Technical Report
CMU/SEI-96-TR-016
ESC-TR-96-016

A Case Study in Successful Product Line Development

Lisa Brownsword
Paul Clements

October 1996

Technical Report
CMU/SEI-96-TR-016
ESC-TR-96-016
October 1996

A Case Study in Successful Product Line Development

Lisa Brownsword
Paul Clements

Product Line Systems Program

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Table of Contents

Acknowledgments **vii**

1 Introduction **1**

1.1 Purpose and Background 1

1.2 Product Lines and Reuse 3

1.3 Why Did We Visit CelsiusTech? 5

1.4 Gathering Information 6

1.5 Report Structure 6

2 CelsiusTech Systems **9**

2.1 Background 9

2.2 The Ship System 2000 Naval Product Line 10

2.3 Impetus for a Product Line Approach 12

2.4 What Was New? Everything! 14

2.5 Analysis 14

3 Architectural Foundations for the SS2000 Product Line **19**

3.1 Requirements for an Architecture 19

3.2 Operating Environment 19

3.3 Physical Architecture 20

3.4 Software Architecture 20

3.5 Parameterized Components 27

3.6 Analysis 28

4 Organizational Factors **33**

4.1 Previous Project Organization 33

4.2 SS2000 Organization, 1986 to 1991 34

4.3 SS2000 Organization 1992 to 1994 36

4.4 SS2000 Organization Since 1994 37

4.5 Staffing Characteristics During 1986 to 1991 38

4.6 Staffing Characteristics for 1992 to 1994 40

4.7 Staffing Characteristics Since 1994 41

4.8 Analysis 42

5 Processes, Methods, and Tools **45**

5.1 Software Development and Processes Before the Product Line Approach 45

5.2 Development Approaches, 1986 to 1990 45

5.3 Development Approaches, 1990 to 1994 51

5.4 Development Approaches Since 1994 57

5.5 Analysis 58

6 Managing the Learning Curves 61
6.1 Managing the Learning Curve for Managers 61
6.2 Managing the Learning Curve for Technicians 62
6.3 Managing the Learning Curve for the Customer 64
6.4 Analysis 65

7 Summary Results 67
7.1 Stages of Product Line Development 67
7.2 Substantial Initial Investment Required 68
7.3 The Payoff 69
7.4 Architecture Was the Foundation 69
7.5 Organizational Issues 70
7.6 CelsiusTech and the Domain Engineering / Application Engineering Life-Cycle Model 70
7.7 Conclusion 72

Appendix A SS2000 Documentation Set 73

Appendix B Correspondence to DoD-STD 2167A Activities 75

Appendix C SS2000 Ada Practicum 77

Appendix D SS2000 Re-Education Curriculum Description 79
D.1 Curriculum Components 79
D.2 Typical Usage 81

References 83

List of Figures

Figure 1:	CelsiusTech Systems' Corporate Evolution	9
Figure 2:	Product Schedules: From Start to Final Delivery	11
Figure 3:	Reuse Across SS2000 Product Line	12
Figure 4:	Changing Technical Infrastructures	15
Figure 5:	Typical Physical Architecture of a Ship System 2000 Product	21
Figure 6:	Units of Software Components, Adapted from Cederling [Cederling 92]	23
Figure 7:	Layered Software Architecture for Ship System 2000	24
Figure 8:	A More Detailed Rendering of the SS2000 Layered Architecture	24
Figure 9:	Another Description of the Architectural Layers	25
Figure 10:	Using (and Bypassing) the Common Object Manager Repository	26
Figure 11:	Mk2.5 Project Organization (1980 - 1985)	33
Figure 12:	SS2000 Organization, 1987 - 1991	35
Figure 13:	SS2000 Organization Since 1994	37
Figure 14:	Approximate Software Staff Profile	39
Figure 15:	Timeline for Early Product Line Development	46
Figure 16:	System Product Development Phases Showing Documents Produced at Each Phase [Cederling 92]	52
Figure 17:	System Family Development Phases [Cederling 92]	55
Figure 18:	SS2000 Training Program	64
Figure 19:	Domain Engineering and Application Engineering	71

List of Tables

Table 1:	Systems Built by CelsiusTech Before 1985	13
Table 2:	System Product Development Phases	53
Table 3:	System Family Development Phases	56
Table 4:	Observed Product Line Development Stages	67
Table 5:	Enabling Organizational Attributes for Product Line Stages	68
Table 6:	SS2000 Documentation Set, Adapted from Cederling [Cederling 92]	73
Table 7:	DoD-STD 2167A Reviews and CelsiusTech Documentation, Adapted from Cederling [Cederling 92]	75
Table 8:	Estimated Student Time Per Exercise	78
Table 9:	Typical Curriculum Usage Scenario	81

Acknowledgments

The authors gratefully acknowledge the gracious assistance of CelsiusTech Systems, and especially that of Ulf Olsson. We are grateful for the time provided by all of the people at CelsiusTech Systems who spoke with us, and we are grateful to the management of CelsiusTech Systems, particularly Jaak Urmi, for allowing us to visit.

The authors also gratefully acknowledge the hard work of our reviewers, whose comments and questions improved the result: Paul Kogut, Linda Northrop, Sholom Cohen, David Bristow, Jim Withey, and Dan Burton. Finally, Bill Pollak's excellent editorial skills are also deeply appreciated.

A Case Study in Successful Product Line Development

Abstract: A product line is a set of related systems that address a market segment. Building a product line out of a common set of core assets, as opposed to building each member system separately, epitomizes reuse. Although software technology is key to achieving a product line capability, organizational and process considerations are just as crucial. This report describes the experience of one company, CelsiusTech Systems AB of Sweden, that builds large, complex, embedded, real-time shipboard command-and-control systems as a product line, developed in common from a base set of core software and organizational assets. The report describes the changes that CelsiusTech had to make to its software, organizational, and process structures to redirect the company toward a product line approach that yielded substantial economic and marketplace benefits to the company.

1 Introduction

1.1 Purpose and Background

This report relates the experience of a company (CelsiusTech Systems AB, a Swedish naval defense contractor) that has successfully adopted a *product line* approach to building large, complex, software-intensive systems. A product line is a group of related systems that address a market niche or mission. In the case of CelsiusTech the product line is called Ship System 2000 and consists of approximately a dozen shipboard command-and-control systems for Scandinavian, Middle Eastern, and South Pacific navies.

1.1.1 Product Lines

Building a product line from a common asset base requires careful management, for it often entails multiple development efforts operating in parallel. However, if those development efforts proceed independently without taking advantage of each other, a powerful opportunity for gaining economies from sharing and reusing knowledge is lost.

Through careful management and engineering, a product line can be developed that exploits a common set of *assets,* ranging from reusable software components to work breakdown structures for the individual projects. A group of systems built from a common set of assets is a *product family*. A product family need not constitute a product line (the individual systems may have no clearly coordinated role in meeting a market). Conversely, a product line need not be built from a product family (the individual systems may each be developed independently). However, building a product line as a product family leverages and amortizes prior investment to the maximum degree possible. Assuming the product line has a sufficient number of members, the sum cost of building the product line as a whole becomes much less than the sum cost of building the individual systems independently. Producing a new system becomes less a matter of creation than one of generation or modification (from a previous system, or from a generic version of a “prototypical” family member). Uncertainties associated with building an unprecedented system are nullified, or at least isolated, by the product family approach: Problems will usually have already been faced and solved by early members of the family.

1.1.2 Benefits—And Costs

CelsiusTech Systems reports extremely high productivity and extremely low cost with their product line[1] approach, compared to traditional development paradigms. CelsiusTech Systems is able to turn out large (1-1.5 million lines of Ada code), embedded, performance-critical, highly distributed (up to 70 processors on up to 30 local area network nodes) shipboard command-and-control systems for modern naval vessels with

- software accounting for approximately 20% of the system cost, as opposed to 65% for similar systems not developed using the product line approach
- high schedule and cost predictability
- high customer satisfaction
- integration handled by one or two people at most
- predictable performance and distributed-system issues in new systems

These results were not without cost. A formidable initial investment in time and capital was required to initiate the product line approach at CelsiusTech Systems. Much of this investment was attributed to the fact that the company needed to embrace new technologies that were dawning in their application area—Ada, for example—but much of it was directly attributable to the business decision to pursue product line development. For example, an organizational restructuring and intensive training program were required.

1.1.3 Technology Isn’t Enough

The technological approaches to product line development flow directly from the software design and reuse communities. Domain models, abstract interfaces, layered architectures, objects and information-hiding, and parameterized code units all play a major role in CelsiusTech’s success. We will report no new software engineering technologies. What we discovered is that the non-technical issues—business, organizational, personnel, managerial, process, and cultural—are at least as critical as the software engineering principles involved. We will answer the following questions and others relative to the CelsiusTech experience:

- personnel issues
    - What kind of training is required for personnel?

1. Henceforth, when we speak of a product line we mean one that was developed using a common set of core assets—i.e., a product line built as a product family.

    - How have the company's personnel needs shifted? What kind of expertise is in most demand now?
- customer interaction
    - How are new contracts negotiated with a customer in the context of a product line approach?
    - How do the customers interact with each other to collectively manage the evolution of the product line?
- process issues
    - What is the process for producing a new member of the product line? What is the process for upgrading an old one?
    - What are the units of reuse?
    - How are testing and integration performed to take advantage of reuse?
- Organizational issues
    - What is the organizational structure that maintains the product line and each specific member of it?
    - What is the nature of the initial investment required in establishing a product line?

1.1.4 Purpose

We intend this report, therefore, to help fill a gap in the literature about producing product lines. Until now, this literature has concentrated on the technical aspects. We intend this report to be useful to the following audiences:

- managers who are considering instigating a product line approach in their organizations and would like to benefit from experience. For them, we detail the factors and problems involved in the successful development of a product line and recount one organization's experience in dealing with those factors and overcoming those problems.
- advocates of the product line approach who would like to convince their management to adopt it. For them, we provide proof that the product line approach can work, can be made to be economically feasible, and can result in a robust business climate.

This report illustrates what it took to help an organization make a successful transition to a new business and technical paradigm: the right technology, an effective transition plan, and strong management committed to making it work.

1.2 Product Lines and Reuse

In the early days of computer programming, when computers were expensive and software was (by comparison) cheap, little thought was paid to reusing programming assets. That is, of course, no longer the case. Software is usually by far the most expensive component of an operational computer system. Even if the computer is part of a larger system (such as an aircraft or a nuclear power plant), the cost of the software is a significant if not overwhelming factor, especially when the system is long lived and incurs significant modification throughout its service life. The economics of software engender approaches to reduce the cost of development, deployment, and life-cycle maintenance of software systems.

A basic approach to reducing cost is to leverage the investment across more than one system. Thus, *software reuse* is a key strategy for reducing the per-system cost of software. This strategy avoids special-case development, but instead reuses already existing assets to the extent possible. Tactics include

- buy versus build: Acquire software systems (or their components) from commercial vendors or external suppliers.
- plan for reuse: For those components developed for internally developed components, build them so that they are useful in systems other than the one for which they were developed, ideally in systems that the organization is planning to build in the future.

Clearly an asset is more likely to be reused in a future system (hence reducing its overall cost) if that system is similar to the one for which the asset was acquired. For example, software embodying a navigation algorithm for an aircraft is very likely to be reused in another aircraft system, somewhat likely to be reused in a ship application, and not likely at all to be reused in a banking system.

Thus, an organization has a certain economic incentive to structure its marketable products so that they are functionally and structurally similar to one another. A group of similar products, structured to take advantage of each other's assets in order to yield economies of production, is a product line built as a product family. When a new order is received, the organization produces a product that may still perform to the new customer's specific requirements, but its production is viewed as instantiating a version of a generic product rather than developing anew.

Product lines epitomize reuse, sometimes in ways not immediately evident. Consider the assets that can be reused across members of a product line:

- components: Components are applicable across individual products. Far and above mere code reuse, component reuse includes the (often-difficult) design work that was invested initially. Design successes are captured and reused; design dead ends are avoided, not repeated. This includes the design of the component's interface, its documentation, its test plans and procedures, and any models (such as performance models) used to predict or measure its behavior.
- personnel: Because of the commonality of the applications, personnel can be fluidly transferred among projects as required. Their expertise is applicable across the entire line.
- defect elimination results: Because errors eliminated in a component on one product raise the quality of the corresponding components in all the other products, reliability and overall quality increase.
- project-planning expertise: Because past experience is a high-fidelity indicator of future performance, budgeting and scheduling are more predictable. Work breakdown structures need not be invented each time. Teams, team size, and team composition are all known quantities.

- performance analyses: Performance models, schedulability analysis, distributed-system issues (such as proving absence of deadlock), allocation of processes to processors, fault-tolerance schemes, and network-load policies all carry over from product to product.
- processes, methods, and tools: Configuration-control procedures and facilities, documentation plans and approval processes, tool environments, and a host of other day-to-day engineering-support activities can all be carried over from product to product.
- exemplar systems: Deployed products serve as high-quality demonstration prototypes. Because feasibility is not an issue, customer satisfaction increases (and risk decreases). Deployed products also serve as high-quality engineering prototypes, providing highly detailed performance models.

Although not standard practice, the technical means for achieving software reuse based on product lines are well established. Applicable areas, all extensively covered in the software engineering literature, include

- domain analysis: producing a domain model of the product line that identifies what is common to all members, and identifying the ways in which members can vary from one another
- software architecture: adopting a standard skeletal infrastructure that applies to all (actual and potential) members of the product line, as identified by the domain analysis
- design methodology, including object-based design: applying the principles of separation of concerns and information hiding to identify the components that the architecture comprises, and assigning roles and responsibilities to each; then designing each component with an interface that provides the services of the component to the rest of the system in an implementation-independent fashion

Experience shows, however, that a technical approach alone is not sufficient to produce a successful capability for product line production. Economic, organizational, personnel, managerial, customer, marketplace, and process factors all play a crucial role in establishing and maintaining a product line; but unfortunately these factors are all but overlooked in the literature.

1.3 Why Did We Visit CelsiusTech?

Our investigation into the CelsiusTech Systems experience began by selecting CelsiusTech as one of a series of case studies in software architecture. We intended the case studies to illustrate how architecture can be an important tool in satisfying high-risk project requirements such as high performance, high reliability, or (in the case of CelsiusTech Systems) reusing assets from one member of a product line to another.

In CelsiusTech Systems, we discovered one of the very few organizations that had embarked consciously and purposefully on producing a product line as its main business approach. Further, the evolution from previous business practices had been comprehensive, and represented a fundamental change in CelsiusTech's business approach. The process was well documented; the corporate memory was largely intact, and papers had been written about the process that recorded the transition. Finally, CelsiusTech Systems personnel were willing to share their experiences with us candidly and without the usual mantle of corporate secrecy.

The CelsiusTech Systems experience represented a rare opportunity to record and report on an organization-wide transition to a product line business approach. This report is the result.

1.4 Gathering Information

Information gathered in this report came from the following sources:

- previous knowledge. One of the authors of this report (Brownsword) had previously worked for Rational Software Corporation, which supplied part of the development environment and provided technology-transition support throughout the development of the product line.
- previously published reports. We consulted CelsiusTech Systems' marketing literature, which touted their product line approach. Also, CelsiusTech Systems had been previously featured in a dissertation, from a Swedish university, that outlined a set of industrial software-development practices. Although the focus of this dissertation was not on product lines per se, it provided an excellent summary of CelsiusTech Systems' product line architecture and methodological approach to engineering [Cederling 92].
- interviews. In May of 1995, the authors visited CelsiusTech Systems at their headquarters near Stockholm, Sweden and conducted interviews with more than two dozen key personnel involved in the planning, development, and production of their family of shipboard command-and-control systems. Senior managers, line managers, project managers, designers, programmers, testers, and trainers and educators were all interviewed over a period of three full days. On the fourth day, we presented a technical seminar outlining the major lessons that we had learned during our visit.
- documentation. We reviewed CelsiusTech Systems' Ada interface specification that provides developers with information regarding key component interfaces and usage guidelines. We also reviewed the document that records CelsiusTech's development methodology and processes.

As a condition of cooperation, CelsiusTech Systems management was given the right to review this report before publication. Although everyone with whom we spoke was forthright and candid, we understood that some of the data (especially concerning the company's business plans) were confidential. The purpose of the review was to make sure that we did not inadvertently divulge any sensitive information and that the report was free of factual errors.

1.5 Report Structure

The report consists of the following sections:

- Section 2 describes CelsiusTech Systems: the company, its business domain and product line, the impetus for creating a product line business approach, and the results that have been achieved thus far.
- Section 3 outlines the architectural foundations for the Ship System 2000 product line.
- Section 4 discusses the organizational, management, and personnel factors associated with CelsiusTech Systems' creation, use, and evolution of its product line.
- Section 5 summarizes the process, methods, and tools applied to the development and maintenance of the assets used to produce the product line.

- Section 6 discusses the learning curves that CelsiusTech had to climb to educate its management, its technicians, and its customers in the new approach.
- Section 7 concludes the report with our analysis of the CelsiusTech experience.

We provide additional information in appendices, including detail on the life-cycle processes and education curriculum applied by CelsiusTech Systems.

2 CelsiusTech Systems

2.1 Background

CelsiusTech Systems is one of Sweden's leading suppliers of command-and-control systems. It is part of Sweden's largest—and one of Europe’s leading—defense industry groups. Other members include Bofors, Kockums, FFV Aerotech, and Telub.

CelsiusTech is composed of CelsiusTech Systems (advanced software systems) and CelsiusTech Electronics (defense electronics). Until recently, CelsiusTech IT (information technology systems) was also a member company. CelsiusTech has approximately 2000 employees and annual sales of $300 million[1]. The main site is near Stockholm, Sweden, with subsidiaries located in Singapore, New Zealand, and Australia.

This study focuses on CelsiusTech Systems[2]. CelsiusTech’s areas of expertise include command, control, and communication (C3) systems, fire-control systems, and electronic warfare systems for navy, army, and air force applications. The current organization has undergone several changes in ownership and name since 1985. Originally Philips Elektronikindustrier AB, the division was sold to Bofors Electronics AB in 1989 and reorganized into NobelTech AB 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 have remained, thereby providing continuity and stability.

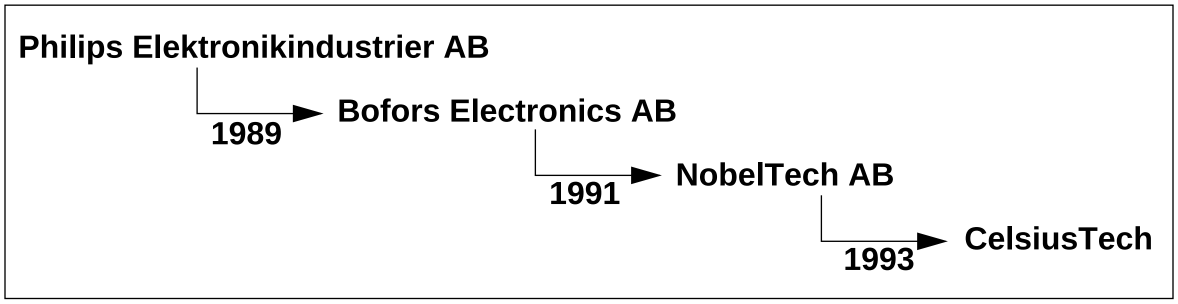


Figure 1: CelsiusTech Systems’ Corporate Evolution

In this section, we first summarize the naval product line, describe its current status, and preview CelsiusTech’s results from creating it. Next, we explore the events and motives that led CelsiusTech to create a product line. Finally, we detail the technical and organizational chang-

1. All monetary amounts will be given in U. S. dollars.

2. To increase the readability of this report, all subsequent references to CelsiusTech Systems are shortened to CelsiusTech.

es that CelsiusTech initiated to transition its organization to a product line approach. We conclude the section with a summary of the key points and our analysis.

2.2 The Ship System 2000 Naval Product Line

CelsiusTech refers to its naval product line as the *Ship System 2000 (SS2000)* family; this product line is also known as the *Mk3* family of products. Each member of the product line provides an integrated system unifying all weapons systems, command-and-control, and communication functions on a warship. Typical system configurations include 1-1.5 million lines of Ada code distributed on a local area network (LAN) with 30-70 microprocessors.

2.2.1 Systems in the Product Line

Widely differing naval systems, both surface and submarine, have been or are being built as part of the SS2000 product line. These include

- Swedish Goteborg 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 (3225 tons)
- Danish Thetis class Ocean Patrol vessels (2700 tons)
- Swedish Gotland class A19 submarines (1330 tons)
- Pakistani Type 21 class frigates
- Republic of Oman patrol vessels
- Danish Niels Juel class corvettes

The last three are recently initiated projects for which there is insufficient data for substantive contribution to this report. The Naval Systems Division has sold a total of 55 of its Mk3 naval systems to seven different countries around the world [CelsiusTech 96].

2.2.2 Flexibility Yields Variability

Ships for which CelsiusTech supplies systems vary greatly in size, function, and armaments. Each country requires different operator displays on differing hardware and presentation languages. Sensors and weapons systems and their interfaces to the software vary. Submarines have different requirements from those of surface vessels. Computers in the product line include 68020, 68040, RS/6000, and DEC Alpha platforms. Operating systems include OS2000 (a CelsiusTech product based on Microware Systems Corporation's OS/9 real-time operating system), AIX, POSIX, Digital UNIX, and others. Yet the SS2000 product line is flexible and robust to support this range of possible systems through a single architecture, a single asset base, and a single organization. Section 3 of this report elaborates on the architectural attributes we found contributing to such flexibility.

2.2.3 Applying the SS2000 Product Line Assets to Related Business Areas

Although CelsiusTech's largest business area is naval applications, it has expanded its business into the ground-based air defense systems market through a merger with the Command and Control Division of Ericsson. The current project is a new C3 system for the Swedish Air Force. By reusing the abstractions inherent in the SS2000 architecture, CelsiusTech was able to quickly build the new architecture, lifting 40% of its components directly from the SS2000 asset base. Other business opportunities are being explored in new domains where the architecture and other product line assets are likely to prove a good fit.

2.2.4 Shrinking Schedules

Figure 2 shows the status and schedules for the current systems under development in the CelsiusTech product line. System A is the basis of the product line. Customer project A has run almost nine years, although functional releases were running on the designated ship since late 1989. System B, the second of the two original projects, required approximately seven years to complete; not dissimilar to the previous non-product line Mk2.5 system. System B was built in parallel with system A, validating the commonality in the product line. While neither system A nor B *individually* showed greater productivity, CelsiusTech was able to build two systems (and the product line assets) with roughly the same number of staff as for a single project.

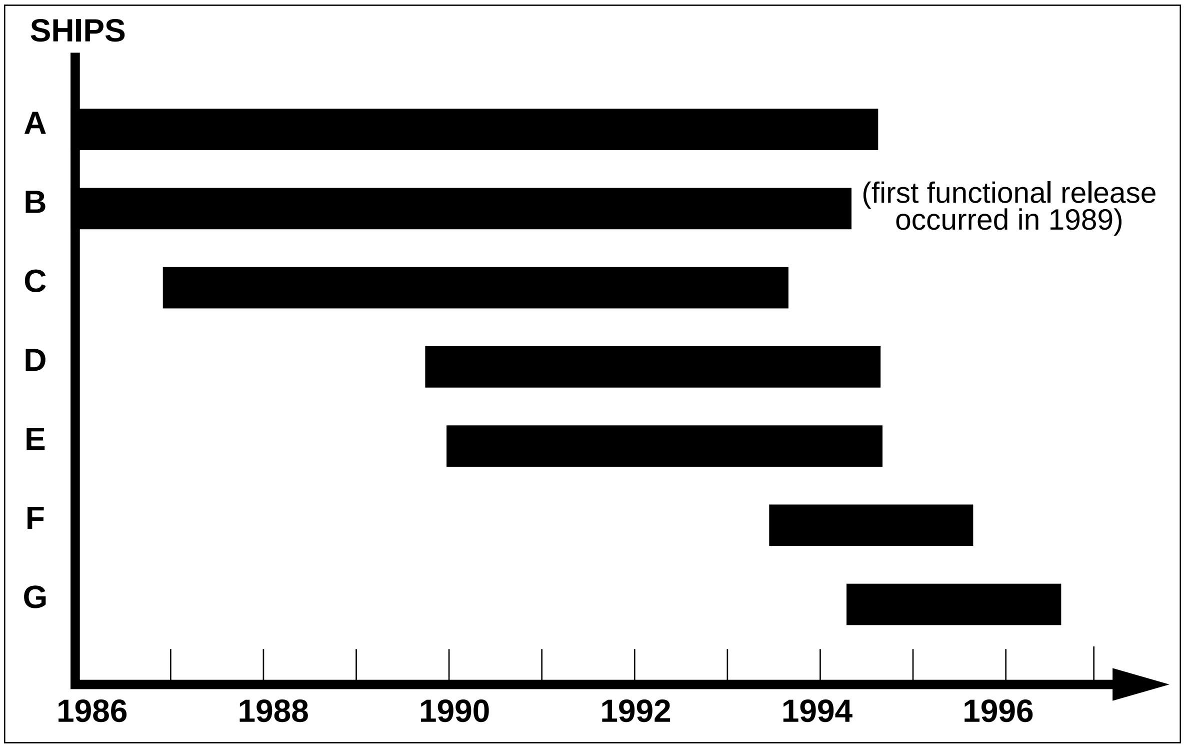


Figure 2: Product Schedules: From Start to Final Delivery

Systems C and D were begun after much of the product line asset base existed, with a correspondingly shortened completion time. Since they are fully leveraging the product line assets, systems E and F indicate a dramatic further schedule reduction. CelsiusTech reports that the last three ships are all *predictably* on schedule.

While the production schedules show time to market for a product, they do not indicate how well the systems reflect the use of a common asset base. Figure 3 shows the degree of commonality across the CelsiusTech naval systems[1]. On average, 70-80% of the systems consist of components that are used verbatim (i.e., checked out of a configuration-control library and inserted without code modification).

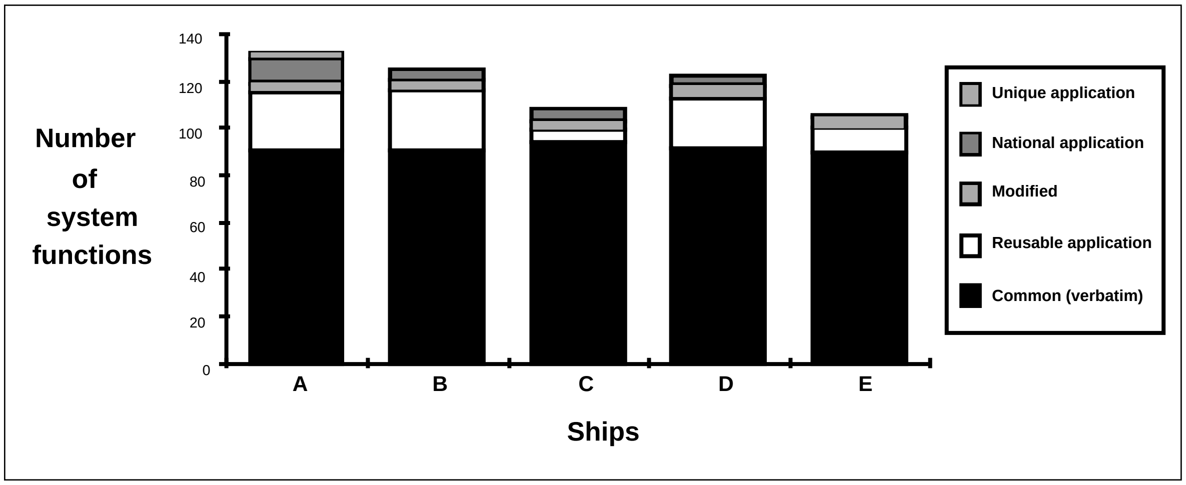


Figure 3: Reuse Across SS2000 Product Line

2.3 Impetus for a Product Line Approach

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

From 1975 to 1980, CelsiusTech shifted its technology base from analog to 16-bit digital systems, creating the so-called Mk2 systems. These systems tended to be small, real time, and embedded. The company progressively expanded the functionality and its expertise with real-time applications in the course of building and delivering 17 systems.

From 1980 to 1985, customer requirements were shifting toward the integration of fire control and weapons with command-and-control systems, thus increasing the size and complexity of

1. A “system function” as used in this figure is defined in Section 3.4.1.

delivered systems. The Mk2 architecture was expanded to provide for multiple autonomous processing nodes on point-to-point links, resulting in the Mk2.5 systems. These systems were substantially larger, both in delivered code (700,000) and number of developers (300 engineer years over 7 years). Conventional development approaches were used. While these had served the company well on the smaller Mk2 systems, difficulties in predictable and timely integration, cost overruns, and schedule slippage resulted. While such experiences are painful, 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 programming language (in this case, RTL/2, a Pascal-like real-time language designed by John Barnes).

	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 command, control, communications, and intelligence (C3I); RTL/2
Size	30-100K source lines of code (SLOC)	700K SLOC; 300 engineer-years over 7 years
Platforms	Analog and 16-bit digital systems	Multiprocessors, minicomputers, point-to-point links

Table 1: Systems Built by CelsiusTech Before 1985

In 1985, CelsiusTech (then Philips) was awarded two contracts simultaneously—one for the Swedish navy and one for the Danish navy. Both ships' requirements indicated the need for systems larger and more sophisticated than the previous Mk2.5. Looking at recent experience with the Mk2.5 system and the need to build two even larger systems in parallel, management and senior technical staff were stimulated to reevaluate their business and technical approaches. In its analysis, CelsiusTech determined that the continuation of the development technologies and practices applied on the Mk2.5 system were insufficient 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: recognizing 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 life span for naval systems. During that time, changes in threat requirements and technology advances must be addressed. The more flexible and extendable the product line, the greater the business opportunities. These business drivers or requirements forged the technical strategy.

The technical strategy would have to provide flexible and robust building blocks to populate the product line assets 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 asset base to sustain the business viability of the strategic decision.

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 of system (the Mk3) to include new hardware and software, and a new supporting development approach. These would serve as the infrastructure for new systems development for the next decade or two.

2.4 What Was New? Everything!

CelsiusTech's decision to convert its business strategy to a product line approach coincided with a time of high technology flux. This meant that in order to implement the technical strategy for the SS2000 product line, virtually all aspects of the hardware, software, and development support changed. 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 fail-soft and fault-tolerant requirements. The software approach shifted from RTL/2-based, structured analysis/design, waterfall development processes to Ada83 with more object-based[1] 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 4.

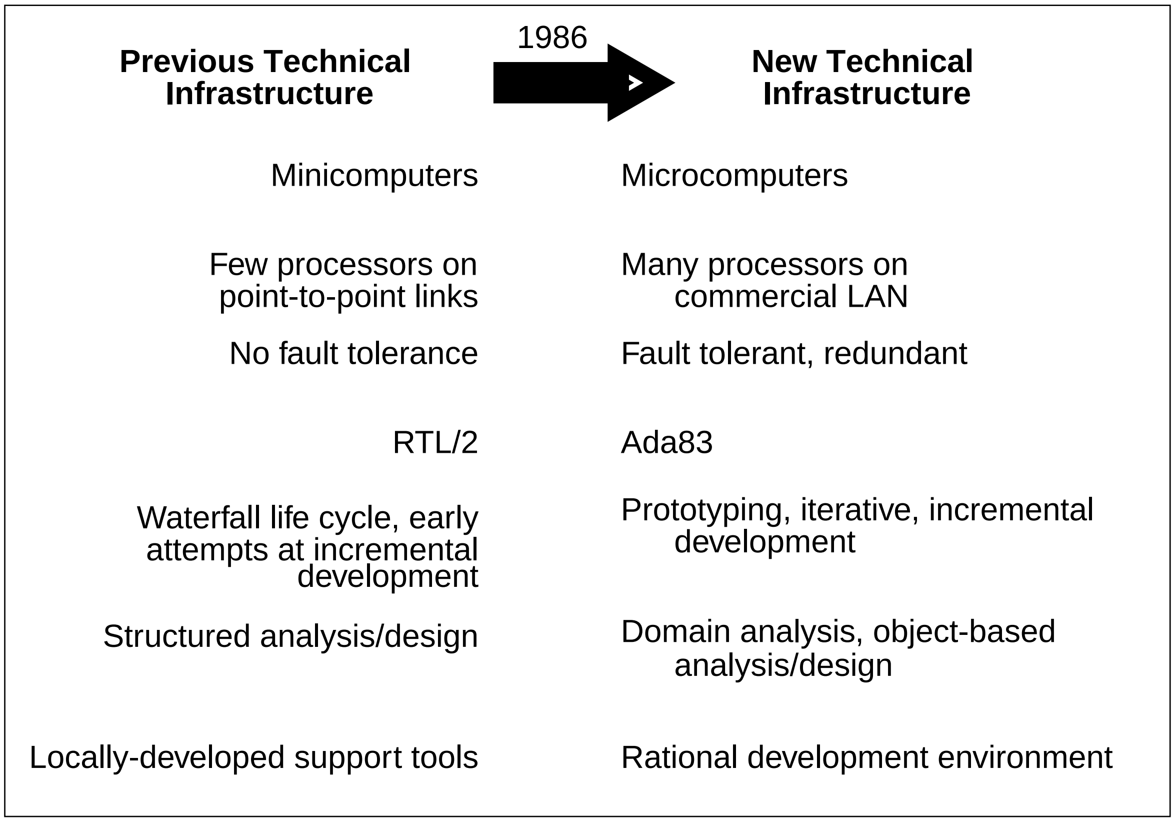
2.5 Analysis

The CelsiusTech experience reveals several key factors in the establishment of the SS2000 product line. Some were advantages; some were inhibitors. They include the following.

2.5.1 Ownership Changes

While it is routine to buy, sell, and restructure companies, the effect on an organization attempting to adopt significantly different business and technical strategies is fundamental and potentially devastating. Typically, management changes associated with company-ownership transactions are sufficient to stop any transition or improvement efforts under way. Thus, to start and *sustain* the transition to a new technology approach as significant as product lines across three changes in ownership is remarkable. There is some evidence that the company's highest level management was preoccupied with the ownership changes. Middle management thus was allowed to proceed on its course unfettered by management who might otherwise

1. We distinguish between object-based and object-oriented design. At the time, what CelsiusTech did was considered object-oriented; since then, however, that term has come to mean object-based design (i.e., design of information-hiding modules such as abstract data types whose facilities are invoked by access programs—methods—on the interface) plus class-based inheritance (which CelsiusTech did not use).



1986

Previous Technical Infrastructure	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/design	Domain analysis, object-based analysis/design
Locally-developed support tools	Rational development environment

Figure 4: Changing Technical Infrastructures

have been hesitant to approve the necessary up-front investments to achieve the product line capability. On the whole, while commending CelsiusTech for staying the course in the midst of chaos, we believe that ownership changes did not play a decisive role in the product line story at CelsiusTech. The story does demonstrate, however, that given a bold vision and strong leaders in the right places to carry out that vision, even the most severe organizational turmoil can be withstood.

2.5.2 Necessity Is the Mother of Invention...

The award of two major naval contracts in 1986, ostensibly a reason for celebration, was regarded as the precipitation of a crisis by CelsiusTech. They immediately realized that they had neither the technical means nor the personnel resources to prosecute two large development efforts, each pioneering new technologies and application areas, simultaneously. Since all CelsiusTech contracts are fixed-price, large-scale failure meant large-scale disaster, and previous 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 company's viability was clearly at stake.

2.5.3 ...But Determination Is the Midwife

Deciding to adopt a product line approach was one thing; executing it was quite another. It was important that CelsiusTech committed its entire naval business to the approach rather than trying to hedge its bets by only partially embracing it. Further, the commitment was maintained by CelsiusTech's management. During the paradigm shift, when the new ways of doing things conflicted with the old, a multitude of problems arose. Unfamiliarity breeds insecurity, and it must have been very tempting to abandon the new approach or at least to dilute it. CelsiusTech had strong managers in the right places to solve problems as they arose in keeping with the new approach, to maintain the company's direction, and to make it clear that the organization was going to stay the new course.

2.5.4 Riding a Wave of Technology Changes

In 1986, all the selected technologies indicated in Figure 4 were immature with limited use in large, industrial settings. Large, real-time, distributed systems making extensive use of Ada tasking and generics were envisioned but at the time were unprecedented. Object-based development for Ada was still a theoretical discussion. Thus from 1986 to 1989, CelsiusTech was coping with

- the maturation of immature technologies, such as Ada and object technology
- the maturation of immature supporting technology, such as networking and distribution
- the maturation of immature infrastructure technology, such as development environments and tools to assist in the automation of the 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 requirements across several customers

These maturation issues significantly increased the time required to create the product line capability. Another organization making the same development paradigm shift today would, we believe, be in a much less tenuous position. 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 are all either mature or at least usable and readily available. CelsiusTech estimates that up to 1/3 of its initial technology investment was spent building assets that can now be purchased commercially. The remaining 2/3, however, still represent a formidable investment.

2.5.5 Product Line Creation Time

The fact that it took many years to develop the product line capability must be placed in the context of the migration to new and immature technology, and the shifts in company ownership. We would expect it to take significantly less time if undertaken in a more stable corporate and technological environment.

3 Architectural Foundations for the SS2000 Product Line

3.1 Requirements for an Architecture

To derive new products efficiently from an asset repository, the products must be structured similarly so that they share components. This means there must be standard componentry with agreements about the responsibility, behavior, performance, interface, locality of function, communication and coordination mechanisms, and other properties of each component. This family-wide structure, the componentry it comprises, and the properties about each component that are constant across all members of the product line constitute the *architecture* for the family.

The primary purpose of an architecture is to facilitate the building of a system that meets its requirements; architecture is a means to this end. The architecture for each Ship System 2000 (SS2000) product line member is no exception: It must allow for the production of a system that meets its behavioral, performance, resource utilization, and quality requirements. However, the SS2000 architecture carries the additional burden of application to an entire class of systems, not just an individual one. Requirements for such an architecture include the ability to replace components with ones tailored to a particular system without disrupting the rest of the architecture.

This section presents the *technical* foundations that CelsiusTech laid for the product line approach upon which it embarked. In particular, it highlights the architectural decisions that the CelsiusTech senior designers made to meet the operational and programmatic requirements for their family of systems.

There are many facets to an architecture, and SS200 is no exception. Each view conveys (and suppresses) certain information and is useful for different purposes. No single view is authoritative or complete; each presents a portion of the whole story. In the following sections, we present different views of the SS2000 architecture, including:

- the physical architecture, which describes the networks and processor configurations on which the software must run
- the major software components of SS2000 products, and the protocols and conventions by which they coordinate
- layering, which is a static structure that provides for much of the flexibility of the architecture

3.2 Operating Environment

The requirements of modern shipboard systems influence design solutions in fundamental ways. Sensors and weapons systems are deployed all over the ship; crew members interact with the system through a multitude of separately housed workstations. The man-machine interface 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. In addition, the man-machine interface varies across members of the product line, as do details of the operational mission, hardware environments, and so forth. The software of SS2000 must meet operational requirements of each ship as well as flexibility requirements to cover the family as a whole.

The requirement to run on a distributed computing platform has broad implications for the software architecture. Distributed systems 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. When the group of developers we interviewed were asked to explain the architecture, their first response was that it was distributed.

3.3 Physical Architecture

Figure 5 illustrates a typical physical architecture for the system. A redundant LAN is the communications backbone for the system, connecting from 30 to 70 different processors that co-operate to perform the system's work. 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. A node can host up to 6 processors. The LAN is currently 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. Multiple workstations are supported.

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

3.4 Software Architecture

This section describes the software architecture by identifying the major components of the software, the way they are arranged to provide flexibility across the product line, and the ways in which the components communicate and coordinate with each other.

3.4.1 System Functions and System Function Groups (SFGs)

Several types of components exist in the SS2000 architecture. Each serves a particular purpose. Functional requirements are embodied in *system functions*. A system function is a collection of software that implements a logically connected set of requirements; it may be thought of as a subsystem. A system function is composed of a number of Ada code units, which are the basic unit of work for a program, and the unit of combination for the Ada compiler and environment. System functions comprise roughly 5K-25K lines of Ada. An example is a component that handles all the ballistic calculations for a gun.

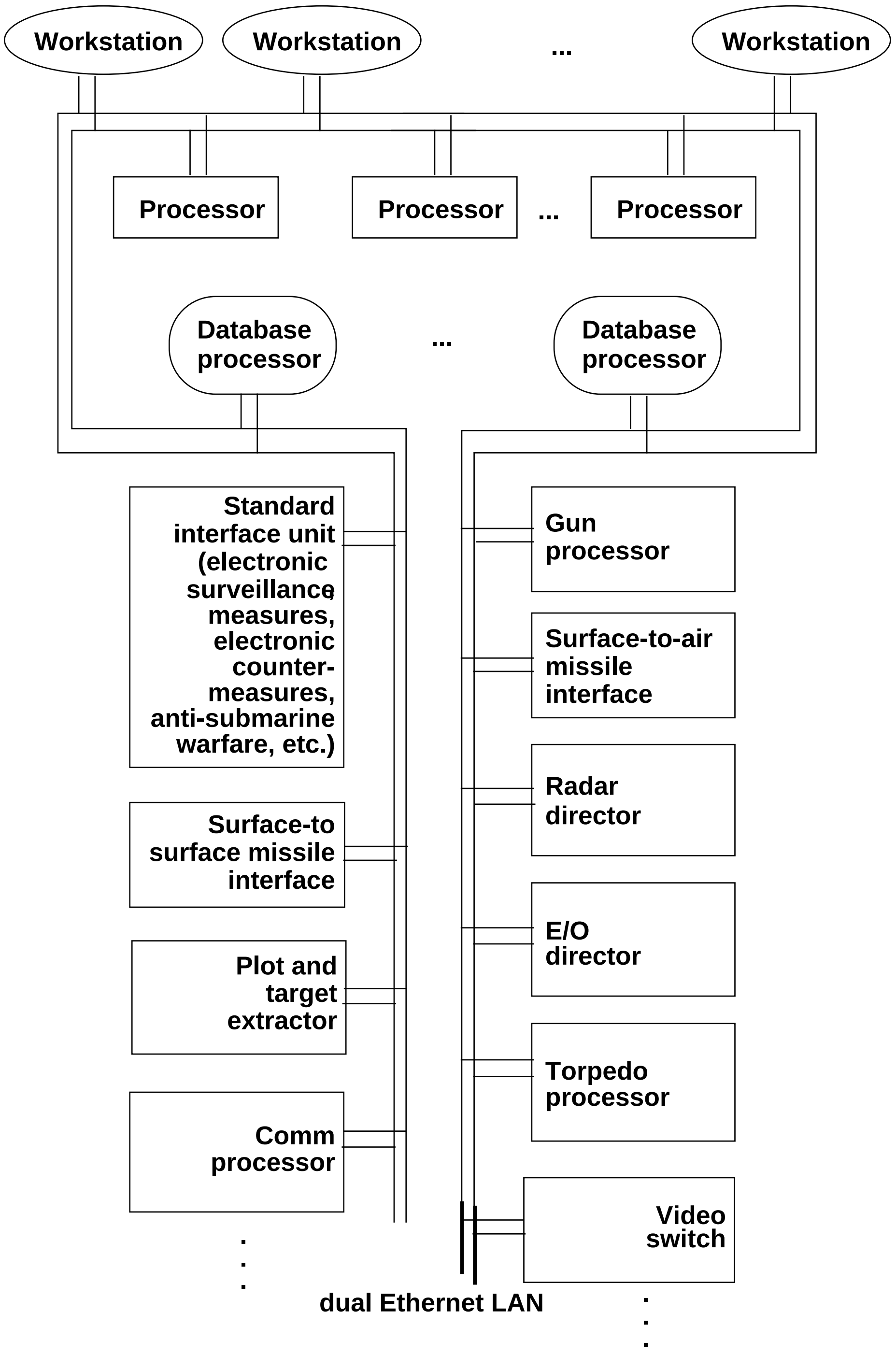

Workstation
Workstation
...
Workstation
Processor
Processor
...
Processor
Database processor
...
Database processor
Standard interface unit (electronic surveillance measures, electronic counter-measures, anti-submarine warfare, etc.)
Surface-to surface missile interface
Plot and target extractor
Comm processor
Gun processor
Surface-to-air missile interface
Radar director
E/O director
Torpedo processor
Video switch
dual Ethernet LAN

Figure 5: Typical Physical Architecture of a Ship System 2000 Product

A *system function group* comprises a set of system functions and forms the basic work assignment for a development team. System function groups, primarily organizational units for management, are clustered around major functional areas, including

- C3
- weapons control
- “fundamentals,” meaning facilities for intrasystem communication and interfacing with the computing environment
- man-machine interface (MMI)

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

System functions, not the Ada code units, are the basic units of test and integration for systems in the product line. This is a crucial point. The primacy of system functions and system function groups allows a new member of the product line to be treated as the composition of a few dozen high-quality, high-confidence components that interact with each other in controlled, predictable ways as opposed to thousands of small units that must be regression tested with each new change. Assembly of large components without the need to retest at the lowest level of granularity for each new system is a critical key to making reuse work.

Figure 6 illustrates the relationship between the various kinds of componentry SS2000 consists of about 30 system function groups, each comprising up to 20 or so system functions.

3.4.2 Architecture as Layers

The SS2000 software architecture is layered, meaning roughly that

- Components are divided into groups called layers. The grouping is based roughly on the type of information that each group encapsulates. Components that must be modified in the event of a change in hardware platform, underlying LAN, or internode communication protocols all form a layer. Components that implement functionality common to all members of the family form another layer. Finally, components specific to a particular customer product form a layer.
- The layers are ordered, with hardware-dependent layers at one end of the relation and application-specific layers at the other. Layers that are neither hardware- nor member-specific reside in the middle.
- Interactions among layers are restricted. A component residing in one layer can access components only in its own or the next lower layer.

In Ship System 2000, the bottom layer is known as Base System 2000; it provides an interface between the 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

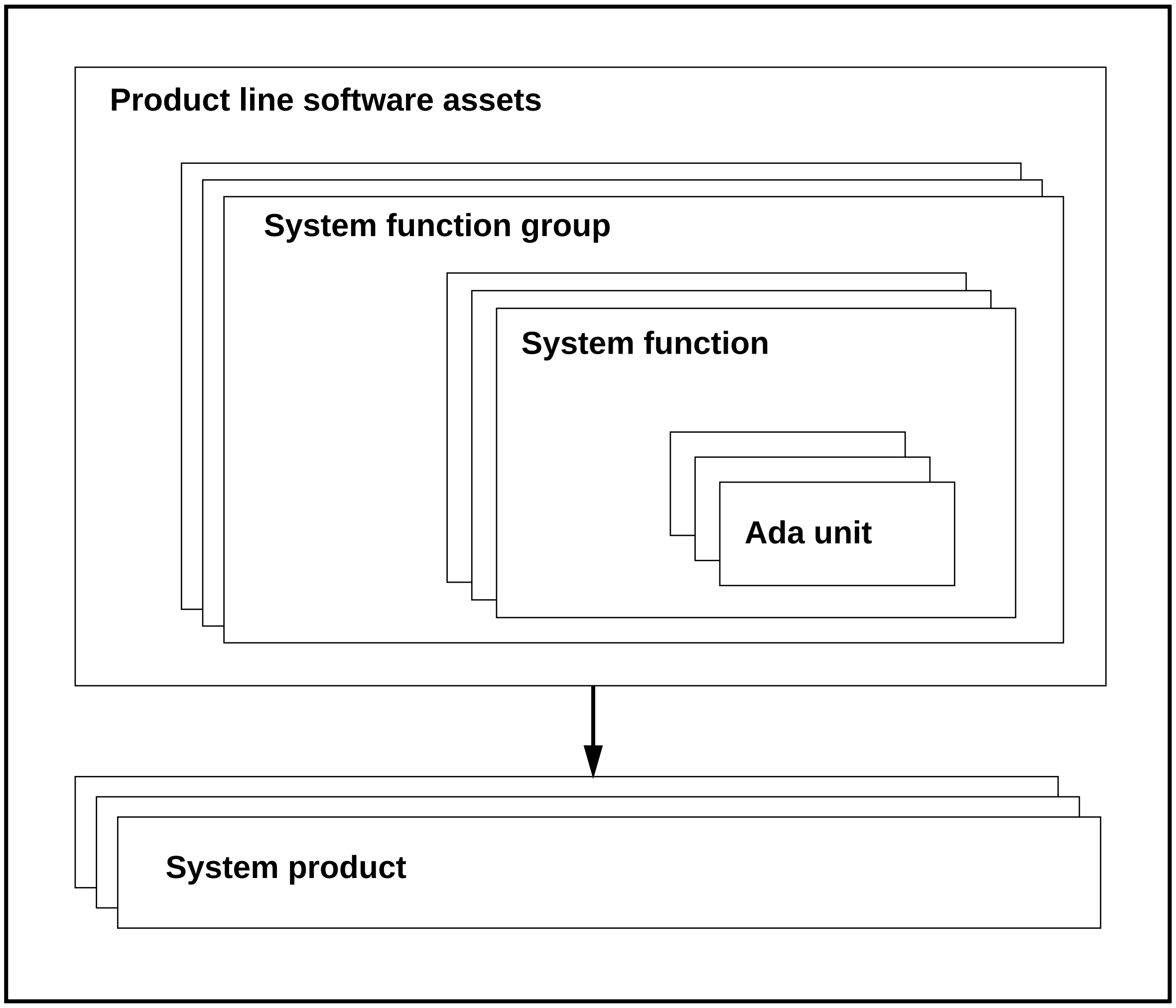


Figure 6: Units of Software Components, Adapted from Cederling [Cederling 92]

being sensitive to the underlying computing platforms, network topologies, allocation of functions to processors, etc.

Each layer in the software architecture comprises a set of system functions; that is, each system function is assigned to exactly one layer. Lowest layers include those system functions that embody requirements independent of the ship-based application, while system functions at the highest layer embody requirements specific to some but not all customer systems. System functions in middle layers embody requirements that are at various levels in between.

Note that a layer is primarily a pre-runtime entity. That is, assignment to a layer primarily describes the knowledge, or lack of knowledge, that is allowed to be reflected in the coding of that component. Also, implementation techniques such as inline macro expansion can, at runtime, actually allow a high-level layer to invoke quite low-level facilities directly.

Interviews made clear that layering is a central design principle that is well promulgated throughout the development organizations. However, it was also clear that there is more than one rendition of the layering scheme, depending on the audience or desired level of detail. In other words, there was no central dictionary that named the single layer to which a code unit belonged. Rather, the layers seemed to be more of a device to help explain the product line philosophy for insulating the system from changes in platform, network, etc., and for rapidly producing new members of the product line. An exception is Base System 2000, which is maintained by an organizational element devoted to it; hence, its contents constitute a specific, well-defined layer. Figure 7 illustrates one rendering of the architectural layers of SS2000 that we encountered; Figure 8 illustrates another, and Figure 9 yet another. They do not conflict with each other; rather, they provide complementary explanations of the same ideas.

Customer-specific applications	Generated where possible; contains presentation language, customer-specific configuration information, etc.
Generic applications	Standard computations: e.g., ballistic calculations
Technology	Platform portability: operating system, hardware, file systems, etc.

Figure 7: Layered Software Architecture for Ship System 2000

General applications: 100% Ada	Target tracking	Fire control	anti-submarine warfare (ASW)	electronic counter-measures (ECM)
	Operator's console			
Common applications: 100% Ada	Common applications			
	Picture compilation	Ships information		
Fundamentals: 95% Ada, 5% assembly language	Fundamentals			
	Tactical configuration	Database	Diagnostics	Common functions
Base System 2000: 10% Ada, 90% C	LAN		Interprocess communication (IPC)	
	OS-2000			
	Base system hardware			

Figure 8: A More Detailed Rendering of the SS2000 Layered Architecture

Applications	MMI	Fire control	Picture compilation	...
Distributed database layer	Common Object Manager (database repository)			
Base system	operating system, IPC, etc.			

Figure 9: Another Description of the Architectural Layers

The existence of more than one model of the layers is of minor operational consideration, since (except for Base System 2000) the layers do not determine job function, and a component could probably be correctly allocated to a layer in any of the above renderings. The primary quality of a layered system is that elements closer to the bottom are more likely to be applied unchanged across all members of the product line[1]. For SS2000, this includes software that directly interfaces to the physical architecture and software that implements functionality common to all systems in the domain. Elements at or near the top—for example, software implementing customer-specific requirements—are less likely to be generic. Hence, reusability is enhanced when an element uses (in the sense used by Parnas [Parnas 78]) only those elements that fall at its level or below it in the layered view of the system. Otherwise, if a component were used in a new customer product, a higher level component it relied upon may not have transferred with it, and so this otherwise reusable component would have to be rewritten.

3.4.3 Intercomponent Coordination

SS2000 is a family of distributed systems; the components must work together in parallel in a coordinated fashion. The coordination mechanisms used are those common to distributed Ada systems: asynchronous message passing with conventions to reduce network traffic. CelsiusTech designers have identified conventions for distributed applications, some of which have been adopted for SS2000 [CelsiusTech 92]. Other design conventions and protocols were divulged to us in interviews. Still others are codified in a document called the *Application Interface Standard*. Note how the conventions apply to both the distributed requirements of the architecture and its product line aspects.

- Tasking and communication conventions include the following:
  - Tasks communicate by passing strongly typed messages. The abstract data type and the manipulating programs are provided by the component passing the message. Strong typing allows for compile-time elimination of whole

1. Or even beyond. Note the application of SS2000 architecture and components to the air defense system discussed in Section 2.

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 representation of data.

  - Interprocess communication (IPC) is the protocol for data transport between Ada applications. IPC allows communication between applications regardless of where they reside 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 tasking facilities are used.

- Data-producing conventions include the following:
  - A producer of a datum does its job without knowledge of whom the consumer of that datum is. Data maintenance and update is conceptually separate from data usage. This design dictum leads to a blackboard-style repository called the Common Object Manager. The repository provides a subscription service for data-consuming components, which can ask to be notified when a datum of interest changes value. Data-producing components "write" (through access programs or "methods" in object-oriented parlance) to the repository. Data-consuming components (particularly the MMI software) retrieve required data from the repository. Figure 10 illustrates the role of the repository at runtime. Track information, carried in a very large data structure, is passed directly between producer and consumer, by-passing the repository for reasons of performance. Trackball information, because of its very high update frequency, also bypasses the repository.

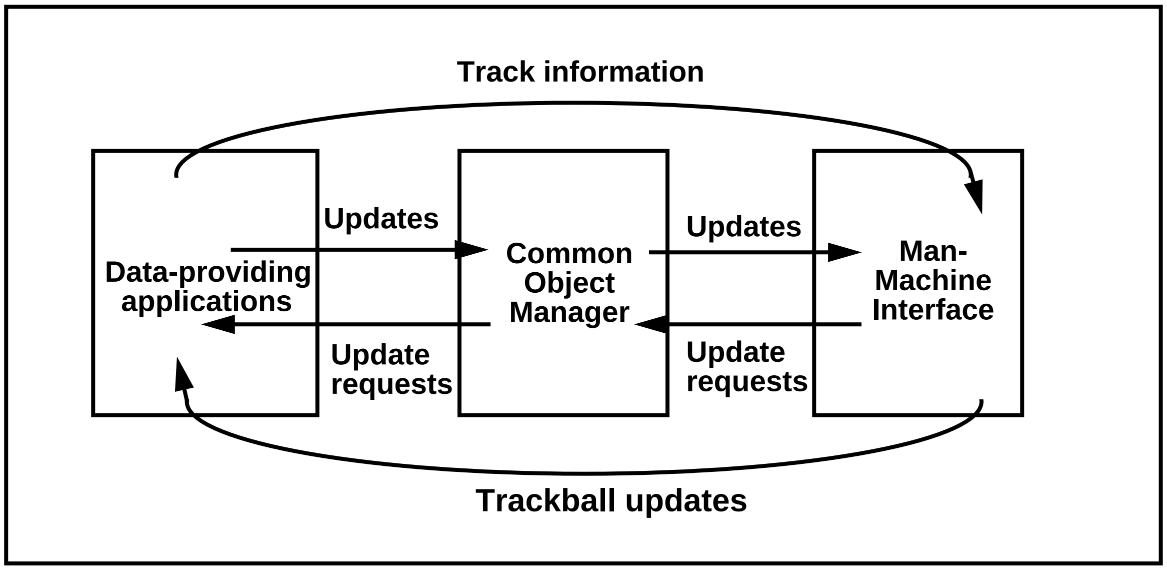


Figure 10: Using (and Bypassing) the Common Object Manager Repository

  - Data is sent only when altered. This prevents unnecessary message traffic from entering the network.

    - To insulate programs from changing implementations, data is presented as an information-hiding abstraction such as an abstract data type. Strong typing allows compile-time detection of type-related errors.
    - Components own the data they alter and supply access procedures that act as monitors. This eliminates race conditions, since each datum is accessed directly only by the component that owns it.
    - Data is accessible to that all interested parties at all nodes in a system. Assignment to a node does not affect the data to which a component has access.
    - Data is distributed so the response time to a retrieval request is short. This also has fault-tolerance implications.
    - 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 and controllable.
    - Data channels are error resistant. Applications resolve errors internally so far as possible.
    - It is acceptable for an application to “miss” a data update. For instance, since ship’s position changes in a continuous fashion, a position update may be missed and interpolated from surrounding updates.
- Miscellaneous conventions
    - Heavy use is made of Ada generics, for reusability.
    - Ada standard exception protocols are used.

Many of these conventions (particularly those regarding abstract data types, IPC, message passing, and data ownership) allow a component to be written independently of many changeable aspects over which it has no control. In other words, the components are more generic and hence more directly usable in different systems[1].

3.5 Parameterized Components

Although components are reused with no change in code in the vast majority of cases, they are not always reused entirely without change. Many of the components are written with symbolic values in place of absolute quantities that may change from customer system to customer system. For example, a computation within some component may be a function of how many processors there are; however, the number need not be known when the component is written. Therefore, that component may be written with the number of processors as a symbolic value, a parameter. The value of the parameter is bound as the system is integrated; the

[1]. A treatise on the principles of information hiding and object-oriented design is beyond the scope of this paper. See Britton et al. [Britton 81] instead.

component 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 component reuse [Britton 81]. However, in practice, they tend to multiply at an alarming rate. Almost any component can be made more generic through parameterization. The componentry for SS2000 currently features 3000-5000 parameters that must be individually tuned for each customer system built in the product line. CelsiusTech currently has little or no methodological approach to ensure that no parameters are in conflict with each other; that is, there is no formal way to ensure that a certain combination of parameter values, when instantiated into a running system, will not lead to some sort of illegal operating state.

The fact that there are so many parameters in fact undermines 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 produce a version of the system that has never before been tested. Each combination of parameter values may theoretically take the system into operating states that have never before been experienced, let alone exhaustively tested.

In practice, the multitude of parameters seems to be mostly a bookkeeping worry; there did not appear to be any experience with incorrect operation that could be traced back to the parameter situation. Often, a large component is imported with its parameter set left unchanged from its previous utilization. However, there is clearly a need for some sort of technology that will help provide assurance about which combinations of parameters will or will not result in illegal states or states that have not yet been adequately tested.

3.6 Analysis

3.6.1 Maintaining the Component Base

Although we will discuss this more fully in Section 4 and Section 5, it is important to underscore that the enduring product at CelsiusTech is not an individual ship system for a specific customer or even the set of systems deployed so far. Rather, the central task is viewed as maintaining the *product line itself*. Maintaining the product line means maintaining the reusable components in such a way that any previous member of the product line can be re-generated (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 the components up to date and generic with respect to their instantiation in individual systems. No product is allowed to evolve in isolation from the product line asset base.

Not every component 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 components that are used in a few systems. In a sense, this yields product lines within the major product line: a Swedish set of products, a Danish set of products, etc. Some components are used but once; however, 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, it develops and grows 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. We will discuss all of these issues in more detail elsewhere in this report.

3.6.2 Enabling Technology

Several technical advances have come together to help (but not enable) a product line approach to work. They include

- Memory has become cheap. This means that, unlike embedded systems of the 1970s and 1980s, dead code[1] in modern systems is not automatically a prohibitive waste of resources. Further, memory efficiency, which sometimes suffers under object-oriented implementation schemes, is not an overriding concern, and thus the designers are more free to build components that are more abstract with respect to the problem they are being used to solve.
- Off-the-shelf processors are robust and inexpensive, and network technology is stable and reliable so that adding processors to a network is not prohibitive. Again, unlike embedded systems in previous years, there is no overwhelming need in modern systems to keep the number of processors on a ship to a minimum. In addition to easing some of the performance constraints, this facilitates building a product line whose products work on different numbers of processors.
- Networks are reliable and robust, and network management systems encapsulate many of the hard issues associated with operating on a network. This facilitates building a product line whose products work on different network configurations.
- Ada allows object-oriented programming, providing a built-in infrastructure for designing generic components and specifying public interfaces and private implementations.

1. “Dead code” refers to instructions that are loaded in memory but are never executed. It may come about as the result of making a component generic with respect to the individual products in which it is used. For example

```
if ship_class = swedish_corvette then do; ... end;

else if ship_class = swedish_submarine then do; ... end;

else if ship_class = anzac_frigate then do; ... end;
```

and so forth. A component containing this code can be used, unchanged, on any of the ship classes it mentions, but on each ship class only one of the do-groups will be executed. In practice, CelsiusTech systems have little if any dead code in their fielded products; macro preprocessing can eliminate unnecessary instructions such as the ones in this example.

- Open systems are becoming the norm. Machine portability, network independence, and remote procedure call (allowing flexibility about allocating processes to processors) are becoming accepted practice. This facilitates building a product line whose products run on different computers, different operating systems, and infrastructure provided by different vendors.

3.6.3 Architecture Was the Foundation

Although we emphasize in this report that technical approaches to achieving a product line capability are insufficient without taking into account business, organizational, and process issues as well, it remains a fact that the architecture for SS2000 provided the means for achieving the product line capability. Toward this end, abstract design and layering were vital. Abstract design allowed components to be created that encapsulated changeable decisions within the boundaries of their interfaces. Where the component is used in multiple products, the changeable decisions are instantiated where possible by parameterization. Where the component may change across time as new requirements are accommodated, the changeable decisions held inside the module assure that wholesale changes to the asset base are not needed.

Note that the architecture itself, while the keystone to the whole enterprise, is a fairly pedestrian one by the standards of software engineering literature (and to a somewhat lesser extent, by the standards of current software engineering practice). The layered architecture is a classic one for operating systems, for instance, providing “layers of virtual machines” that are closer and closer to the hardware at the lower levels. Real-time embedded systems using layering and information-hiding modules (the precursor of objects) are uncommon but not unprecedented; see Parnas et al. [Parnas 85], for example.

Nevertheless, the size and complexity of this architecture and the components that populate it make clear that understanding of the application domain is required to partition a system into modules that (a) can be developed independently; (b) are appropriate for a product line whose products are as widely varied as those in SS2000; and (c) can accommodate future evolution with ease. In Section 4, we will discuss the team that created the architecture, but we note here that a thorough domain analysis is essential before embarking on the architectural design phase. In the case of CelsiusTech, the domain analysis flowed from years of previous work on shipboard computer systems; the domain models were manifested in the expertise of the architects.

3.6.4 Maintaining Large Pre-Integrated Chunks

In the classical literature on software reuse repositories, the unit of reuse is typically either a small fine-grained component (such as an Ada package, a subroutine, or an object) or a large-scale independently executing subsystem (such as a tool or a commercial stand-alone product). In the former case, the small components 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 that makes much more sense. The unit of reuse is a system function, a thread of related functionality that comprises elements from different layers in the architecture. System functions are pre-integrated—that is, the components 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 reusing components, it is also reusing the integration, component test, and unit test effort that would otherwise have to be needlessly repeated for each application.

3.6.5 Managing Configuration Parameters

The multitude of configuration parameters raises an issue that may well warrant serious attention. Formal models for parameter interaction (that would identify legal and illegal configurations) would solve the problem, but might be too powerful a solution for the problem. 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 components.

4 Organizational Factors

In this section, we report on the organizational-related practices used at CelsiusTech and our observations of their importance to the company's success with product lines. CelsiusTech's organizational structure and practices have not remained constant over the past 10 years. Our investigation identified several distinct structures. The kind of knowledge and skills required of the staff also were found to have changed during the past 10 years. In this section, we first characterize the organizational structure, then we describe the staff profiles, knowledge, and skill sets. Our analysis concludes the section.

4.1 Previous Project Organization

The previous development efforts for the naval command-and-control system (Mk2.5) were 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 11 shows the organizational structure used during the Mk2.5 period. Each functional area was in turn led by a functional area manager who had direct authority for staff resources and for all system-development activities through release to system integration. Functional area managers did not report to the project manager in terms of line responsibility, but were loaned to the project.

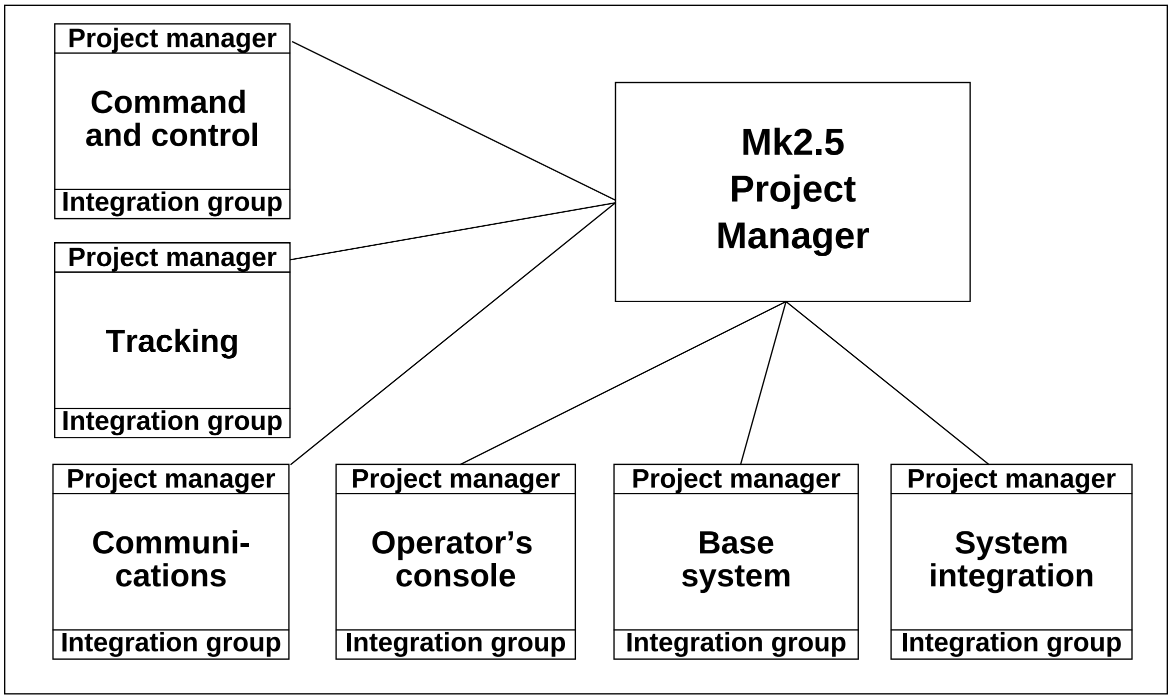


Figure 11: Mk2.5 Project Organization (1980 - 1985)

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

- assignment of major segments of the system 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. This resulted in individual interpretations of requirements and interfaces throughout design, implementation, and test
- interface incompatibilities typically not discovered until system integration, resulting in time wasted assigning responsibility and a protracted, difficult integration and installation
- functional area managers having little understanding of the other segments of the systems
- limited incentives for functional area managers to work together as a team to resolve program level issues typical of any large system development

4.2 SS2000 Organization, 1986 to 1991

With the advent of the SS2000 product line in late 1986, a number of organizational changes from the Mk2.5 project organization were put into place. Figure 12 shows the organizational structure that CelsiusTech used from 1987 to 1991 for the creation of the product line. A general program manager was designated to lead the program. He was responsible for both the creation of the product line and the delivery of customer systems built in the product line. CelsiusTech sought to remedy the problems associated with the compartmentalized structure of the past by creating a strong management team focussed on the development of the product line 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 MMI), common services (used by the functional areas), or the Base System.

A small, technically focused architecture team with total ownership and control of the architecture was created. The architecture team reported directly to the general program manager. CelsiusTech determined that the success of a product line hinged on a stable, flexible, and viable architecture, requiring visibility to and authority from the highest levels of management [Brooks 95].

CelsiusTech identified the coordinated definition and management of multiple releases as central to the creation of a product line and determined that high-level management visibility was essential. To improve support for their release management, CelsiusTech combined the software/system integration and configuration management (CM) activities into a new group, reporting directly to the general program manager.

Both the architecture team and the integration/CM group were novel approaches for CelsiusTech and factored heavily in the creation of the SS2000 product line. Each is further described below.

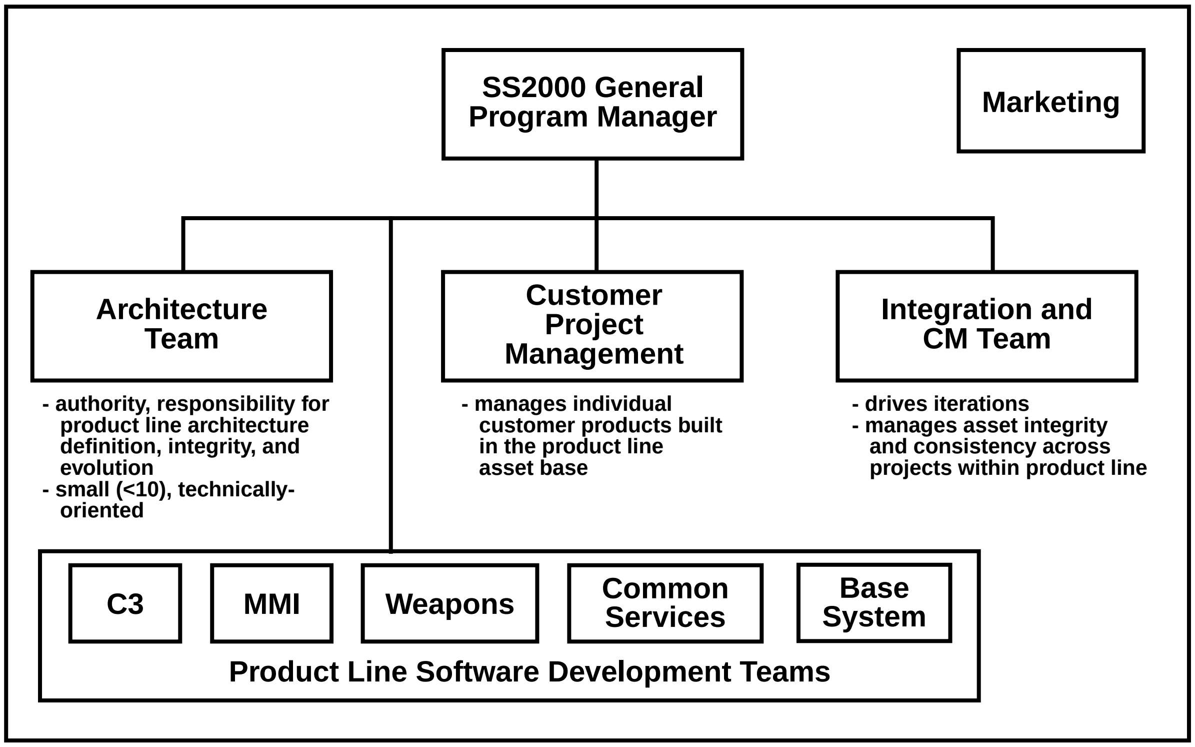

SS2000 General Program Manager
Marketing
Architecture Team
Customer Project Management
Integration and CM Team
- authority, responsibility for product line architecture definition, integrity, and evolution
- small (<10), technically-oriented
- manages individual customer products built in the product line asset base
- drives iterations
- manages asset integrity and consistency across projects within product line
C3
MMI
Weapons
Common Services
Base System
Product Line Software Development Teams

Figure 12: SS2000 Organization, 1987 - 1991

4.2.1 Architecture Team

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

- creation of the product line concepts and principles
- identification of layers and their exported interfaces
- interface definition, integrity, and controlled evolution
- allocation of capability, 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 architecture team was not a committee or engineering review board. Rather, the team was a small group (10) of full-time senior engineers with extensive domain and engineering experience. The first iteration of the architecture was created by two senior engineers over a two-week period and remains as the framework for the existing product line. The first iteration included the organizing concepts and layer definition, as well as identification of approximately

125 system functions (out of the current set of 200), their allocation to specific layers, and the principle mechanisms for distribution and communication. After completion of the first iteration, the architecture team was expanded to include the lead designers from each of the functional areas. The full architecture team then continued to expand and refine the architecture, in sharp contrast to the past when functional areas had autonomy for the design and interfaces for their area.

4.2.2 Combined Integration and Configuration Management Team

The combined integration and CM team was responsible for

- development of incremental build schedules (in conjunction with the architecture team)
- integration and release of valid subsystems
- CM of development libraries
- CM of customer release libraries
- development of test strategies, plans, and test cases beyond unit test
- coordination of all tests runs
- creation of software-delivery medium

For the SS2000 program, the integration and CM functions were centralized rather than distributed as in the Mk2.5 systems. The combined integration and CM team was not an organizational artifact in which two functions were “co-located” under the same manager. CelsiusTech redefined these functions into a fundamentally new entity to support the continual integration aspect of its development approach more effectively. Developers no longer controlled the CM of their development artifacts. Rather, the combined integration/CM team managed and controlled development and release libraries. Senior development engineers formed the core of this group. Integration and CM were not merely clerical functions, but a foundational engineering function, and the integration/CM team performed it accordingly.

4.3 SS2000 Organization 1992 to 1994

From 1992 to 1994, the emphasis increasingly shifted from the *development* of the architecture and product line components to the *composition* of new customer systems from the product line assets. 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 components. The production of components was distributed across component project areas consisting of the functional areas (weapons, C3, and MMI), common services, and the Base System 2000. Component project managers were rotated regularly, providing middle management with a broader understanding of the product line. The components are provided to the customer projects.

- customer projects that are responsible for all financial management, scheduling and planning, and technical execution from requirements analysis through delivery for their customer's product. Each customer system built in the product line was assigned a project manager who was responsible for all interactions and negotiations with the project's customer. Project managers were also required to negotiate customer requirements across all product line users.

The architecture team and the integration and CM team remained. The former became responsible for managing the disciplined evolution of the product line assets; the latter handled all test, integration, and delivery functions.

4.4 SS2000 Organization Since 1994

As CelsiusTech has completed the basic product line asset base and gained further experience using it, it has moved to the identification of more efficient ways to produce systems and tailor the product line to take advantage of newer technology and changing customer-mission needs. This resulted in the 1994 organizational structure shown in Figure 13.

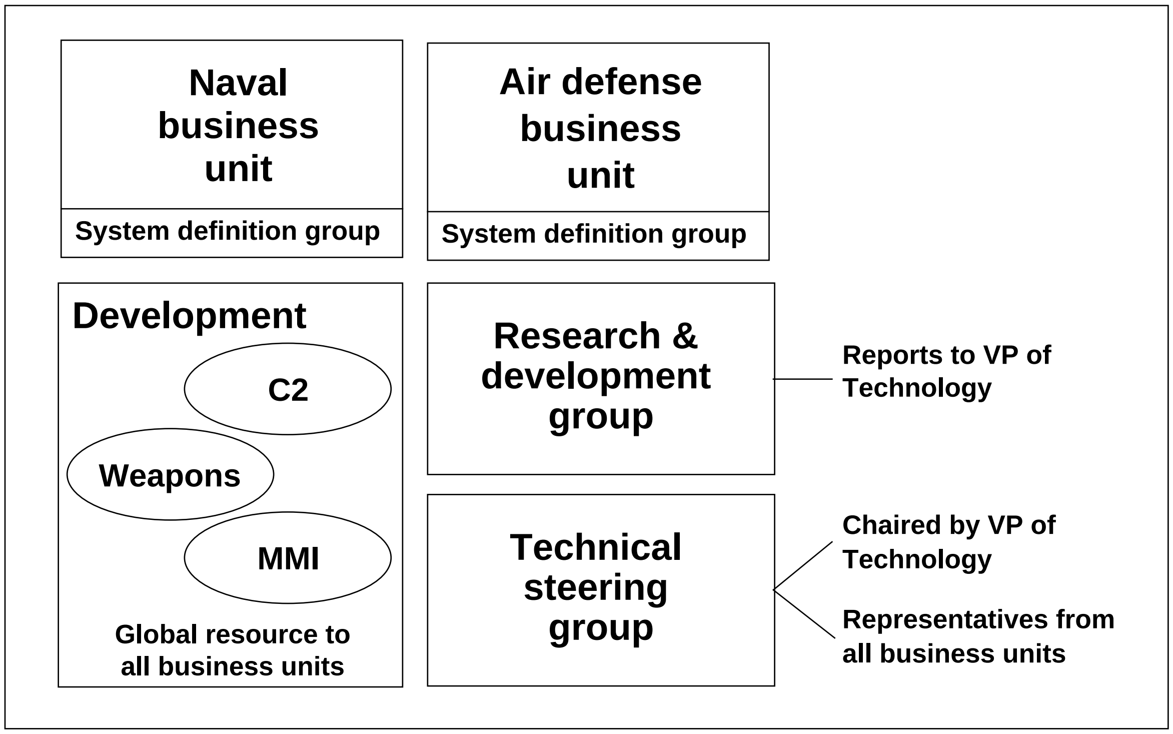

Naval business unit
System definition group
Air defense business unit
System definition group
Development
C2
Weapons
MMI
Global resource to all business units
Research & development group
Reports to VP of Technology
Technical steering group
Chaired by VP of Technology
Representatives from all business units

Figure 13: SS2000 Organization Since 1994

Each major application domain (naval and air defense[1]) became its own business unit with its own manager. Each business unit has a marketing group, proposal group, customer projects,

1. In 1992, through the acquisition of Ericsson's Command and Control division, CelsiusTech began applying the naval product line technology to the air defense domain.

and systems definition group. The business unit owns the components and customer project managers. Each business unit’s operations are guided by a set of consistent plans: marketing, product, and technical/architecture. The marketing group is responsible for the marketing plan that describes the business unit’s marketing opportunities and value of each market segment. The product plan describes the products that the business unit sells and is owned by the proposal group. The product plan implements the marketing plan. The system definition group is responsible for the technical/architecture plan for its business unit. The technical/architecture plan in turn implements the product plan, outlining the direction for evolution of the business unit’s architecture. New project proposals take into account the business unit’s product plan and technical/architecture plan. For the naval business unit, this approach keeps the projects aligned with the product line.

Components are supplied by the Development Group. Any customer-specific tailoring or development is managed from the business-unit customer project using development resources matrixed from the Development Group.

The business unit’s Systems Definition Group is responsible for the architecture. This group owns and controls the evolution of the architecture and major interfaces and mechanisms. For the Naval Business Unit, the Systems Definition Group is a small group (typically six members) of senior engineers with extensive knowledge of the naval product line. The group remains responsible for overall arbitration of customer requirements and their effect on the product line.

The Naval Business Unit has created a SS2000 Product Line Users Group to serve as a forum for shared customer experiences with the product line approach and provide direction setting for future product line evolution. The users group includes representatives from all SS2000 customers worldwide.

The Development Group provides developer resources to all business units. Integration, CM, and quality assurance resources are also a part of the Development Group and are matrixed to the business units as required. To further optimize the creation of new systems from a product line, a Basic SS2000 Configuration Project was recently formed. The goal is the creation of a basic, pre-integrated core configuration of approximately 500K SLOC, complete with documentation and test cases that would form the nucleus of a new customer system.

The Technical Steering Group (TSG) became responsible for identifying, evaluating, and piloting potential new technology beneficial to any of CelsiusTech's business areas. The TSG is headed by the vice president of technology and staffed by senior technical personnel from the naval and air defense business units, Development Group, and the R&D Group. The TSG also ensures that each Systems Definition Group creates and evolves their architecture and technology plan.

4.5 Staffing Characteristics During 1986 to 1991

Figure 14 shows the software developer staff levels from 1986 through 1996. During the formative years of the product line, 1986-1991, the software staff levels ranged from 20 to 30 initially up to a peak of 200. During these earliest years of the program when the product line concepts and architecture were being defined, CelsiusTech found that the staff levels were too high while concepts and development approaches were in a state of flux.

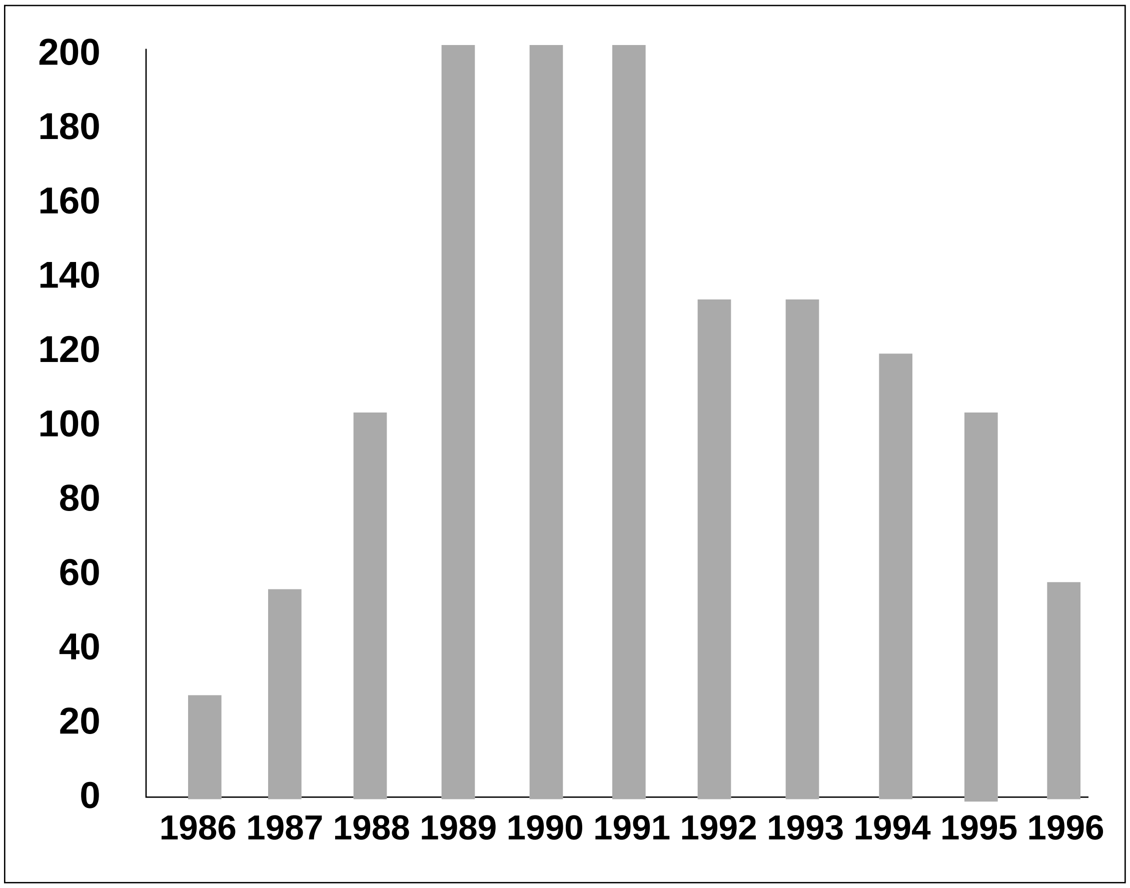


Figure 14: Approximate Software Staff Profile

4.5.1 Technical Skills and Knowledge

The architecture team was responsible for the creation of the framework and specific building codes (interface standards, design conventions, etc.) used in a product line. Team members needed solid domain and customer-usage knowledge. This knowledge was combined with excellent engineering skills and an ability to look and think broadly to find relevant common mechanisms or product line components. Communication and teaming skills were also mandatory.

Developers needed to understand the framework and building codes and how their respective components should fit. During the formative period of the product line, the development staff required skills in the use of Ada, object-based design, and their software development environment, including the target testbed. In addition, broad areas of knowledge were required: product line concepts, SS2000 architecture and mechanisms, creation of reusable components, incremental integration, and at least one functional area domain.

Integration and CM team members required a solid knowledge of the product line architecture, past domain systems experience, the Rational Environment, CelsiusTech's integration and release process, and Ada. Integration and CM were viewed as a single engineering function. The early team members had development experience with the previous systems and the SS2000 system. They were highly skilled, results-oriented engineers. Part of the responsibility of the integration and CM team was to forge an efficient process for development and customer releases. During this time, each customer system had three to five team members with responsibility for all aspects of integration, test, and CM.

4.5.2 Management Skills and Knowledge

With much of the necessary technology immature, the management team (and senior technical staff) were operating, to large degree, 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 attempting to transition immature technology encounter resistance to the technologies as the inevitable problems with the technologies arise. Key to sustaining the early phases of such transitions are strong, solutions-oriented managers. For example, the general program manager focused on finding solutions rather than finding fault with the various immature technologies, their suppliers, or the development team. He encouraged managed experimentation rather than penalizing it, and he supported technical innovations. The general program manager became a role model for other managers to follow.

Managers in the formative years of the product line required strong knowledge of the product line concepts, the technical concepts to be applied, and the business rationale for the product line approach. In addition, they needed strong planning, communication, and innovative problem-solving skills.

Management also had to cope with the inevitable discontent and resistance associated with the transition to a new business paradigm and 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 loss of domain knowledge that took time to re-develop.

4.6 Staffing Characteristics for 1992 to 1994

By the end of 1991, four customer systems were under way. Not only did reusable components exist, but CelsiusTech had delivered them as part of the two original systems. The basic product line was maturing rapidly. Instead of being built from all new components, systems were now composed from existing product line components on an increasingly routine basis. Overall software staff levels began declining (refer to Figure 14). Designers were needed less and were reassigned to other projects within the company. With the increase in parallel customer projects, more integrators were needed, although the average of three to five integrators per customer system remained steady. During this period, the number of management staff did not decrease due to the increasing number of projects.

4.6.1 Technical Skills and Knowledge

With greater emphasis on the composition of systems from the product line assets, developers needed stronger domain and SS2000 product line knowledge than during the creation of the product line. The use of Ada, object technology, and the development environment had become more routine. The integration group focus turned to the integration and release management of many parallel systems. Increasing emphasis was placed on reusing test plans and data sets across customer systems. With integration and release management now routine, more junior staff could augment and start replacing the more skilled engineers used in the formative years.

The architecture team had to maintain a solid knowledge of the product line and factor in the growing set of current and approaching customer-mission needs. Communication skills with customer project managers (for negotiation of multiple customer needs) and developers (desiring to optimize major interfaces and mechanisms) continue to be extremely important. Engineering skill to balance new needs yet preserve the overall architectural integrity are vital for team members.

4.6.2 Management Skills and Knowledge

As more of the product line was available, less emphasis on technology maturation and training was required of management. With more customer systems existing, the coordination of changing customer (existing and new) requirements across multiple customers emerged as a major management focus and 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 knowledge of the existing and anticipated future directions of the product line.

4.7 Staffing Characteristics Since 1994

Since 1994, the staffing profile has continued to change. As the product line and its use have further matured, CelsiusTech's software staff levels have declined dramatically, as Figure 14 on page 39 shows. The current software staff level is at a modest 50. CelsiusTech has used fewer designers, developers, and integrators for the two most recent customer systems. Even fewer designers are needed, potentially moving to the other business unit. The downward trend is most notable in the area of integration. For the two newest customer systems, CelsiusTech has budgeted for an integration staff of one or two per system. Continuing system composition optimizations, such as the Basic SS2000 Configuration Project, are expected to further reduce the development-related staff levels. With the continued increase in parallel customer projects, the number of management staff remains constant.

4.7.1 Technical Skills and Knowledge

Developers continue to need strong domain and SS2000 product line knowledge with the emphasis on composing systems rather than building systems.

The architecture team now must maintain a solid knowledge of the product line, current and approaching customer-mission needs, and advances in relevant technology. This knowledge must then be balanced with engineering skill as the architecture and its major interfaces and mechanisms continue to evolve. For example, CelsiusTech is currently in the process of upgrading its user-interface technology to exploit X-Windows and Motif. The architecture team has been involved in technical evaluations, prototype development of new interfaces (both for the external user and for application developers), and assessing the effect of the new technologies on the product line.

4.7.2 Management Skills and Knowledge

Management focus continues on the coordination of changing customer (existing and new) requirements across an increasing number of customers. Negotiation skills remain vital for customer project managers. Managers must also not only retain a current knowledge of the existing requirements but must increasingly be aware of anticipated future directions for the product line.

4.8 Analysis

Since 1986, CelsiusTech has evolved from a defense contractor providing custom engineered point solutions to essentially a commercial-off-the-shelf (COTS) vendor of naval systems. CelsiusTech found that the old ways of organizational structure and management were insufficient to support the emerging business model. CelsiusTech 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.

CelsiusTech's organizational changes to create the product line were heavily influenced by lessons learned from the Mk2.5 integration and management difficulties, emerging understanding of the vital nature of architecture to a product line, and awareness of the risks involved with amount of new technology.

4.8.1 Creating an Architecture

The architecture served as the foundation of the approach, both technically as well as culturally. In some sense, the architecture became the tangible thing whose creation and instantiation was the goal of all of the component work. Because of its importance, high visibility was important. A small high-powered architecture team had authority as well as the responsibility for the product line architecture. Because of this, the architecture achieved the “conceptual integrity” cited by Brooks as the key to any quality software venture [Brooks 95].

4.8.2 Maintaining an Architecture

Defining the architecture is only the beginning of achieving a foundation for a long-term development effort. Validation through prototyping and early use was 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 explicit approval of the architecture team. This approach had the full support of project management; this working arrangement was supported by the authority vested in the architecture team through the organization of the company. The architecture team thus became a centralized design authority that could not be circumvented. The conceptual integrity was maintained.

4.8.3 Creating Versus Maintaining a Product Line

The organization that is used to create a product line is different from that needed to sustain and promote the evolution of a product line. Management must 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 evolution of the product line is managed.

4.8.4 Managing Change

The effect of change on an organization of product line development is far greater than even such transition-intensive technologies such as Ada or object technology. Everything is subject to change: management, organizational roles and responsibilities, training, personnel skills and experience, development processes. A program manager must be concerned with positioning the organization for tomorrow’s potential needs as well as today’s customer requirements.

The development of the product line, its parallel use on several initial customer products, and significant new technology caused much upheaval for management and development staff. The visionary management of the initial SS2000 general project manager, we believe, was instrumental in these formative years of the product line approach. In the mid-1980s, developing a product line for such a large and varied domain with life-critical reliability and real-time performance had not yet been attempted.

Management provided incentives for long-term strategies by supporting technical innovations. Managers encouraged experimentation in management style. Throughout all the chaos in the initial years, the management focus stayed clear and on the goal. While this may appear quite esoteric, we see these visionaries as vital to the success of organizations taking on new business and technical approaches.

Managing change is the focus of Chapter 6.

5 Processes, Methods, and Tools

While some organizations treat processes, methods, and tools as distinct concepts, CelsiusTech does not. Rather, CelsiusTech takes an engineering approach and thinks about processes, methods, and tools as a synergistic whole, which we will refer to as their "development approach." In this section, we summarize the processes, methods, and tools that played a significant role in the formation and use of the SS2000 product line. We first characterize the development approach used before the development of the product line on earlier CelsiusTech systems.

5.1 Software Development and Processes Before the Product Line Approach

With the previous systems, such as the Mk2.5, a traditional waterfall life cycle following the 1978 U.S. military standard MIL-STD-1679 [DoD 78] was used. Software definition or requirements analysis, top-level design, detailed design, coding, unit testing, integration, and delivery were the sequential stages a project followed. Activities of each stage were generally completed before the subsequent stage was begun. The primary results of each stage were documents and formal reviews, particularly early in the life cycle. Design focused on the stepwise refinement of subsystems into successively smaller components ending with source-code units. Code units were integrated once all units were implemented—late in the life cycle. The result was late discovery of significant problems leading to a scenario of late delivery, cost overruns, reduced functionality, reduced quality, and dissatisfied customers.

In the early 1980s, CelsiusTech developed new systems by re-using developers who modified existing systems. Previous structures formed the basis of a new system; developers used code from previous systems with modifications as needed. In time, a new system would inherit structures, design decisions and constraints, and code, potentially from several parent systems. While this approach was conceptually similar to a product line approach, there are important differences. Each working system was designed and implemented without consideration of future uses and needs. With each successive descendant, a potentially increasing amount of the parent system required redevelopment. Maintenance was needed for each individual system rather than being leveraged across all descendants. Reuse was ad hoc and purely opportunistic. Although not product line development, this early development strategy laid the foundation for the subsequent product line paradigm.

5.2 Development Approaches, 1986 to 1990

The period 1986 through 1990 marked the time during which CelsiusTech made the business decision to create a product line and delivered the first functional system to the Danish navy for shipboard installation and sea test. Figure 15 shows a timeline for this period with key development activities. Part of the product line business decision was the evaluation of CelsiusTech's existing development processes, methods, and tools during early 1986. The technology evaluation resulted in the identification of a next-generation development infrastructure during the remainder of 1986 and early 1987. Creation of the product line architecture was the primary focus of activity during 1987, followed in 1988 through 1990 by the development of a sufficient functional set of product line components for the first shipboard installation.

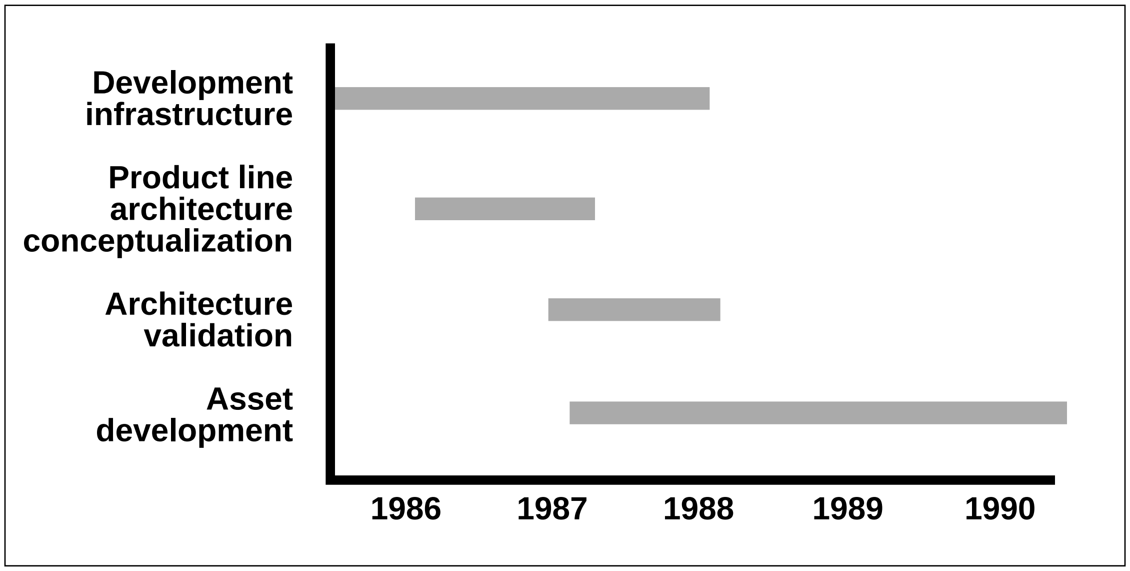


Figure 15: Timeline for Early Product Line Development

The following subsections, chronologically organized, summarize the evolution of the development approach CelsiusTech used to accomplish the activities for this period. Bear in mind that since no defined and validated development approach existed in 1986, CelsiusTech was charting new territory. CelsiusTech's development processes, methods, and tools remained dynamic in order to respond to new and changing needs. Three overarching questions guided their selection of processes, methods, or tools:

1. Did the approach contribute to achieving the business objective of product lines?
2. Did the approach contribute to maintaining a central architecture for the product line?
3. Did the approach scale to large, geographically distributed, parallel teams?

1986: Identification of development infrastructure and initiation of product line requirements analysis

The contracts for the original two systems specified the use of RTL/2, structured analysis and design, and the use of CelsiusTech's VAX-based proprietary development environment. Nothing regarding product lines was specified. With the change in business and technical strategy came the need to re-examine available technology. A small group of senior technical staff began investigating various candidate technologies to form the next-generation development infrastructure for the product line.

CelsiusTech determined that the investment in the creation of a new infrastructure would require its use for a long time. Thus, the choice of language and development environment was very important. After careful evaluation, CelsiusTech decided to use a commercially available development environment, from Rational Software Corporation, providing state-of-the-art Ada support, thus influencing the language decision. Senior technologists at CelsiusTech had followed the development of Ada and felt that it showed promise as an implementation language for large, long-lived systems.

The Rational Environment and Ada were evaluated through a pilot effort using an actual Swedish army project involving 7 developers and resulting in 55,000 lines of Ada code. The pilot not only validated the technology but also provided CelsiusTech with developers more experienced with the chosen technologies.

As part of the development infrastructure, CelsiusTech identified the need for a strong requirements-management and traceability capability for the product line assets. Processes and automated support were investigated, although limited tools were available in 1986 that were capable of supporting the size of the SS2000 program. CelsiusTech decided to use a VAX-based hypertext documentation tool to support a product line requirements database and documentation. The database would also support the traceability of requirements to specific customer system products.

In addition to the choice of language and development environment, a number of key development process issues were under consideration by senior technical and management staff but were not resolved, integrated into their development processes, or broadly disseminated. These included

- iterative development and continuous integration processes
- the development library structure and iteration-build management
- the central role of software architecture for product lines
- the possible use of object-based design
- CM and release management

During 1986, CelsiusTech started the requirements analysis at the system and software levels for the first two systems concurrently. This would form the basis of the domain analysis for the product line. The SS2000 program initially used a real-time structured analysis (SART) method developed by Ward and Mellor [Ward 85] but with limited success. CelsiusTech found SART better suited to systems with well-defined functionality and a small number of external stimuli. This was not the case for the application domain, in which CelsiusTech had a large number of events and partially specified requirements. While CelsiusTech had made a sizable investment in attempting to use the structured analysis method, it was not wasted. Although they would still need to identify the commonalities for the product line, the exercise provided the architects and designers with a better understanding of the systems they would need to build.

The year 1986 ended with the tool choices mostly completed. The life-cycle process had been established at the senior levels but not widely promulgated nor translated into details. The method choices were only partially stable. The considerable debate regarding a practical mapping of structured analysis into an object-based design had just begun.

1987: Further product line requirements analysis and creation of product line architecture

With the importance of the product line architecture conceptually established, focus on the development approach into three major (and overlapping) stages began in 1987. Product line architectural conceptualization occurred when the architects' domain knowledge was coalesced with the requirements analysis of the initial two ships, leading to the creation of the initial architecture. Product line architectural validation involved validating and refining the product line architecture by building key interfaces and mechanisms and designing the system function groups in more detail. Product line asset development followed, resulting in production of most of the product line assets.

Key activities during the architectural conceptualization stage for the product line included

- capturing domain understanding of operational capability. Although many resources were focused on the requirements analysis of the first two ships, CelsiusTech had made little progress on capturing the domain knowledge for the product line and had made no progress on forming the architecture—a case of "analysis paralysis." SS2000 program management intervened to change the current emphasis from analysis of specific ships to development of a product line architecture from which ship-specific products could be built. The first step in creating the product line architecture was to establish the initial architecture team of two people. Their first task was to identify and document the similarities and differences of operational requirements for systems in their domain, based on previous experience, the two initial systems, and potential future systems. This task resulted in a set of main operational function groups (MOFGs), categories of specialization and generalization of functionality for customers (e.g., system independent, customer dependent), and assigning of MOFGs to specialization/generalization categories.
- capturing necessary basic services or mechanisms of the domain. Based on previous system experience, the architecture team identified and documented those common services, such as radar track management or distributed process communication, required of command-and-control or weapons systems in the application domain. The identified common services added to the domain requirements for the product line.
- creating the product line requirements database. Product line requirements and customer-unique requirements were captured in the documentation database.
- identifying the product line architectural structure and unit of reuse. The early architecture team established the architecture layers (approximately 12) based on identified levels of specialization or generalization of required functionality or services for the product line and individual systems. The unit of reuse, and thus the building blocks of the product line, was based on a conceptual entity (the system function) that provided a capability recognizable to a customer. The architectural components of system functions and system function groups were established. The unit of reuse would also be the unit of CM.

- allocating system functions to specific layers in the product line architecture. The architecture team identified the basic set of system functions (125 initially). The system functions captured the product line requirements. Ada package specifications were used to capture system function interfaces.
- identifying critical interfaces between major layers of the architecture. Relationships between the basic set of system functions were established. Interfaces at major layers were identified. The critical interface definitions also used Ada package specifications.
- ensuring product line architecture and design consistency and integrity. Rational Subsystems concept and environment support were used to capture the product line architecture including the critical interfaces and relationships between the layers.

As the product line requirements and architecture, basic set of system functions, and critical interfaces were identified, the architectural validation stage began. Principal activities were

- prototyping of critical interfaces and the key common mechanisms. Since the product line capability would depend on the use of key interfaces and mechanisms, particularly those at the lower layers of the architecture, CelsiusTech determined that it would be important to stabilize and promote their maturity as quickly as possible. For example, construction on the Base System 2000, the lowest layer of the architecture, began with emphasis on characterizing and validating the distributed, fault-tolerant target hardware. Prototyping of the IPC mechanism was begun. The IPC prototypes indicated changes in the interface. Further operating-system and compiler support for the IPC were also identified.
- expanding the set of system functions. The architecture team membership grew to include senior designers from each major functional area of the product line architecture. The basic set of system functions was expanded by the architecture team to cover all mission application areas through further design. By late 1987, nearly all of the approximately 200 system functions were identified. Ada package specifications were also used to document the design within a system function.
- validating the architecture. The architecture was validated through building actual parts of the product line component asset base. A layer was developed with both a top-down perspective (to establish and extend the requirements for the layer and its interfaces) and a bottom-up perspective (to validate and build upon the interfaces of lower layers). The Rational Environment was used for all Ada development, which accounted for 97% of the total system.
- creating the documentation for the product line. Documentation was defined early as a reusable asset. CelsiusTech worked to compose new system documentation from a documentation database rather than creating the documentation for each system from scratch. Thus, documentation as well as software was created as a product line. Documentation entered around CelsiusTech's architectural components (system functions, system function groups). Requirements, design, and test documents were defined in the style of the U. S. military standard MIL-STD-2167A [DoD 88]. Appendices A and B list the documents and their relationship to typical MIL-STD-2167A reviews.
- developing the requirements and design documents for each system function group. As further system functions were identified for the mission application areas, appropriate levels of documentation were begun.
- defining and prototyping a standard incremental, continuous integration and release management process and tools. The overarching goal for integration and CM processes was to make them an integral part of the overall development process, continuously performed. First, CelsiusTech defined the mechanics of continuous integration and

release management. Second, CelsiusTech established the process of defining each of the increments or iterations. This required the cooperation and consensus of the architecture team and integration team. The integration and release management approach was prototyped, and early failures led to quick changes in the process. Celsius Tech defined unit of integration and test at the system function or system function group level, and worked, froze and distributed interfaces early in each iteration so that each developer had time to build upon the updated interfaces. The architecture team received feedback based on actual implementations. Rational Subsystems and Rational's Configuration Management and Version Control (CMVC) tools were used to support parallel team development. CelsiusTech's release management tools were extensions of the Rational Environment. Integration frequency was initially one per quarter.

Mid-year, CelsiusTech decided to shift the program to use object-based analysis and design. Structured analysis would be used in limited areas with a heavy algorithmic component. This object-based approach was based on the work of Grady Booch [Booch 83]. CelsiusTech found that an object-based approach provided a more direct mapping of elements from the real world of naval vessels to the elements in the software system resulting in a system that was easier to understand and evolve. No tool support for the Booch method existed in 1987, although several vendors now provide such tools. CelsiusTech used a MacIntosh drawing tool with standard icons defined by the architecture team to represent their designs.

1988: Continued architecture validation and asset implementation

Key interfaces in the common service layers were exercised by mission-application system functions, thus helping to stabilize and expand the interfaces and architecture. Use of the product line architecture and basic components on the Swedish system began in earnest. This provided early validation of the product line assets' suitability for use on multiple systems. CelsiusTech modified system function interfaces and implementations to ensure high levels of verbatim reuse. The architecture team controlled changes to system function interfaces.

Asset-development activities focused on iteratively designing, coding, and integrating more functionality into the product line assets. System-function-group developer teams would typically do significant analysis, followed by many iterations of design, implementation, test, and integration. Internal design and code reviews were followed early on, but as schedules became tight, code reviews became less routine.

CelsiusTech created test-case generators and managers initially, but these reportedly were not used. Documentation for the Ada units and system functions was completed as they were built. Target issues were addressed such as characterizing the target environment (compiler, runtime, operating system, target processor, network) for functionality, performance, and memory utilization.

Integration frequency rose to approximately once every six weeks as the process, tools, and trained development staff matured.

1989 - 1990: Continued asset implementation and functional system deliveries

The focus for 1989 was to complete a sufficient set of the product line components and customer-specific components for the Danish and Swedish system for shipboard installation. CelsiusTech optimized parallel integrations and release management (for the Swedish and Danish systems) processes and emphasized performance monitoring at the individual developer, subsystem, and full-system levels. 1989 concluded with the first customer delivery and installation onboard the ship for the Danish system—approximately 700K SLOC of Ada.

CelsiusTech began the third customer system in 1990 and completed shipboard installation of the first release for the Swedish system. Emphasis was on completion of the product line assets and delivery procedures.

5.3 Development Approaches, 1990 to 1994

The initial Danish and Swedish ships represented the creation of the product line, focusing on the creation and validation of both the architecture and the product line component assets. With the addition of further customer systems, CelsiusTech had new issues and concerns to address. For instance, with the addition of ships C through E, management of increasing sets of potentially conflicting requirements and priorities was necessary. An individual customer project manager could no longer agree to customers' requests without negotiating with all other customer project managers and the product line architecture team. CelsiusTech had to invent new processes to optimize these negotiations. With many of the product line assets built, creating new systems became an issue of composing rather than building from scratch. Monitoring of system performance and component-usage trends across multiple customers systems led to identification of opportunities to optimize production processes. Different development processes were now necessary.

CelsiusTech began organizing its processes around the production of a system product for a new customer and development of new or modified functionality for the product line. Documentation, reviews, code, and test cases form the principal products of this process. Controlling the configuration and consistency of documentation and code is vital to any large software-development project. For product lines, it is even more important. With a product line, documentation of the components must never be out of date, as it is potentially needed for each new customer product. A single point for updates is also mandatory for consistency and for economic reasons. Developing processes and tools to manage the configurations and the consistency requires early attention. As a result, much of the SS2000 process is defined in terms of the documents produced.

5.3.1 System Product Development

For CelsiusTech, a system product is an actual system built for a customer from a set of systems functions, including hardware, software, and documentation. The production of a new system product follows the six process phases shown in Figure 16. Each phase produces at

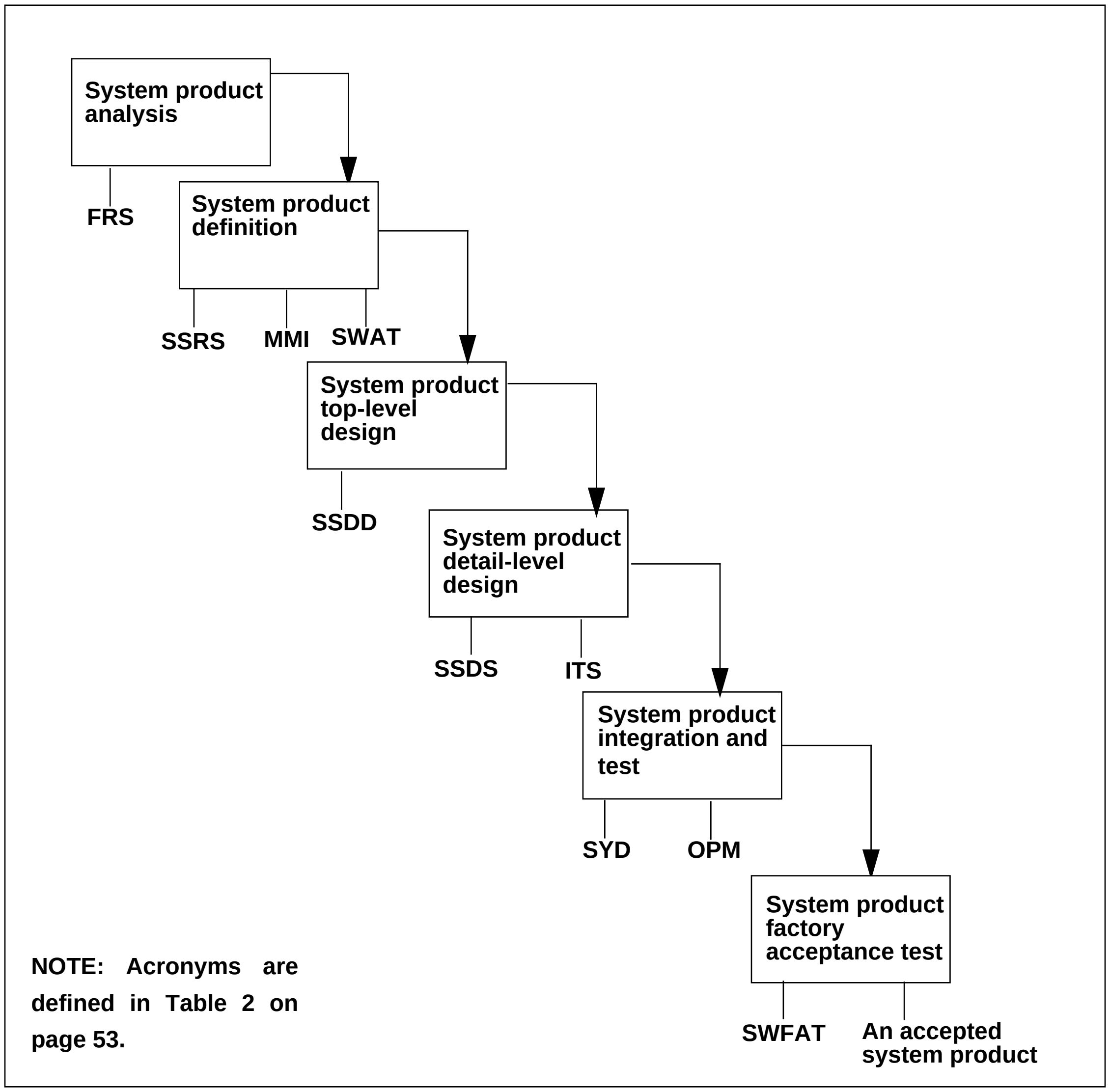

System product analysis
FRS
System product definition
SSRS
MMI
SWAT
System product top-level design
SSDD
System product detail-level design
SSDS
ITS
System product integration and test
SYD
OPM
System product factory acceptance test
SWFAT
An accepted system product
NOTE: Acronyms are defined in Table 2 on page 53.

Figure 16: System Product Development Phases Showing Documents Produced at Each Phase [Cederling 92]

least one document and concludes with a formal review. The phases are based on a typical waterfall sequence, although the traditional code, unit test, and integration phases are replaced by a single phase, system product integration and test. System product development is a process of *composing*, or integrating, components from the product line asset base rather than typical software development. This is the type of situation in which a waterfall life-cycle model can work effectively: building of a very similar system with known performance and reliability characteristics. Table 2 on page 53 summarizes the major activities and products for each phase.

At CelsiusTech, many system products may be under development or maintenance simultaneously. When a customer requests a change for a product, the customer's program manager cannot simply accede and provide the appropriate schedule and cost adjustments. The program manager must determine if the request could be provided through existing SS2000 product line components. If not, the program manager must determine if the requested capability is likely to be needed in the future by existing or new customers. If that is the case, the capability is developed with the rigor necessary for reusable components and added to the product line asset base. If the capability is determined to be unique to a customer, the capability is developed specifically for this customer with no attempt to expend the extra resources to make the resulting components reusable.

Phase	Activities	Makes use of	Produces
System product analysis	Understand customer requirements. Match customer requirements to existing product line requirements.	System family description (SFD) System function group requirements specification (SFGRS)	Final requirements specifications (FRS) (or product specification)
System product definition	Define operator interface requirements (screens, dialogue protocols, etc.). Determine hardware/software partition of functionality. Define test to verify requirements satisfied.	SFGRS requirements database	MMI System software requirements specification (SSRS) Software acceptance test specification (SWAT)
System product top-level design	Identify system functions & system function groups needed to fulfill SSRS requirements. Describe allocation of load module to hardware.	System function group design description (SFGDD)	System software design description (SSDD)
System product detailed-level design	Define & describe integration of system functions & system function groups into product. Define & describe integration verification criteria & procedures.	System software design specification (SSDS) Integration test specification (ITS)	SFGDD software design specification (SDS) SSSDD System function group user's guide System function user's guide

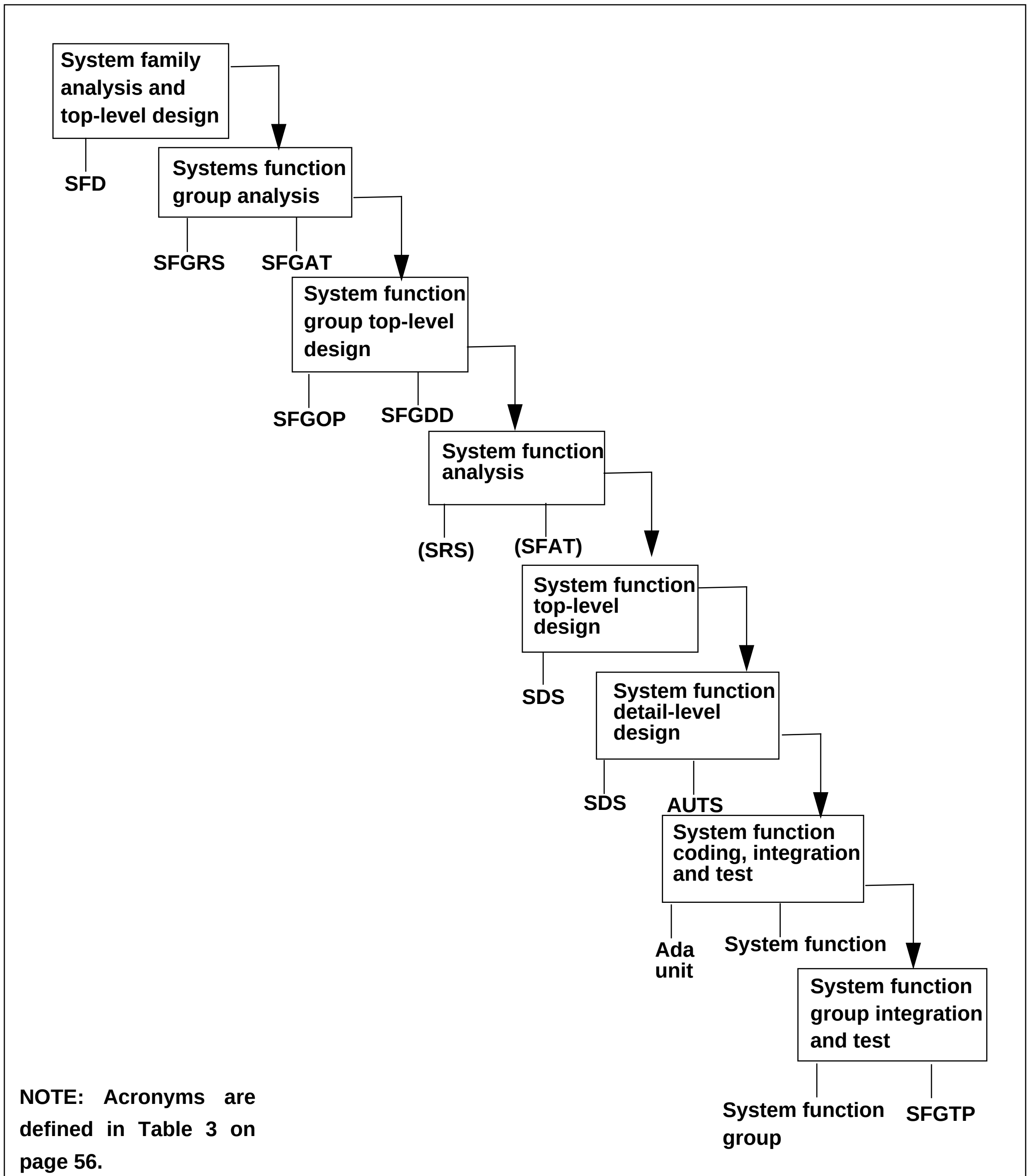
Table 2: System Product Development Phases

Phase	Activities	Makes use of	Produces
System product integration & test	Perform integration & verification. Perform software acceptance testing.	Released system product Operators manual (OPM) System description (SYD)	SSDS ITS SWAT
System product factory acceptance	Define acceptance specification for hardware, software. Build test environment. Perform system software & hardware acceptance testing. Define harbor & sea acceptance test definitions. Perform test.	Software factory acceptance test (SWFAT) Hardware factory acceptance test (HWFAT) Harbor acceptance test (HAT) Sea acceptance test (SAT)	

Table 2: System Product Development Phases (Continued)

5.3.2 Product Line Enhancements

The development of a new system function or system function group follows the process depicted in Figure 17. Each phase produces at least one document and most conclude with a formal review. The diagram reflects the hierarchy of architectural components: system family, system function group, and system function. The major activities and products are summarized in Table 3 on page 56. Most activities are typical for a large government software system. This process is cyclically repeated for each new release of the component to account for functional enhancements, performance improvements, etc. Releases happen in parallel stages: While release *n* is in the field, *n*+1 is being integrated, *n*+2 is being designed, and *n*+3 is having its requirements ironed out.


System family analysis and top-level design
SFD
Systems function group analysis
SFGRS
SFGAT
System function group top-level design
SFGOP
SFGDD
System function analysis
(SRS)
(SFAT)
System function top-level design
SDS
System function detail-level design
SDS
AUTS
System function coding, integration and test
Ada unit
System function
System function group integration and test
System function group
SFGTP

NOTE: Acronyms are defined in Table 3 on page 56.

Figure 17: System Family Development Phases [Cederling 92]

Phase	Activities	Makes use of	Documents Produced
System family analysis and top-level design	Describe overall design philosophy and function of system family.		System Family Description (SFD)
	Define SFG including interfaces, functionality.		
	Describe SS2000 architecture.		System Software Architecture (SSA)
	Describe software developers design.		Software Design Principles (SDP)
	Define principles and guidelines.		Software Design Guidelines (SDG)
System function group analysis	Analyze and describe requirements for SFG.	Requirements database	System Function Group Requirements Specification (SFGRS)
	Define test cases, test environments, test configurations.		
	Identify test case to requirements traceability.		System Function Group Acceptance Test (SFGAT) Specification
	Define overview plan for the SFG.		System Function Group Overview Plan (SFGOP)
System function group top-level design	Define & describe partitioning of SFG into system functions (SF).		System Function Group Design Description (SFGDD)
	Define & describe SFG external interface.		
	Define & describe SF interfaces.		

Table 3: System Family Development Phases

Phase	Activities	Makes use of	Documents Produced
System function analysis			(optional) Software Requirements Specification (SRS) (optional) System Function Acceptance Test (SFAT) Specifications
System function top-level design	Decompose requirements into program functions.	SFGDD or SRs	Software Design Specification (SDS)
System function detail-level design	Decompose program functions into Ada units. Define & describe Ada unit interfaces, internal program structure. Define & describe how Ada units integrate to form SF. Write test cases for Ada units.	System software (SS)	Updated SDS with ADA package specifications Ada Unit Test Specification (AUTS)
System function coding, integration & test	Code Ada units. Test Ada units. Integrate Ada units to form SF.	SDS AUTS	Integrated SF
System function group integration & test	Integrate SFs into an SFG. Test SFG.	SFGAT	System Function Group Test Protocol (SFGTP)

Table 3: System Family Development Phases (Continued)

5.4 Development Approaches Since 1994

With ships F, G, and beyond, CelsiusTech's program focus has continued to evolve. CelsiusTech is now working to further optimize its production process at a broad level. CelsiusTech currently has a project under way to define a pre-integrated, reusable core system of approximately 500,000 lines of code. This will dramatically increase the granularity of reuse. Keeping its technology base current—both for mission application areas and infrastructure—and anticipating future directions is seen as a high priority. In response, CelsiusTech has created a technical steering group under the leadership of a vice president of technology to find, evaluate, and validate technology opportunities. With the increasing size of its customer-installed base, setting the direction of evolution for the product line has become critical. CelsiusTech has encouraged the formation of an SS2000 users group to optimize its process of identifying and prioritizing its customers' emerging needs.

5.5 Analysis

CelsiusTech's development approach focused on creating a product line architecture and assets for its particular domain to meet its business objectives. This focus became the selection criteria for its processes, methods, and tools. Overall, we found a very pragmatic development approach based on sound engineering practices, such as

- early focus on the architecture, its critical interfaces and key mechanisms, and the high-level design
- continuous integration to reduce and manage the risk of pulling so many new and different components
- monitoring and analysis of performance characteristics for the deployed system components, thus forming an "as-is" system-behavior model
- balancing of identifying effective processes versus automation

5.5.1 Architecture

CelsiusTech uses architectural elements (system functions, system function groups) to describe its domain's functionality. These same conceptual pieces of the static architecture form the unit of reuse, the unit of configuration, and work management. Development processes are organized around these architectural elements. They also form the basic structure for the product line documentation. The logical consistency, or mapping, that this provides between the documents, processes, architecture, and understanding of the domain appear to improve the overall development approach and its efficiency and ease the learning curve for technical staff.

Large numbers of product line components resulted in many interfaces. Individual developers had to be concerned with interfaces—how to use them, their importance, and how to create them. Interfaces became everyone's concern.

5.5.2 Integration

The unit of integration, system test, and reuse was defined at the system-function level rather than at the Ada-package level, which is the more typical unit of reuse discussed in the literature. We find it significant that CelsiusTech found it economically infeasible to integrate large systems at the Ada-unit level.

CelsiusTech's product line assets include not only software but documentation; requirements; and test plans, cases, and data. Managing the integration and configurations for the product line assets plus all customer systems in the product line was vital to CelsiusTech's success. The integration and CM functions must support change coordination such that product line assets are maintained in coordination with all other customer projects. One principal aspect of CelsiusTech's solution in this area was the institutionalization of continuous integration rather than the more traditional all-at-once approach.

5.5.3 Tools

Tool support for the magnitude of integration and CM required, we believe, was critical. Robust integration and CM tools that support the large-scale, potentially geographically distributed development and parallel development are essential for product line development.

6 Managing the Learning Curves

CelsiusTech's turnaround from one-at-a-time systems to a product line involved education and training on the part of management and technicians. It also required re-educating the company's customers about what they could expect from the product line approach. This section discusses how CelsiusTech managed the many learning curves necessary to climb before the business could succeed.

6.1 Managing the Learning Curve for Managers

For the SS2000 program, managers needed to

- understand and support the business motivation and strategy for the SS2000 product line
- understand the role the infrastructure technologies, such as Ada, the Rational Environment, or object technology would play in achieving the product line capability
- understand how to monitor progress and identify potential problems within their area of the program
- define effective solutions to identified problems without undermining the overall corporate product line strategy
- sell their direct supervisory reports on the business and technical approaches
- handle evolution of technical and management approaches

Because of the immaturity of the infrastructure technologies and building software systems as product lines, little transition support existed for these technologies during 1986 to 1990. To help managers learn how to operate within the product line paradigm, CelsiusTech relied on flexible, motivated managers, training sessions under development for technicians, and external consulting.

Formal training was developed and oriented for technicians, although a number of the lower level managers went through portions of the technicians training program. While not oriented specifically for managers, the training gave those managers an understanding of the technologies and development approach to be used on the SS2000 program, the importance of the selected technologies, and an appreciation of the difficulty for technicians to shift to the technologies.

CelsiusTech used on-the-job mentoring from external consultants, particularly for the integration-team manager and the general program manager.

Finally, the first SS2000 general program manager was particularly adept at separating real problems from perceived problems and gave incentive to his staff to find suitable solutions without compromising the product line business strategy.

6.2 Managing the Learning Curve for Technicians

At CelsiusTech, training was used as a risk-management strategy for the transition of new technology and the creation of a new development and management culture supportive of its product line business approach. CelsiusTech's training was substantial by industry norms. Training was a management priority for all development staff and technical supervisors.

6.2.1 Initial Training Approach

In early 1987, requirements analysis, early architectural design, and design and implementation of key lower level services were under way. The early development team consisted of approximately 30 people geographically distributed across 5 sites and 2 countries. Few had actual Ada experience; many were experienced with structured programming. There was limited knowledge of object technology and experience with the new software-development environment. The product line concepts, philosophy, and architecture were not widely understood.

The initial training approach was typical at the time, consisting of

- a one-week Ada course with small hands-on programming exercises focused on the introduction of Ada syntax and constructs and the terminology and concepts of object-based design
- a three-day object-based design course composed of lecture and case studies
- a three-day hands-on software-development environment course

Courses were taught by external consultants. Both the courses and the instructors were well regarded by the participants. Several groups of the early developers attended the suite of courses and began development of the lower level common services. Early analysis of the resulting designs and code found limited use of key software engineering principles that were essential for a flexible, robust architecture and resulting product line. The developers still had a limited understanding of the architecture and associated concepts.The initial training was found suitable for transmitting concepts, features, and mechanisms, but did not help students learn how to put into practice the software engineering and object-based concepts and principles. What was needed was an education approach. CelsiusTech also found the traditional classroom performance an unreliable predictor of post-training performance on projects.

6.2.2 Expanded Re-education Approach

In response, the training was rapidly changed to a re-education approach that included

- classroom instruction for Ada and the development environment (similar to the original training strategy)
- an introduction to the SS2000 product line architecture and concepts
- a six-week practicum in software engineering using Ada
- on-the-job mentoring

The practicum was developed by an external consultant from Rational to integrate Ada, object-based analysis/design/code, and incremental development. Through a series of graduated hands-on exercises, lectures, and individual and group feedback sessions, students learned to apply the requisite concepts. Appendix C contains a more detailed description of the original practicum. The practicum became a reliable predictor of project performance; those who did well in the practicum also did well in their projects. Total re-education time, excluding on-the-job mentoring, was eight weeks.

While the new practicum went a long way toward bridging the gap between theory and practice, mentoring was found necessary to continue and reinforce the new paradigms. One of CelsiusTech's early goals was to develop an in-house training and mentoring capability. The initial practicums trained the key designers and project leaders. External consultants provided the initial training and mentoring to this core group. As the core group's experience grew, the group in turn provided the mentoring to the next groups. CelsiusTech instructors were trained and "certified" by the external consultants through a phased train-the-trainer program. This bootstrapping approach leveraged scarce resources and helped in rebuilding a common development culture.

Over 125 developers were trained between 1987 and 1989. The individual elements of the re-education program were refined and tailored to CelsiusTech's evolving methods, processes, and tools. The ANZAC project provided 70 days of training to all 50 developers new to SS2000. Similar training was used with all other customer projects. To date, CelsiusTech has trained a total of 680 developers[1] worldwide. A summary of the courses and duration is shown in Figure 18. Appendix D provides a brief description of the re-education curriculum elements.

6.2.3 Continuing Education Improvements

CelsiusTech continues to expand its training repertoire. With the emphasis shifting to the composition of new systems and the potential for building new product lines, CelsiusTech is creating an architects training program. The new program aims to develop future product line architects. One group of selected participants has gone through the 300-hour architect's course; a second group has begun the course. Course offerings were reported by several attendees as primarily theoretical at this point, lacking in criteria and rationale for the selection of an architecture.

1. While CelsiusTech has trained 680 developers since 1987, Section 4 shows a staffing level that peaked at 250 developers. The 680 includes all subsuppliers and subsidiaries. Normal attrition has necessitated additional training.

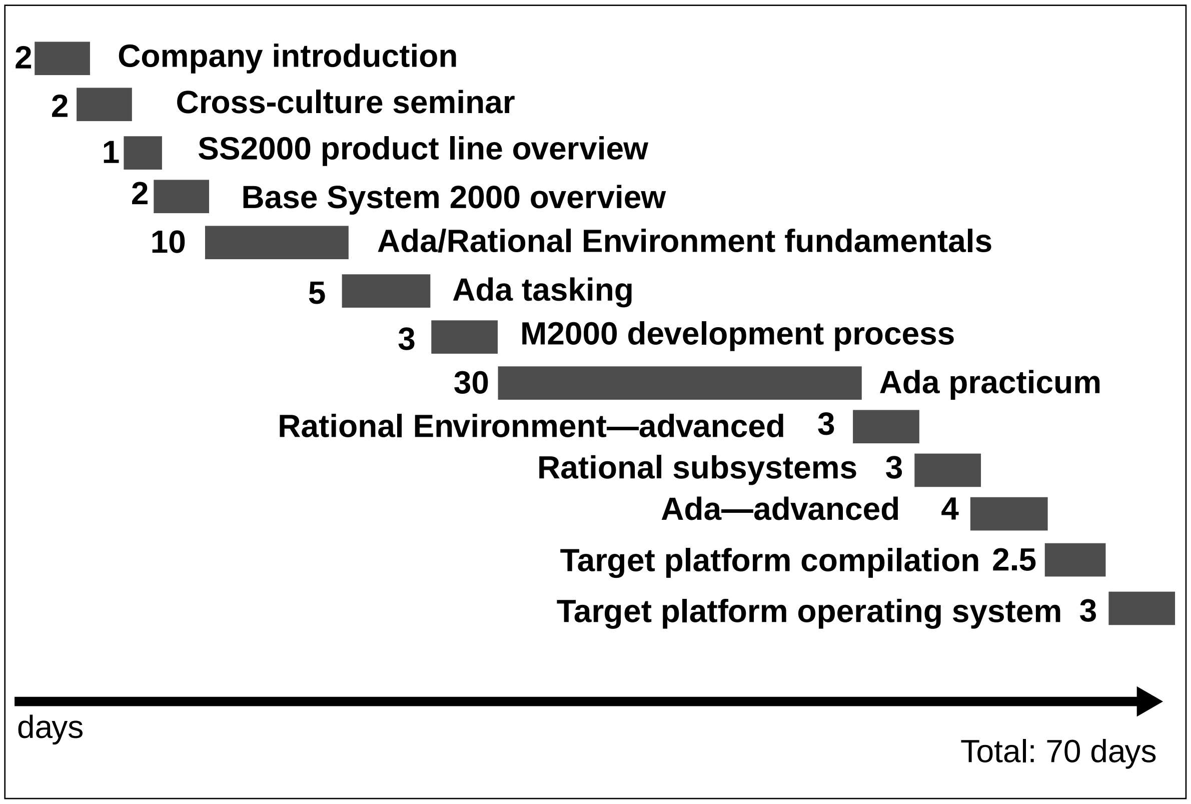


Figure 18: SS2000 Training Program

6.3 Managing the Learning Curve for the Customer

CelsiusTech's marketers and their customers were faced with the need to change the ways in which they carried out their respective roles. The following examples show the scope of changes:

- CelsiusTech customer project managers, who provide the primary liaison with customers after contract award, must negotiate with the product line organization for any requested system changes before committing to any actions.
- CelsiusTech's marketers must base new work proposals on very different cost profiles for naval systems.
- Customers must make tradeoffs between required versus desired specialization of devices and functionality.

As part of marketing its naval product line, CelsiusTech has developed numerous presentations to explain the approach and the benefits to customers. Senior technical staff members routinely were asked to assist in customer marketing situations and in new work proposals. As CelsiusTech staff members have increased their ability to explain the product line approach internally, they have become more effective in conveying their message to their customers.

The level of customer sophistication is growing. In 1995 CelsiusTech and their SS2000 product line customers formed a users group, much like the users groups that form for COTS-based products. To the participants, the users group offers the chance to jointly work out future requirements for ship systems and procure them at a lower cost than they could acquire them individually. To CelsiusTech the users group presents the chance to maintain a tightly controlled product line capability, since their customers now have the chance to migrate their systems as a collection rather than a set of disjoint products.

6.4 Analysis

In comparison with other organizations using similar technologies, the CelsiusTech training is astonishing. Some of the most comprehensive training programs provide 10 days of Ada (introductory and advanced), 10 days of object-based analysis and design (introductory and advanced), and 3 to 5 days on tools. CelsiusTech's training, both in terms of content and duration, is more representative of corporate retraining programs. For example, Motorola's software engineering curriculum is approximately 60 days covering their software development process, methodology, tools, and development language.

Currently, CelsiusTech managers budget 3% of total work hours per person per year for training for all new projects. CelsiusTech has stated that the training investment was well worth the expenditure of time and resources. From our observations, their training strategy was an effective means of transitioning new technology and ensuring the common use of a defined development process.

6.4.1 Transition Effects

CelsiusTech's transition to a product line business has been sustained since 1986 and through three changes of ownership. Sustaining any change—technological, business, or organizational—can be extremely difficult. It is important to understand how CelsiusTech has not only survived revolutionary change but triumphed. A number of factors appear to account for this success. First, Swedish business culture has a good balance between short-term and long-term gains. The culture is less oriented toward end-of-quarter results, as we sometimes see in the U.S. Second, because all CelsiusTech's contracts are fixed price, efficiency and predictability of development are even more precious commodities than if the contracts had been based on cost.

7 Summary Results

We believe that the CelsiusTech experience with product line development illustrates the organizational, technical, and economic transformation that a company should expect if embarking on a similar path. This section summarizes the main points of the CelsiusTech experience.

7.1 Stages of Product Line Development

Having an effective product line capability is not simply a matter of the right software and system architecture. The organizational structure, management practices, and personnel support are also affected, and we contend that they are a vital part of the successful creation, use, and evolution of a product line.

Since 1986, CelsiusTech Systems has evolved from a defense contractor to a COTS supplier. The old ways of organization were insufficient to support the emerging business model. We found that the changes in organizational-related issues reflect key stages in the development and use of the product line. Each product line stage has a unique focus that, when reflected in the organizational approach, enhances the ability of the organization to exploit a product line approach. First, the distinct stages we observed are

- pre-product line
- product line creation, including initial use to validate the product line
- product line routine use
- product line evolution, to include structured transformations of the product line assets and the product line production process over time, keeping the product and process current with changing customer needs and available technology

The stages of product line development are characterized in Table 4.

Product Line Stage	Characteristics of Stage
Pre-product line (prerequisite for product line)	• recognition of economic leverage • characterization of customer base • identification of business discriminators
Product line creation (and validation)	• prioritization of business and technical drivers • capture of domain knowledge • definition of architecture and unit of reuse • build, stabilization, and validation of technical and organizational infrastructure
Product line routine use	• definition of customer requirements negotiation process • definition and common use of production process
Product line evolution	• continual optimization of product process • adaptation of product applications • migration of infrastructure to newer technology

Table 4: Observed Product Line Development Stages

For each of the stages proposed above, we can postulate a set of enabling organizational attributes, such as key organizational entities, management practices, and personnel expertise. Table 5 captures our observations from CelsiusTech. Each row defines the attributes necessary to move to the next product line stage.

Product Line Stage	Key Organization Elements (Authority, Responsibility)	Management (Decision Criteria, Practices, Objectives)	Personnel (Training, Experience, Skill)
Pre-product line	• visionary decision maker • plan linking business goals and technical strategy	success criteria for profit, domain definition, business discriminators.	• domain experts • system and software architects • technical visionary
Creation	• designated architecture group with control and responsibility for domain models & architecture • strong integration and CM group	• decision criteria for priorities, validating product infrastructure, cost-benefit analysis • investment in technology transition	• domain modeling training and consulting • architecture creation and validation consulting
Routine use	designated technical project managers who arbitrate customer requirements	• negotiation minimums criteria • indicators in place and used to measure product and process	• technical negotiation • expertise in concurrent software engineering and product line engineering
Evolution	• designated group to monitor/evaluate new technology • designated owner for production process optimizations • customer user group to control evolution	• criteria for assessing added value from infrastructure modernization and adapted mission areas • criteria for improving production process • formulization of on-going business goals and technology strategy	• developers with product line experience • future architects • infrastructure experts

Table 5: Enabling Organizational Attributes for Product Line Stages

7.2 Substantial Initial Investment Required

CelsiusTech's adoption of the product line approach required a substantial upfront investment in time, capital, and resources. The actual costs are kept in confidence by the company, but they were by any measure intimidatingly large. At any point during the adoption process, CelsiusTech might have lost faith, for the outcome was by no means assured. What kept them staying the course was unwaveringly strong management—plus the realization that the old way of doing business was no longer viable.

However, it is also the case that much of the initial investment was coincidental to the product line approach. The adoption of Ada with its associated tools, environmental systems, and training accounted for a large portion of the investment required. Adopting other advanced technological foundations led to much of the rest. CelsiusTech estimates that the Base System 2000 effort contributed about 1/3 of the startup cost and provided functionality now available commercially at a fraction of the cost.

7.3 The Payoff

CelsiusTech's payoff was the technical ability to produce large, sophisticated, real-time software systems that met its customers' specifications, and to do so quickly and reliably. In short, CelsiusTech acquired for itself the ability to be agile in its marketplace, which in turns gives it a strong competitive advantage in an arena in which competition is keen and the individual products are large enough to make or break lesser companies.

Beyond this, CelsiusTech has acquired the capability to enter new markets quickly and flexibly. Its air-defense systems, taking 40% of their code directly from Ship System 2000, demonstrate this. CelsiusTech is also considering entering other markets in which multiprocess real-time sensor-based human-in-the-loop command-and-control systems play a dominant role.

In the technical sense, CelsiusTech's payoff was to reap the benefits of large-scale company-wide reuse. Verbatim code reuse on new systems averages almost 80%. A product line approach enables reuse of documentation, design decisions, personnel skills, management decisions, budgets, schedules, team structure, test plans, test cases, distribution procedures, configuration-control processes, coding standards, code, performance models, and a host of other assets across all products in the family. As a result, CelsiusTech has inverted its software/hardware cost ratio from 65:35 to 20:80.

7.4 Architecture Was the Foundation

At the heart of CelsiusTech's technical approach was the architecture for the product line. Every product was viewed in the terms of the tailoring that was required to each component in the standard architecture. The architecture served as the conceptual backbone of the product line. The architecture gave birth to the components that were created, maintained, pre-integrated, tested, and deployed in each of the products. The architecture defined the unit-testing requirements. The architecture spawned the product line-wide organizational units (for example, to maintain Base System 2000). The architecture provided the vocabulary for discussing how a new product would be produced from the standard assets. And the architecture provided the foundation for building the generic performance and process models.

The architecture could not have been successful without the architects' thorough and deep knowledge of the domain. Although formal domain-analysis methods did not exist in 1986, the architects performed the essential domain-analysis activities of identifying and capturing the corporate knowledge of their integrated weapons and command-and-control domain. The key information captured was the kinds of operational functionality, their similarities and differences, and the relationships among the pieces of functionality.

After the architecture came sound design practices rooted in principles of information hiding and encapsulation. This was the design-level key to achieving reuse.

7.5 Organizational Issues

A central theme of this study has been that technical approaches to achieving product line practices are insufficient. Organizational issues, both inside CelsiusTech and outside, played a central role in the story. As we have seen in Section 4, the company was compelled to adopt a whole new internal structure to match its technical and business practices, and in Section 6 we saw how this was reinforced with extensive and rigorous training.

Of equal interest are the organizational changes visible from outside the company, having to do with customer interaction. Marketers can no longer simply take orders, but must negotiate with the guardians of the central product line assets to see if the proposed product is a close enough fit. If not, they negotiate with the customer to trade functionality for price and reliable delivery. Customers of CelsiusTech, once separate entities with no common interest, now exist in a users group to jointly manage evolution of their ship systems and drive CelsiusTech to provide upgrades for them all at a lower price. Both sides benefit: CelsiusTech can continue to produce products that match its product line capabilities closely, and the various world navies receive their systems more economically.

Finally, we were told an anecdote that illustrates how much cultural change has occurred with CelsiusTech's new approach. The Swedish navy discovered a bug in a delivered system. CelsiusTech investigated, discovered the source of the bug, and prepared to deliver a source code change to the affected ships, after which it would upgrade its product line components to eliminate the bug in all future releases of all products. The customer balked, asking instead that CelsiusTech perform the central upgrade first. The Swedish navy, they said, would be pleased to wait for the fix until the entire product line was upgraded. It is somewhat difficult to conceive of a customer turning down a source-code-level repair just for the sake of a concept, but the Swedish navy has become such a strong proponent of the product line concept that this is precisely what happened.

7.6 CelsiusTech and the Domain Engineering / Application Engineering Life-Cycle Model

Recently several organizations have promoted an idea, rooted in the reusability community, that systems should be developed using domain analysis to generate a domain-wide model of requirements, followed by a domain-wide architecture, followed by domain-applicable software components. This activity is called *domain engineering*. A complementary activity, *application engineering*, then takes place to produce the requirements document, design, and software components for a specific member of the family. Figure 19 shows a highly simplified view of this life-cycle model in which the feedback flows have been suppressed.

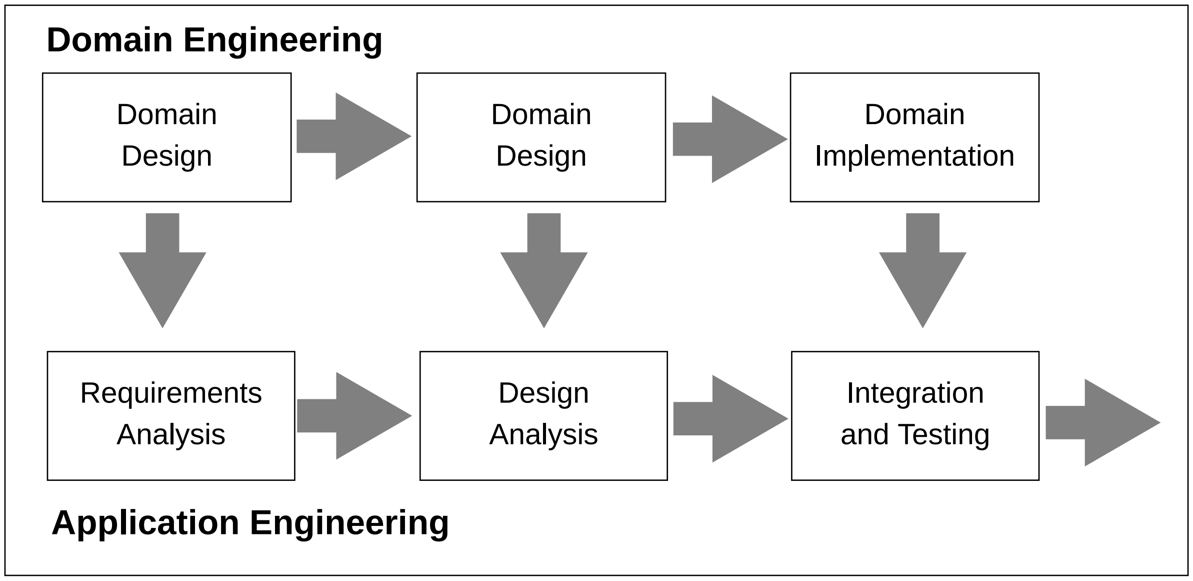


Figure 19: Domain Engineering and Application Engineering

CelsiusTech's accomplishment is in the spirit of this model, but with some important differences. First, CelsiusTech's domain artifacts were immediately and uncompromisingly constrained by the first two ship systems that it was committed to develop. This provided instant feedback as to the validity of the domain-wide assets: If they didn't apply to the two applications at hand, they certainly were not appropriate for the product line at large. This led to the second difference. CelsiusTech's application engineering was performed in concert with (and in some cases, before) its domain engineering. In particular, the family-wide requirements were developed roughly at the same time as the product line architecture and after the requirements analysis for the first two applications had been performed.

The moral is that there is more than one path through this life-cycle model, and CelsiusTech's immediate need to validate the product line assets for the two applications at hand was an important factor in its success.

7.7 Conclusion

We have tried to present, with as much fidelity and accuracy as we could, the experience of a business organization when it adopted a product line approach for large software systems. We do not know how much of CelsiusTech's experience will apply to other companies making the same decision, but we hope that others find the account useful. We believe that product lines present an opportunity for increased efficiencies and economies, more reliable and predictable and higher quality production, a more robust relationship with a company's customers, and opportunities for expansion in current and new markets. We hope that this report will help more organizations make the attempt.

Appendix A SS2000 Documentation Set

Architecture Component	Documents	
System family	SFD SSA SDP SDG	System family description System software architecture Software design principles Software design guidelines
System function group	SFGRS SFGAT SFGOP SFGDD SFGTP	System function group requirements specification System function group acceptance test System function group overview plan System function group design document System function group test protocol
System function	SRS SFAT SDS AUTS	Software requirements specification System function acceptance test Software design specification Ada unit test specification
System product	FRS MMI SSRS SWAT SSDD SSDS ITS SYD OPM SWFAT	Final requirements specification Man-machine interface System software requirements specification Software acceptance test specification System software design description System software design specification Integration test specification System description Operator manual Software factory acceptance test

Table 6: SS2000 Documentation Set, Adapted from Cederling [Cederling 92]

Appendix B Correspondence to DoD-STD 2167A Activities

DoD 2167A activity	SS2000 System Product	SS2000 System Family
System design review (SDR)	FRS, MMI	-
Software specification review (SSR)	SSRS, SWAT	SFD, SFGRS, SFGAT
Preliminary design review (PDR)	SSDD	SFGOP, SFGDD
Critical design review (CDR)	SSDS, ITS	SDS
Test readiness review (TRR)	SWFAT	SFGAT, SFGTP

Table 7: DoD-STD 2167A Reviews and CelsiusTech Documentation, Adapted from Cederling [Cederling 92]

Appendix C SS2000 Ada Practicum

The following describes the Ada practicum as it was originally conceived and delivered in 1987-1988. The practicum was a series of graduated hands-on exercises spread over six weeks using increasingly more elements of Ada, object technology, software engineering, and the development environment. The practicum comprised exercises plus lecture and group feedback sessions. Participants also received individual feedback on their exercises. Lecture and feedback sessions were held once or twice a week for two hours. The instructor was available for questions and discussion outside of the group sessions. Participants spent 75-100% of their time while the instructor devoted approximately 100%. Typical size of the practicum was 10 to 12 participants.

Exercise Description

The focus of the exercises was not on correct syntactic use of Ada. Rather, the emphasis was on the design and coding styles and project-development approaches that would enable reuse, ease of integration, and ease of evolution. Eight exercises formed the practicum. Most consisted of a small number of Ada packages. For many participants, the practicum was the first time they truly understood how Ada systems actually work. The exercise descriptions were as follows:

- Exercise 1: Review of algorithmic uses of Ada. Includes an introduction to the use of incremental development.
- Exercise 2: Implementation of abstractions. Students are introduced to abstract data types in Ada and apply the concepts to the implementation of a low-level abstraction of a small system. Testing of the abstraction is required.
- Exercise 3: Design and implementation of abstractions; using layers of abstraction. Students build the abstraction on top of the results of Exercise 2. Testing of the abstractions and the layers is required.
- Exercise 4: Design of complete small system. From a given set of requirements, students design a small Ada system. Project design notation and Ada package specifications are used to represent the designs.
- Exercise 5: Implementation of design. Following the feedback session for Exercise 4, students may redesign their small systems and then implement their designs. Documentation of all changes as a result of the feedback session or implementation is required.
- Exercise 6: Design and code review. Students critique the design and code of an instructor-provided solution to Exercise 4 and 5.
- Exercise 7: System requirement change. A further requirement is levied on the small system from Exercise 4. Students must modify their design.
- Exercise 8: Implementation of design extension. Students implement their proposed design change and evaluate alternative designs that would have eased the change.

The time allocated for each exercise is shown in Table 8.

Exercise		Duration in days
1	Review of algorithmic uses of Ada	2.5
2	Implementation of abstraction	2
3	Design and implementation of abstractions	2
4	Design of small system	3
5	Implementation of design	4
6	Design and code review	1
7	System requirement change	2
8	Implementation of design extension	4

Table 8: Estimated Student Time Per Exercise

Appendix D SS2000 Re-Education Curriculum Description

The training curriculum is designed to provide project developers and integrators with the knowledge and skills to develop and integrate new components for the product line or to compose systems from the product line asset base. The curriculum is intended to provide just-in-time training to technical members of the staff. Although the total training time to complete the courses is 70 days, students typically spread the training over a 3- to 6-month period. Following are brief descriptions of each component of the training and a possible usage scenario.

D.1 Curriculum Components

Company Introduction

- **Duration:** 2 days
- **Prerequisites:** none
- **Description:** provides participants with an overview of CelsiusTech—its business, history, organizational structure, operational practices.

Cross Culture Seminar

- **Duration:** 2 days
- **Prerequisites:** Company Introduction
- **Description:** This was specific to the Australian development team. It described the differences in the work cultures between Swedes and Australians.

SS2000 Product Line Overview

- **Duration:** 1 day
- **Prerequisites:** Company Introduction
- **Description:** provides an overview of the naval product line—its underlying concepts and purpose, static and dynamic architecture, key common mechanisms.

Base System Overview

- **Duration:** 2 days
- **Prerequisites:** SS2000 Product Line Overview
- **Description:** introduces technical staff to the role of the base system, its components and their specific capabilities, and what developers need to know to use the base system.

Ada/Rational Environment Fundamentals

- **Duration:** 10 days
- **Prerequisites:** SS2000 Product Line Overview
- **Description:** provides an intensive hands-on introduction to Ada and the basic features of the Rational Environment. Emphasis is placed on the use of Ada declarations, statements, subprograms, and packages to solve basic programming problems. Ada generics, exceptions, and tasking are introduced.

Ada Tasking

- **Duration:** 5 days
- **Prerequisites:** Ada/Rational Fundamentals
- **Description:** provides hands-on instruction to the concepts and mechanics of Ada tasking. Emphasis is placed on actual use of Ada tasking and tasking issues. (Note: Tasking is a key aspect of the interprocess communication for the product line and is used heavily throughout the system.)

SS2000 Development Process

- **Duration:** 3 days
- **Prerequisites:** SS2000 Product Line Overview, Ada/Rational Fundamentals
- **Description:** provides a detailed discussion of the processes, methods, and tools used to build customer systems and evolve the product line.

Ada Practicum

- **Duration:** 30 days
- **Prerequisites:** Ada/Rational Fundamentals
- **Description:** provides an intensive hands-on workshop to help project developers develop practical skills in the use of software engineering, Ada, object-based design, the Rational Environment, and the concepts of the product line development process.

Rational Environment - Advanced

- **Duration:** 3 days
- **Prerequisites:** Ada/Rational Fundamentals
- **Description:** provides hands-on experience with additional development environment features.

Rational Subsystems

- **Duration:** 3 days
- **Prerequisites:** Ada/Rational Fundamentals

- **Description:** provides hands-on experience with the purpose and use of Rational Subsystems and CMVC (configuration management and version control)

Ada - Advanced

- **Duration:** 4 days
- **Prerequisites:** Ada/Rational Fundamentals, Ada Tasking
- **Description:** provides hands-on experience with Ada generics and advanced aspects of typing, tasking, exceptions.

Target Platform Compilation

- **Duration:** 1.5 days
- **Prerequisites:** Ada/Rational Fundamentals, Base System Overview
- **Description:** introduces the target compilation, debugging, and testbed usage.

Target Platform Operating System

- **Duration:** 3 days
- **Prerequisites:** Ada/Rational Fundamentals
- **Description:** introduces the target platform testbed usage—target operating system, runtime, and associated tools.

D.2 Typical Usage

Table 9 shows a typical usage scenario of the various curriculum components.

Week	Course or Activity
1	Company Introduction Cross Culture Seminar SS2000 Product Line Overview
2	Ada/Rational Fundamentals
3	Ada/Rational Fundamentals
4	in project areas
5	Ada Practicum
6	Ada Practicum
7	Ada Practicum Rational Environment - Advanced
8	Ada Practicum
9	Ada Practicum

Table 9: Typical Curriculum Usage Scenario

Week	Course or Activity
10	Ada Practicum Ada Tasking
11	Ada Practicum Rational Subsystems
12	in project areas SS2000 Development Process
13	in project areas
14	in project areas
15	in project areas Ada - Advanced
16	in project areas
17	in project areas
18	in project areas
19	in project areas Target Compilation Target Operating System

Table 9: Typical Curriculum Usage Scenario (Continued)

References

[Booch 83] Booch, G.*Software Engineering with Ada*. Menlo Park, Ca.: Benjamin/Cummings, 1983.

[Britton 81] Britton, K.; Parker, R.; & Parnas, D. “A Procedure for Designing Abstract Interfaces for Device Interface Modules,” 195-204. *Proceedings, Fifth International Conference on Software Engineering*. Silver Spring, Md.: IEEE Computer Society Press, 1981.

[Brooks 95] Brooks, F. *The Mythical Man Month—Essays in Software Engineering (20th Anniversary edition)*. Reading, Ma.: Addison-Wesley, 1995.

[Cederling 92] Cederling, Ulf. *Industrial Software Development—A Case Study* (Thesis No. 348). Linköping, Sweden: Linköping University, 1992.

[CelsiusTech 92] CelsiusTech Systems AB. *Introduction to the Application Interface Standard* (Document O/AIS-100139), 1992.

[CelsiusTech 96] CelsiusTech Systems Web page [online]. Available WWW: <URL: http://world.celsiustech.se/CTSallm-E.html>, 1996.

[DoD 78] U.S. Department of Defense. *Navy Weapon System Software Development* (DoD-STD-1679), 1978.

[DoD 88] U.S. Department of Defense. *Defense System Software Development* (DoD-STD-2167-A), 1988.

[Parnas 78] Parnas, D. “Designing Software for Ease of Extension and Contraction,” 264-278. *Proceedings, Third International Conference on Software Engineering*. Silver Spring, Md.: IEEE Computer Society Press, May 1978.

[Parnas 85] Parnas, D.; Clements, P.; & Weiss, D.; “The Modular Structure of Complex Systems,” 408-417. *Proceedings, Seventh International Conference on Software Engineering*, March 1984; reprinted in *IEEE Transactions on Software Engineering SE-11* (March 1985): 259-266.

[Peterson 94] Peterson, A. & Stanley, J. *Mapping a Domain Model and Architecture to a Generic Design* (CMU/SEI-94-TR-08, ADA283747). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1994.

[Ward 85] Ward, P. & Mellor, S. *Structured Development for Real-Time Systems*. N.Y.: Yourdon Press, 1985.

UNLIMITED, UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY N/A 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release Distribution Unlimited
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CMU/SEI-96-TR-016	5. MONITORING ORGANIZATION REPORT NUMBER(S) ESC-TR-96-016

6a. NAME OF PERFORMING ORGANIZATION Software Engineering Institute	6b. OFFICE SYMBOL (if applicable) SEI	7a. NAME OF MONITORING ORGANIZATION SEI Joint Program Office
6c. ADDRESS (city, state, and zip code) Carnegie Mellon University Pittsburgh PA 15213		7b. ADDRESS (city, state, and zip code) HQ ESC/AXS 5 Eglin Street Hanscom AFB, MA 01731-2116
8a. NAME OFFUNDING/SPONSORING ORGANIZATION SEI Joint Program Office	8b. OFFICE SYMBOL (if applicable) ESC/ENS	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F19628-95-C-0003

8c. ADDRESS (city, state, and zip code))
Carnegie Mellon University
Pittsburgh PA 15213

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO	PROJECT NO.	TASK NO	WORK UNIT NO.
63756E	N/A	N/A	N/A

11. TITLE (Include Security Classification)
A Case Study in Successful Product Line Development

12. PERSONAL AUTHOR(S)
Lisa Brownsword, Paul Clements

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (year, month, day)	15. PAGE COUNT
Final	FROM TO	October 1996	95

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS (continue on reverse of necessary and identify by block number)
product line, product family, software architecture, reuse, embedded real-time systems

19. ABSTRACT (continue on reverse if necessary and identify by block number)

A product line is a set of related systems that address a market segment. Building a product line out of a common set of core assets, as opposed to building each member system separately, epitomizes reuse. Although software technology is key to achieving a product line capability, organizational and process considerations are just as crucial. This report describes the experience of one company, CelsiusTech Systems AB of Sweden, that builds large, complex, embedded, real-time shipboard command-and-control systems as a product line, developed in common from a base set of core software and organizational assets. The report describes the changes that CelsiusTech had to make to its soft-

(please turn over)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ■ SAME AS RPT □ DTIC USERS ■	21. ABSTRACT SECURITY CLASSIFICATION Unclassified, Unlimited Distribution

22a. NAME OF RESPONSIBLE INDIVIDUAL Thomas R. Miller, Lt Col, USAF	22b. TELEPHONE NUMBER (include area code) (412) 268-7631	22c. OFFICE SYMBOL ESC/ENS (SEI)

DD FORM 1473, 83 APR EDITION of 1 JAN 73 IS OBSOLETE UNLIMITED, UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

ABSTRACT — continued from page one, block 19

ware, organizational, and process structures to redirect the company toward a product line approach that yielded substantial economic and marketplace benefits to the company.