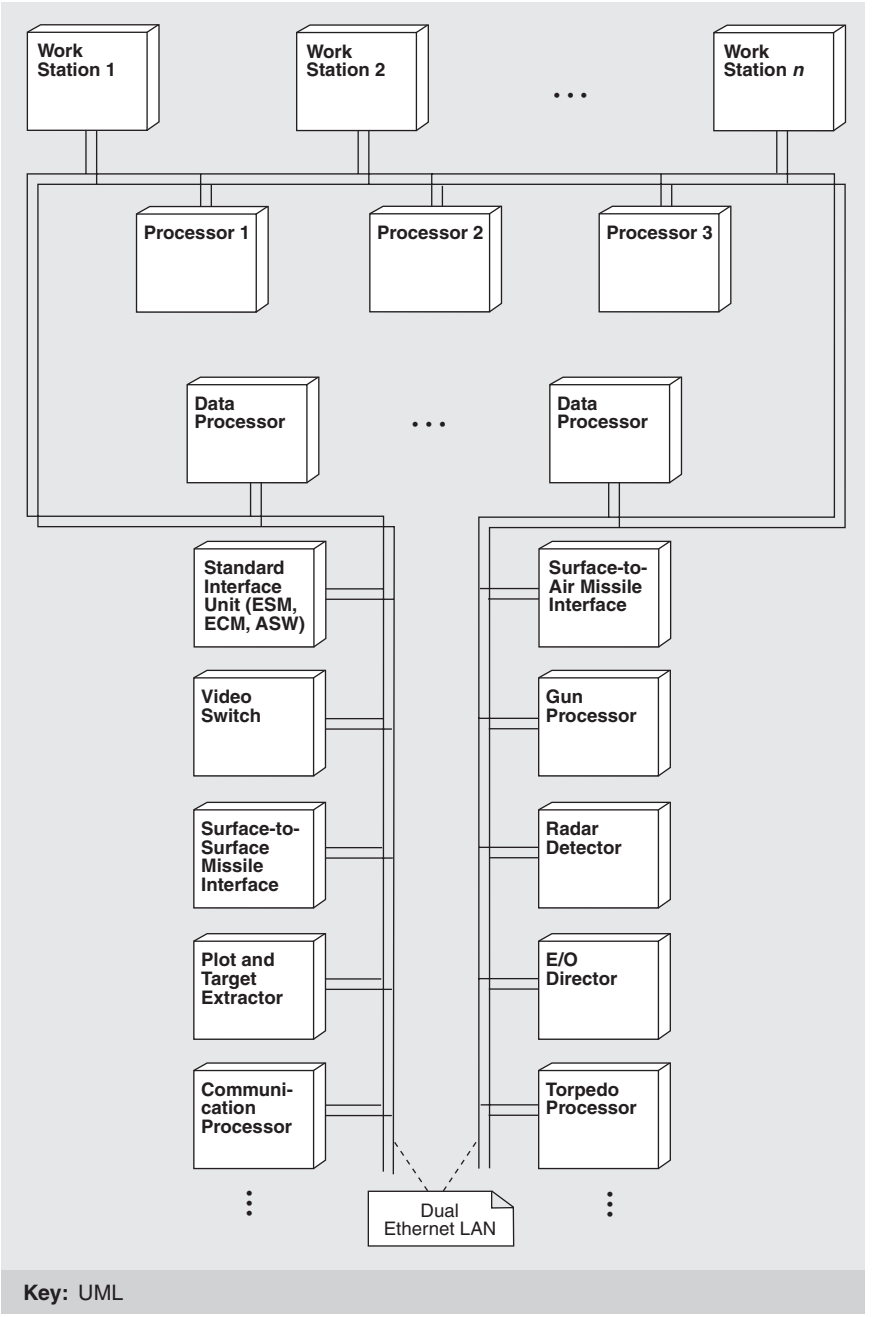


FIGURE 15.12 Typical physical architecture of an SS2000 product

15.3 Architectural Solution

We describe the architecture using three views—the process view so that we can explain how distribution was accomplished; the layered view as a basis for discussing how Ship System 2000 achieves a separation of concerns; and a module decomposition view to show assignment of responsibilities to different large-scale elements of the system, called *system functions* and *system function groups*. After presenting the architecture in terms of these three views, we discuss some of the issues that arose at CelsiusTech that are specific to the maintenance and use of a product line.

THE PROCESS VIEW: MEETING REQUIREMENTS FOR DISTRIBUTION AND PRODUCT LINE SUPPORT

Each CPU runs a set of Ada programs; each Ada program runs on at most one processor. A program may consist of several Ada tasks. Systems in the SS2000 product line can consist of up to 300 Ada programs.

The requirement to run on a distributed computing platform has broad implications for the software architecture. First, it necessitates building the system as a set of communicating processes, bringing the process view into play. Having a process view at all means that the performance tactic "introduce concurrency" has been applied. Distributed systems also raise issues of deadlock avoidance, communication protocols, fault tolerance in the case of a failed processor or communications link, network management and saturation avoidance, and performance concerns for coordination among tasks. A number of conventions are used to support the distribution. These respond to the distributed requirements of the architecture as well as its product line aspects. The tasks and intercomponent conventions include the following:

- Communication among components is by the passing of strongly typed messages. The abstract data type and the manipulating programs are provided by the component passing the message. Strong typing allows compile-time elimination of whole classes of errors. The message as the primary interface mechanism between components allows components to be written independently of each other's (changeable) implementation details with respect to data representation.
- Inter-process communication is the protocol for data transport between Ada applications that supports location independence, allowing communication between applications regardless of their residence on particular processors. This "anonymity of processor assignment" allows processes to be migrated across processors, for pre-runtime performance tuning and runtime reconfiguration as an approach to fault tolerance, with no accompanying change in source code.
- Ada task facilities are used to implement the threading model.

A producer of data does its job without knowing who the consumer of that data is. Data maintenance and update are conceptually separate from data usage. This is an application of the tactic "introduce an intermediary" to achieve modifiability, which the designers accomplished using a blackboard pattern. The main consumer of the data is the HCI component. The component that contains the repository is called the common object manager (COOB).

Figure 15.13 illustrates the role of the COOB at runtime. It shows not only the data flow that uses the COOB but also the data flows that bypass the COOB for reasons of performance. Track information (the positional history of a target), carried in a large data structure, is passed directly between producer and consumer, as is trackball information because of its very high update frequency.

Data-producing conventions include the following:

- Data is sent only when altered. This prevents unnecessary message traffic from entering the network.
- Data is presented as object-oriented abstractions in order to insulate programs from changing implementations. Strong typing allows compile-time detection of variable misuse errors.
- Components own the data they alter and supply access procedures that act as monitors. This eliminates race conditions because each piece of data is accessed directly only by the component that owns it.
- Data is accessible to all interested parties at all nodes in a system. Assignment to a particular node does not affect the data a component can access.
- Data is distributed so that response time to a retrieval request is short.
- Data is kept consistent within the system over the long term. Short-term inconsistencies are tolerable.

Network-related conventions include the following:

- Network load is kept low by design—that is, considerable design effort goes into managing the data flow on the network, ensuring that only essential information is transmitted.

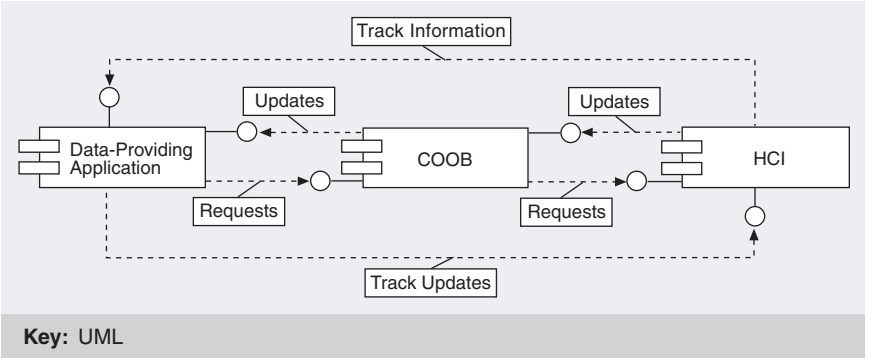


FIGURE 15.13 Using (and bypassing) the COOB

- Data channels are error resistant. Applications resolve errors internally as much as possible.
- It is acceptable for an application to "miss" an occasional data update. For instance, because a ship's position changes continuously, a position update may be missed but interpolated from surrounding updates.

Miscellaneous conventions include the following:

- Heavy use is made of Ada generics for re-usability.
- Ada standard exception protocols are used.

Many of these conventions (particularly those regarding abstract data types, IPC, message passing, and data ownership) allow a module to be written independently of many changeable aspects over which it has no control. In other words, the modules are more general and hence more directly usable in different systems.

THE LAYERED VIEW

The architecture for SS2000 is layered, as follows:

- The grouping of modules is roughly based on the type of information they encapsulate. Modules that must be modified if hardware platform, underlying LAN, or internode communication protocols are changed form one layer. Modules that implement functionality common to all members of the family form another. Modules specific to a particular customer product form a layer also.
- The layers are ordered, with hardware-dependent layers at one end of the relation and application-specific layers at the other.
- The layering is "strict," meaning that interactions among layers are restricted. A module residing in one layer can only access modules in its own or the next lower layer.

In SS2000, the bottom layer is known as Base System 2000; it provides an interface between operating system, hardware, and network on the one hand and application programs on the other. To applications programmers, Base System 2000 provides a programming interface with which they can perform intercomponent communication and interaction without being sensitive to the particular underlying computing platforms, network topologies, allocation of functions to processors, and so on. Figure 15.14 illustrates the architectural layers of SS2000.

THE MODULE DECOMPOSITION VIEW: SYSTEM FUNCTIONS AND SYSTEM FUNCTION GROUPS

As we mentioned in Chapter 2, an organization often has its own terms for the modules it introduces in a module decomposition view. CelsiusTech's modules were called system functions and system function groups.

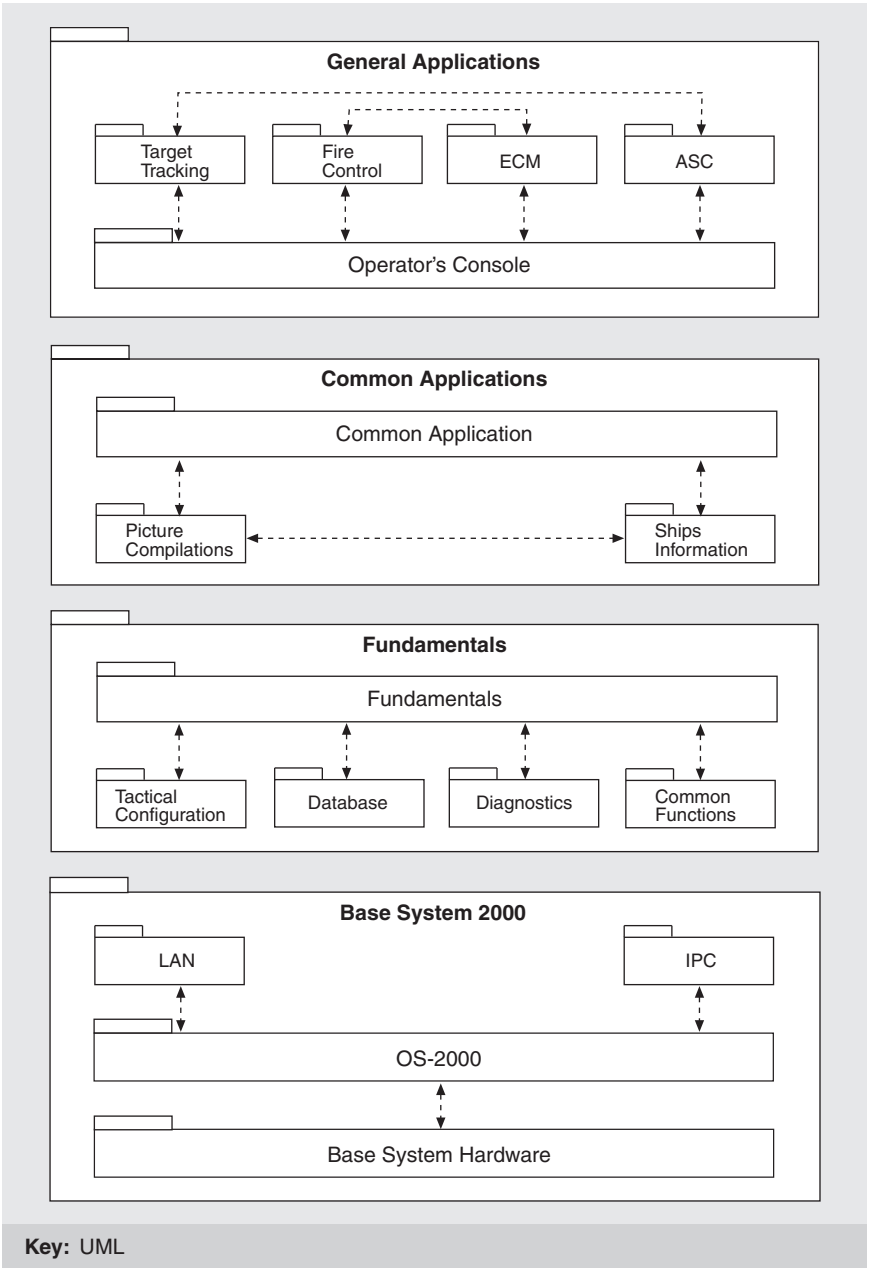


FIGURE 15.14 SS2000 layered software architecture

System functions are the primary element of module decomposition in SS2000. A system function is a collection of software that implements a logically connected set of requirements. It is composed of a number of Ada code units. A *system function group* comprises a set of system functions and forms the basic work assignment for a development team. SS2000 consists of about 30 system function groups, each comprising up to 20 or so system functions. They are clustered around major functional areas, including the following:

- Command, control, and communications
- Weapons control
- Fundamentals—facilities for intrasystem communication and interfacing with the computing environment
- Human–computer interface

Figure 15.15 illustrates the relationship between the various module types.

System function groups may (and do) contain system functions of more than one layer. They correspond to bigger pieces of functionality that are more appropriately developed by a large team. For example, a separate software requirements specification is written for each system function group.

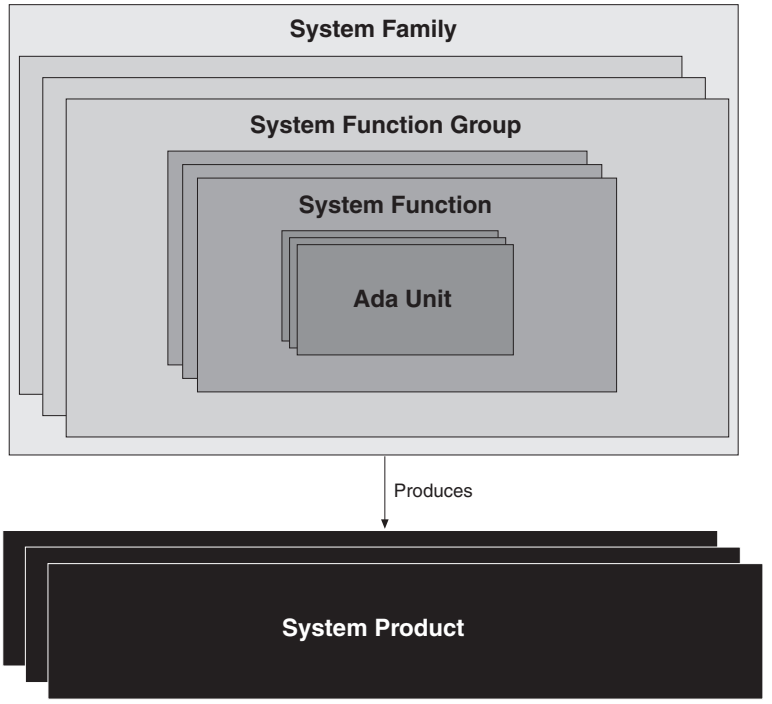


FIGURE 15.15 Units of software in the module decomposition view

System functions and system function groups, not the Ada code units, are the basic units of test and integration for the product line. This is crucial because it allows a new member of the product line to be treated as the composition of a few dozen high-quality, high-confidence modules that interact in controlled, predictable ways, as opposed to thousands of small units that must be regression-tested with each change. Assembly of large pretested elements was a key to making re-use work at CelsiusTech.

APPLYING THE SS2000 ARCHITECTURE

Table 15.2 summarizes the architectural goals for SS2000 and the approaches and tactics (from Chapter 5) used to achieve them. This section concludes the

TABLE 15.2 SS2000 Requirements and How the Architecture Achieved Them

Requirement	How Achieved	Related Tactic(s)
Performance	Strict network traffic protocols; software is written as a set of processes to maximize concurrency and written to be location independent, allowing for relocation to tune performance; COOB is by-passed for high-data-volume transactions; otherwise, data sent only when altered and distributed so response times are short	Introduce concurrency Reduce demand Multiple copies Increase resources
Reliability, Availability, and Safety	Redundant LAN; fault-tolerant software; standard Ada exception protocols; software written to be location independent and hence can be migrated in case of failure; strict ownership of data prevents multi-writer race conditions	Exceptions Active redundancy State resynchronization Transactions
Modifiability (including ability to produce new members of the SS2000 family)	Strict use of message-based communication provides interface isolated from implementation details; software written to be location independent; layering provides portability across platforms, network topologies, IPC protocols, etc.; data producers and consumers unaware of each other because of COOB; heavy use of Ada generics; heavy use of element parameterization; system functions and system function groups provide semantic coherence	Semantic coherence Anticipate expected changes Generalize modules Abstract common services Interface stability Intermediary Configuration files Component replacement Adherence to defined protocols
Testability	Interfaces using strongly typed messages push a whole class of errors to compile time; strict data ownership, semantic coherence of elements, and strong interface definitions simplify discovery of responsibility	Separate interface from implementation

presentation of the architecture by discussing four important issues that arose in building and maintaining the architecture and in building a family of systems from it.

Architecture as the Foundation. Although this case study emphasizes that technical solutions in a product line are insufficient without taking into account business and organizational issues as well, it remains a fact that the SS2000 architecture was the means for achieving a product line. Toward this end, abstraction and layering were vital. Abstraction allowed creation of modules that encapsulated changeable decisions within the boundaries of their interfaces. When a module is used in multiple products, the changeable decisions are instantiated whenever possible by parameterization. When the modules change across time as new requirements are accommodated, the changeable decisions held inside the module ensure that wholesale changes to the asset base are not needed.

The size and complexity of this architecture and the modules that populate it make clear that a thorough understanding of the application domain is required if a system is to be partitioned into modules that can be developed independently, are appropriate for a product line whose products are as widely varied as those in SS2000, and can accommodate evolution with ease.

Maintaining the Asset Base as New Systems Are Produced. As we discussed, the enduring product at CelsiusTech is not an individual ship for a specific customer, or even the set of systems deployed so far. Rather, the central task is viewed as maintaing the *product line itself*. Maintaining the product line means maintaining the re-usable assets in such a way that any previous member of the product line can be regenerated (they change and evolve and grow, after all, as their requirements change) and future members can be built. In a sense, maintaining the product line means maintaining a *capability*, the *capability* to produce products from the assets. Maintaining this capability means keeping re-usable modules up to date and general. No product is allowed to evolve in isolation from the product line. This is one approach to solving the problem, which we identified in Chapter 14, of keeping the evolution of the product line synchronized with the evolution of the variants.

Not every module is used in every member of the product line. Cryptologic and human interface requirements differ so widely across nationalities, for instance, that it makes more sense to build modules that are used in a few systems than to attempt a more general solution. In a sense, this yields product lines within the major product line: a Swedish set of products, a Danish set of products, and so on. Some modules are used only once but even these are maintained as product line assets, designed and built to be configurable and flexible, in case a new product is developed that can make use of them.

Externally, CelsiusTech builds ship systems. Internally, they evolve and grow a common asset base that provides the capability to turn out ship systems. This mentality—which is what it is—might sound subtle, but it manifests itself in the

configuration control policies, the organization of the enterprise, and the way that new products are marketed.

Maintaining Large Pre-integrated Chunks. In the classic literature on software re-use repositories, the unit of re-use is typically either a small fine-grained module (such as an Ada package, a subroutine, or an object) or a large-scale independently executing subsystem (such as a tool or a commercial standalone product). In the former case, the small modules must be assembled, integrated, configured, and tested after checking out; in the latter case, the subsystems are typically not very configurable or flexible.

CelsiusTech took an intermediate approach. Their unit of re-use is a system function, a thread of related functionality that comprises modules from different layers in the architecture. System functions are pre-integrated—that is, the modules they comprise have been assembled, compiled together, tested individually, and tested as a unit. When the system function is checked out of the asset repository, it is ready for use. In this way, CelsiusTech is not only re-using modules but also re-using the integration and test effort that would otherwise have to be repeated for each application.

Parameterized modules. Although modules are re-used with no change in code in most cases, they are not always re-used entirely without change. Many of the elements are written with symbolic values in place of absolute quantities that may change from system to system. For example, a computation within some module may be a function of how many processors there are; however, that number need not be known when the module is written; therefore, the module may be written with the number of processors as a symbolic value—a parameter—the value of which is bound as the system is integrated. The module works correctly at runtime but can be used without code change in another version of the system that features a different number of processors.

Parameters are a simple, effective, and time-honored means to achieve module re-use. However, in practice they tend to multiply at an alarming rate. Almost any module can be made more general via parameterization. The modules for SS2000 feature 3,000 to 5,000 parameters that must be individually tuned for each customer system built from the product line. CelsiusTech had no way to tell that a certain combination of parameter values, when instantiated into a running system, would not lead to some sort of illegal operating state.

The fact that there were so many parameters undermined some of the benefits gained from treating large system functions and system function groups as the basic units of test and integration. As parameters are tuned for a new version of the system, they in fact produce a version that has never been tested. Moreover, each combination of parameter values may theoretically take the system into operating states that have never been experienced, let alone exhaustively tested.

Only a small proportion of the possible parameter combinations will ever occur. However, there is a danger that unwillingness to "try out" a new parameter

combination could inhibit exploiting the built-in flexibility (configurability) of the elements.

In practice, the multitude of parameters seems to be mostly a bookkeeping worry; there has never been any incorrect operation that could be traced back solely to a set of parameter specifications. Often, a large module is imported with its parameter set unchanged from its previous utilization.

15.4 Summary

Between 1986 and 1998 CelsiusTech evolved from a defense contractor providing custom-engineered point solutions to essentially a vendor of commercial off-the-shelf naval systems. They found the old ways of organizational structure and management insufficient to support the emerging business model. They also found that achieving and sustaining an effective product line was not simply a matter of the right software and system architecture, development environment, hardware, or network. Organizational structure, management practices, and staffing characteristics were also dramatically affected.

The architecture served as the foundation of the approach, both technically and culturally. In some sense, it became the tangible thing whose creation and instantiation were the ultimate goal. Because of its importance, the architecture was highly visible. A small, elite architecture team had the authority as well as the responsibility for it. As a consequence, the architecture achieved the "conceptual integrity" cited by [Brooks 95] as the key to any quality software venture.

Defining the architecture was only the first step in building a foundation for a long-term development effort. Validation through prototyping and early use was also essential. When deficiencies were uncovered, the architecture had to evolve in a smooth, controlled manner throughout initial development and beyond. To manage this natural evolution, CelsiusTech's integration and architecture teams worked together to prevent any designer or design team from changing critical interfaces without the architecture team's explicit approval.

This approach had the full support of project management, and it worked because of the architecture team's authority. The team was a centralized design authority that could not be circumvented, which meant that conceptual integrity was maintained.

The organization necessary to create a product line is different from that needed to sustain and evolve it. Management needs to plan for changing personnel, management, training, and organizational needs. Architects with extensive domain knowledge and engineering skill are vital to the creation of viable product lines. Domain experts remain in demand as new products are envisioned and product line evolution is managed.

CelsiusTech's turnaround from one-at-a-time systems to a product line involved education and training on the part of management and technicians. All of these are what we mean by the return cycle of the ABC.

15.5 For Further Reading

There are two reports about CelsiusTech's conversion to a product line. One is from the Software Engineering Institute [Brownsword 96] and is the basis for this chapter. The other is a thesis from Sweden's Linkoping University [Cederling 92].

15.6 Discussion Questions

1. Could the CelsiusTech architecture have been used for the air traffic control system of Chapter 6? Could CelsiusTech have used that architecture? What are the essential differences?
2. CelsiusTech changed management structures several times during its development of the SS2000. Consider the implications of these changes, given our recommendation in Chapter 7 that product structure should mirror project structure.