Domain-specific Engineering

Grady H. Campbell, Jr.

Prosperity Heights Software
8457 Van Court
Annandale, VA 22003

1 703 573 3139
GradyCampbell@acm.org

ABSTRACT

The development of software today is a craft industry. In time, it will become a manufacturing enterprise based on an engineering discipline. One basis for this advance is an increasing standardization within many organizations of their software products and process to form coherent, market-focused domains of capability. Following a methodology of domain-specific engineering, these organizations are developing an ability to mass produce customized software to more effectively respond to the diverse and changing needs of their customers. This trend is most evident in the increasing commercial productization of defense industry systems by organizations worldwide.

AN ENGINEERING DISCIPLINE OF SOFTWARE

The essence of engineering is the standardization of the most effective solutions to classes of similar problems. The idea of reuse, of using standardized parts and processes, is an inherent aspect of engineering. Also, in engineering, it is axiomatic that efficient construction requires the preparation of an enabling infrastructure. This enabling infrastructure, a manufacturing facility in its most complete form, must be specifically tailored to producing products of a particular type. Software development remains a craft industry because the possibility and potential benefit of domain-specific standardization that provides a basis for an engineering discipline of software are not yet widely recognized [1].

In software, most organizations still rely on manual labor aided only by primitive, overly generic tools. This state parallels a mindset that every software product is unique, primarily because each customer always seems to have different needs. However, due to the nature of software, any product is also easily replicated for dissemination to any number of customers (simplistically, a form of mass production). As a consequence of this state, the software industry requires customers either to adapt their operations to match the capabilities of a mass produced software product or to pay the full cost of software handcrafted to match their actual needs. Neither type of software is made to keep pace at a reasonable cost with the changing needs of each customer. Software, rather than providing users with greater flexibility, becomes both an inhibitor to needed change and an enforcer of unwanted change.

A new approach, mass customization, is gaining favor in manufacturing [2]. This has particular relevance for software and systems. Mass customization uses mass production techniques but manufactures each product so that, within well-defined constraints, it is tailored to suit the specific needs of a particular customer. The benefit of mass customization is that a custom product is produced for each customer with the efficiency and consistency advantages of mass production over handcrafting.

For software and systems, the concept of mass customization has emerged in the guise of domain-specific engineering, as a way to achieve manufacturing-like efficiencies. The premise of domain-specific engineering is that productivity and product quality can be significantly improved if the development process is tailored to building a specific set of similar products. An immediate side-effect of this approach, given the potential for flexibility that is characteristic of software, is that each product is customized to the needs of a particular customer.

THE SOFTWARE PROBLEM

The development of software for embedded systems demands skills both in the type of problem being solved and in the techniques of reactive processes and real-time hardware monitoring and control, in addition to many of the techniques common in less complex systems including distributed communication, data management, and user interfaces. Managers and engineers spend decades gaining the knowledge and expertise to build embedded software of a particular type. People with such expertise are scarce and the success of a project often hinges on whether the right people are available to work on it.

In software, demand for new or modified products keeps most organizations too busy to consider distracting key people with the task of creating a domain-specific infrastructure for manufacturing their software. In fact, however, domain-specific engineering is the only way an organization can both get demand under control and gain long-term competitive advantage in how they build products.

THE NATURE OF A DOMAIN

The key idea motivating a domain is that a set of problems that are described similarly can be satisfied with a set of similar solutions. The possibility of systematic reuse depends on this idea. The purpose of a domain is to provide a representation of a set of similar products and a process by which a particular solution product can be derived given only this representation and a constrained description of specific customer needs.

A domain encompasses everything about a particular set of products and how an organization builds those products. By focusing on a set of similar products, it is possible to standardize how customer needs are expressed, the form and content of solutions, and how a solution is derived to satisfy particular needs. By standardizing how needs are expressed, requirements are reduced to a set of precise decisions that distinguish all the products of this type that could be built. The set of products itself is represented by a procedure for tailoring and combining standard software parts to suit the decisions that represent a particular customer's needs. By standardizing everything related to a set of products in terms of key decisions that determine needed tailoring, the process of creating a customized product can be significantly streamlined.

ENABLING BUSINESS AND ORGANIZATIONAL CONTEXT

The business conditions that enable the use of a domain-specific engineering approach contrast sharply with the traditional assumptions of most organizations that develop embedded software. In the past, projects were formed only in response to anticipated government initiation of a complex system acquisition. The government would fully fund development to satisfy elaborate and unique requirements and the resulting software would become government property. Today with less certainty of large future government acquisition initiatives, many of these companies, including Rockwell, Thomson-CSF, and Lockheed-Martin, have started to rethink their business strategies. Many have discovered that they have valuable expertise in the development of certain products that can be sold in reasonable volume if each can be tailored to a customer's particular needs. These companies create products that provide capabilities such as satellite control or payload services, global positioning applications, operational command and control functions, and air traffic control operations.

Figure 1 illustrates a conceptual view of the business and organizational context for domain-specific engineering. The product lines within a business area of an organization each correspond to a targeted market. Each market consists of current and potential customers for the organization's products. A product line encompasses all products that the organization has, will, or could offer to a given market. As conventionally, an application project is initiated to create and support a single product that addresses a particular customer's needs. The scope of a domain in this context is a product family: the set of products within the product line which address similar needs, permitting flexible standardization of those products and an associated production process.

Just as the traditional business context was aligned to react to customer plans, the organization was a mirror of customer acquisition activity. The organization consisted of a bid-and-proposal project for each pending customer acquisition and a development project for each accepted proposal. The quality of the resulting product and enabling practices of each project depended largely on the knowledge and experience of the managers and engineers assigned to it. The larger organization had little relevance once a project was funded and staffed.

In domain-specific engineering, projects are also initiated in response to pending or awarded customer acquisitions. In addition, however, these projects are an adjunct to or extension of a permanently established domain project. A domain project is established as the means to more effectively and efficiently serve a targeted market. It represents the strategic commitment by an organization (or more narrowly, its management) to that market. Further, the domain defines a shared vision and understanding for management, marketing, and engineers of the organization’s business objectives and technical capabilities.

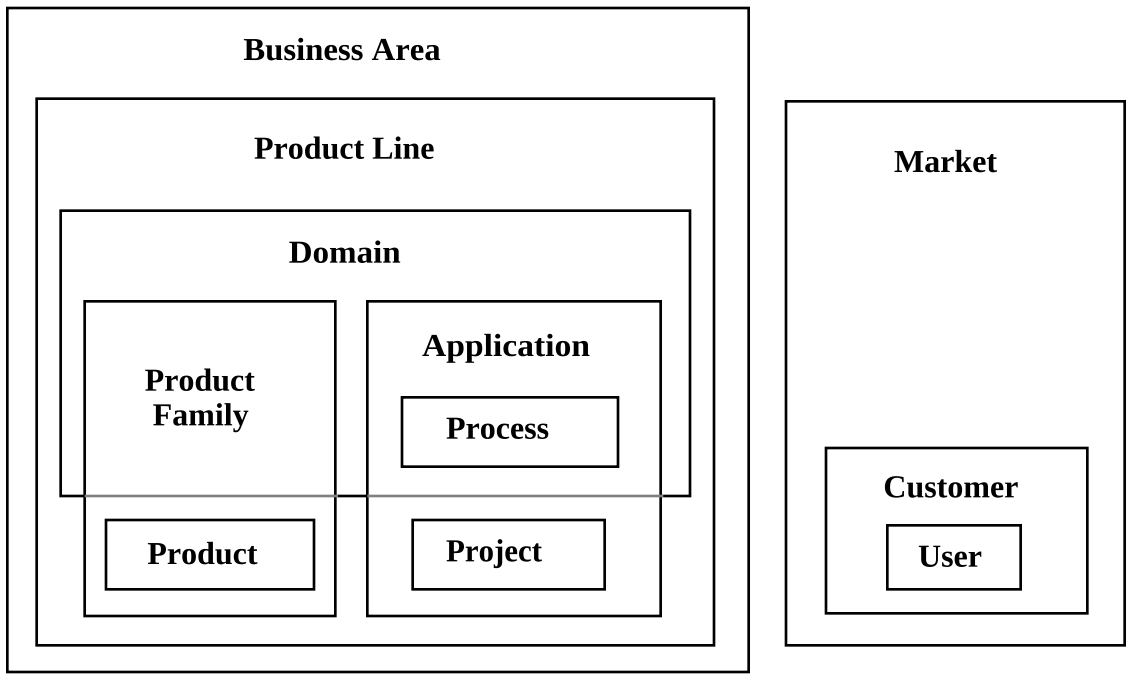

Figure 1. Business Context for Domain-specific Engineering

TWO LIFE CYCLES OF DOMAIN-SPECIFIC ENGINEERING

A software development project is chartered to deliver an application product having specific functionality, performance, and reliability on a schedule and within a budget. It cannot, within these constraints, create the assets or infrastructure needed for future reuse. Neither an individual product nor its components are a cost-effective basis for reuse; a product family [3] is the necessary and only viable basis for reuse. Because of this, a product family must have an independent existence as the product of a domain having its own lifecycle. The purpose of a domain is to enable efficient production of quality standardized application products through reuse of a product family and a standardized process. To achieve this, domain-specific engineering comprises 2 processes, domain engineering and application engineering, which interrelate as shown in Figure 2.

Life Cycle 1: A Domain

The domain engineering process is concerned with the lifecycle of a domain, including a product family and its associated process for creating instances. A domain's lifecycle, represented by the top loop in Figure 2, consists of 4 phases:

- Conception – A preliminary domain scope permits initial development and evaluation to establish a sound business and technical foundation.

- Elaboration – Capabilities are developed which establish a viable market presence by providing distinct value to a core group of key customers.

- Expansion – Diversity and complexity of producible products expands to match the total market vision for the business.

- Consolidation – Further market or technical diversification has limited benefit resulting in a stable or declining business.

When a domain, and its market vision, no longer meet the needs of the business, the organization must decide either to end support of these products or invest in a transition to a new market vision and associated domain that can adequately subsume them.

Life Cycle 2: An Application Product

The lifecycle of a single product, represented by the bottom loop in Figure 2, can be the same as it is in a conventional approach. However, as part of a domain-specific engineering approach, more flexibility is possible. In particular, the traditional distinction between development and maintenance can be largely eliminated in terms of technical practices (though not necessarily from a business perspective). The proper view, for application engineering, is that development and maintenance both comprise the derivation and evaluation of alternative product versions to find a current "best" solution for a customer's specific needs.

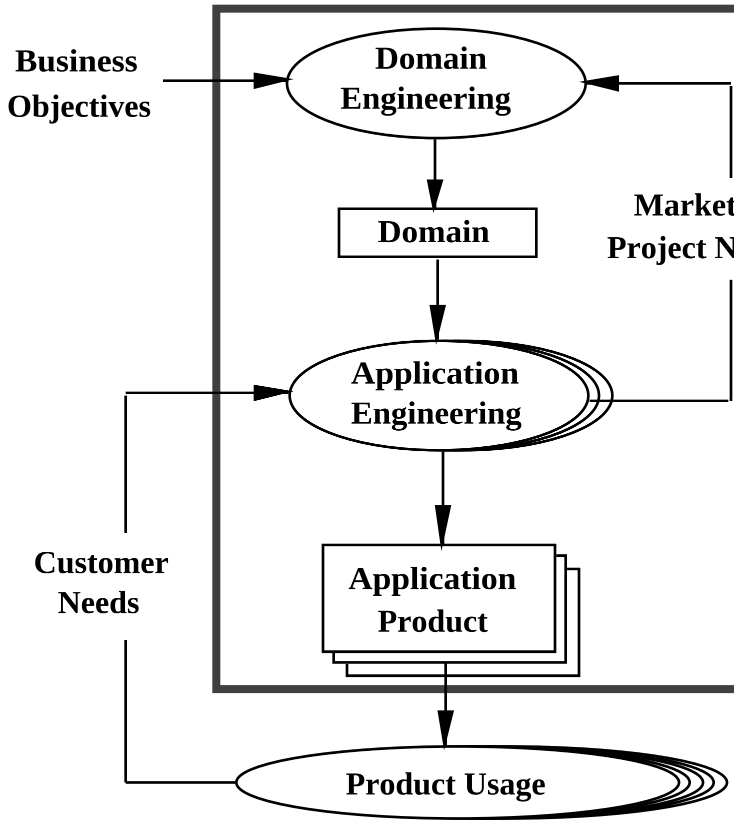

Figure 2. A Domain-specific Engineering Process

THE DOMAIN ENGINEERING PROCESS

The purpose of the domain engineering process is the creation, evolution, and support of a domain. A domain comprises a product family and an associated process infrastructure for deriving instance products. A product family concisely represents a set of products by standardizing everything that is similar about those products, reducing any differences to decisions that are sufficient to distinguish among these products. The process infrastructure comprises detailed guidance and supporting tools for making these decisions to describe a particular product and then to derive it from the product family. The domain project is responsible for creating and maintaining this product family and the associated process infrastructure. The activities of the domain engineering process are identified in Figure 3 and described briefly below. The Reuse-driven Software Processes Guidebook [4], currently the most complete definition of the domain-specific engineering methodology, provides detailed descriptions of all of these activities.

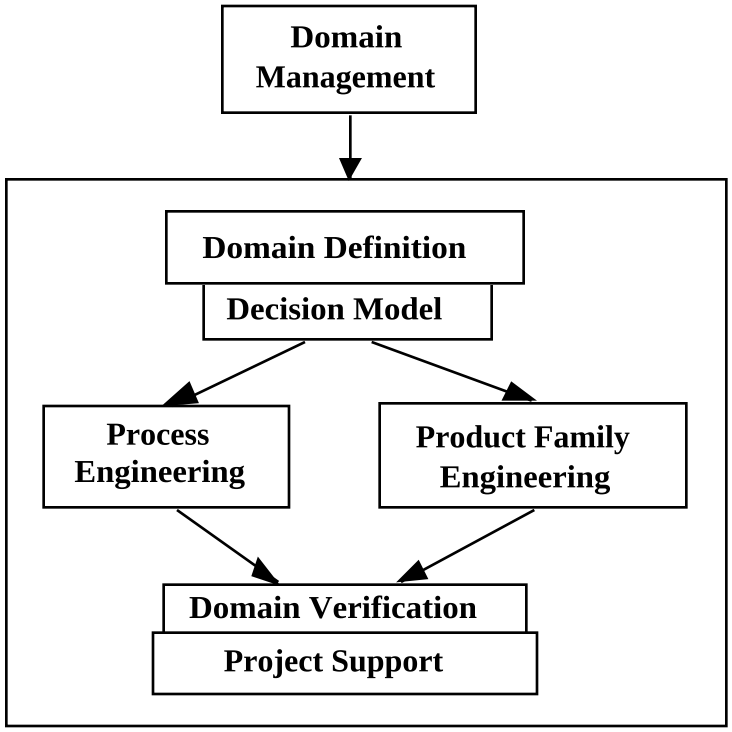

Figure 3. The Activities of Domain Engineering

Domain Management

Domain Management organizes, plans, and coordinates the domain engineering effort in keeping with business objectives delegated from organizational management. All the normal management responsibilities, such as planning, resource allocation, risk management, quality assurance, and configuration management are encompassed. The difference from conventional project management is that a domain is chartered to serve a market rather than a single customer. In contrast with a project's responsibility to a single customer, the responsibility of domain management is to weigh the mutual and competing interests of all customers, both current and potential, to achieve the best results for the business as a whole. This means that an organization's capabilities need not be suboptimized to reflect the needs of individual customers. The goal of domain management is to create a capability to build customized products at lower cost and with fewer errors, allowing each project to satisfy its responsibility more effectively.

Domain Definition

A domain definition is the equivalent of the requirements for a domain. It provides a precise synopsis of what market needs the domain is meant to address and defines shared terminology which people in the organization should use to discuss and understand problems and solutions within the domain. It establishes what products are within the scope of the domain, in terms of features, capabilities, and technologies, and which are excluded. This scoping is generalized by describing a set of assumptions about how such products are similar to each other and why different products are needed by different customers. Essential differences, or variabilities, among the set of products are the basis for defining a set of characteristic decisions that are sufficient to distinguish any product from all the others and, therefore, to allow a particular product within the domain to be described precisely. A decision model is a formal definition of these decisions and is an essential basis for creating a product family and associated application engineering process.

Product Family Engineering

A product family is a unified representation of a set of similar products. A domain focuses on a set of similar problems. A basic premise of domain-specific engineering is that similar problems are amenable to similar solutions. The domain definition establishes a proper, problem-level basis of similarity upon which a product family can then be constructed.

The most effective approach to creating a product family today has been to generalize the conventional way in which individual products are created. The basic idea of this, using whatever methods are commonly used within each organization, is that:

- A requirements specification activity establishes the necessary capabilities of the product and constraints on its construction,
- A design activity establishes a structural, static and dynamic dependency, architecture for the product and defines the responsibilities and capabilities of each of a corresponding set of components,
- An implementation activity provides an integrated realization of the set of components that satisfies the requirements and design,
- A verification activity ensures that the product has been constructed properly,
- A validation activity ensures that the product as constructed satisfies the customer's actual needs.

The key difference from conventional practice is that all work products, whether documents, code, test materials, or other types of asset, are an adaptable composition of adaptable components. The decision model for the domain defines the range of adaptability to which the work products comprising the complete product family should be susceptible. Two ways in which an adaptable component, or component family, could be represented are illustrated in Figure 4. In the first form, corresponding to a conventional reuse library conception, each component implicitly represents some unique combination of resolved decisions. In the other form, variability is represented explicitly in the form of a component family definition and a mechanism for deriving instance components given a set of resolved decisions. The most important difference between these is that, relatively, the first is cheap to create and expensive to use while the second is expensive to create and cheap to use [5].

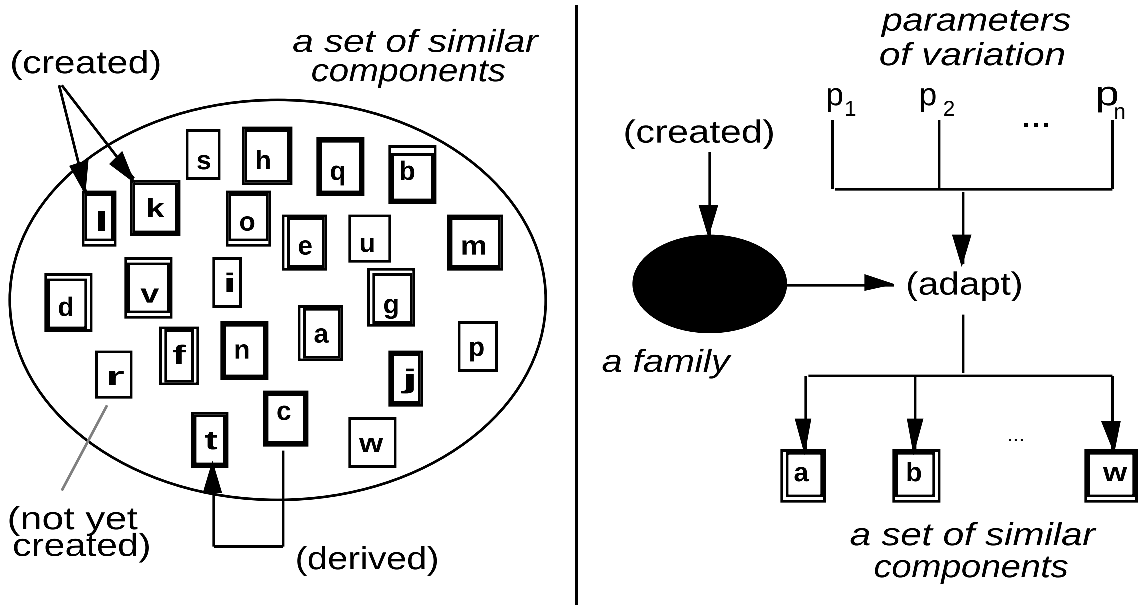

Figure 4. Two Conceptions of a Component Family

Process Engineering

The purpose of process engineering is to establish standards, procedures, and a tool-supported infrastructure to achieve a streamlined application engineering process across all projects that are within the scope of the domain. The goal of this effort is to reduce application engineering, as much as possible, to the resolution of key decisions about the needed product and mechanical derivation of that product, guided by those decisions, from the product family. A key concern is the tradeoff between the difficulties inherent in changing how projects work currently and the benefits attainable from such changes as seem feasible given the scope of the domain.

Project Support

The responsibility of project support is to ensure that a domain capability meets the actual needs of the organization’s application engineering projects. This takes two forms:

- Validating through trial use that a domain can be used as intended by a project. A preceding adjunct role is domain verification to check for consistency among the various parts of the domain product and to the domain plan to ensure that the domain is technically sound.

- Working directly with client projects to ensure effective use and continuing improvement of domain capabilities. This includes delivering domain guidance and infrastructure for the use of each project, providing project managers and engineers with training and assistance in its use, and obtaining feedback from the projects on needed technical improvements and future customer needs.

CREATING AN APPLICATION ENGINEERING PROCESS

The purpose of the application engineering process is the creation, modification, and support of one application product for one customer. In a conventional development approach, a product is constructed by painstakingly writing documents and composing executable code from a primitive set of instructions into a meaningful whole. When the customer's needs change, all relevant parts of the documents and code must be understood and modified with great care.

In a domain-specific approach, every well-defined problem has exactly one solution and all problems and their solutions are assumed to be expressible in a standardized form which domain engineering has defined. A set of decisions that characterize the domain guide how problems are expressed and how the representation of the product family can be adapted to accurately represent a particular problem and its solution. The decisions then guide mechanical derivation of the corresponding product from the product family. When the customer's needs change, the decisions are modified accordingly. The product is then rederived to reflect this new problem understanding.

Domain-specific engineering does not prescribe a single view of how application engineering should work in detail. Instead, it guides the definition by domain engineering of a process that is tailored to the needs of each organization. As a framework for this, domain-specific engineering identifies 4 levels of process capability represented by key factors of the Reuse Capability Model [6]. Each of these levels characterizes an alternative view of how domain and application engineering can work and interrelate.

Opportunistic Level of Capability

- Application projects are independently managed.

- The domain works to provide value to one or more current projects.

- A product family is represented by a well-defined set of work product components.

- The application process is conventional with streamlining of selected activities.

Integrated Level of Capability

- Domain and application projects are cooperatively managed for coordinated planning.
- The domain works to provide a shared capability of value to all current and future projects.
- The product family is represented by a set of adaptable integrated work products.
- The application process is streamlined to focus on variabilities that distinguish work product instances.

Leveraged Level of Capability

- Application projects are managed to maximize use of current and planned domain capabilities.
- The domain works to satisfy the needs of all current customers.
- The product family is represented by an adaptable integrated product.
- The application process is streamlined to focus on variabilities that distinguish products.

Anticipating Level of Capability

- Application projects are chartered by and serve as market agents for the domain.
- The domain works to satisfy the current and future needs of the targeted market.
- The product family is represented by an evolving adaptable product.
- The application process is optimized to the iterative refinement of a product model and derivation of a customized product.

Applying a Targeted Level of Capability

An organization should target a level of process capability that best supports its business objectives for a domain, consistent with its technical capabilities. Given a targeted level of capability, domain engineering develops a domain capability and supporting infrastructure with which application engineering projects can work. Figure 5 depicts an advanced level of application engineering capability for using decisions to describe a problem and mechanically derive a corresponding product. This example is provided specifically to show the potential contrast with a more conventional work product oriented, single-product process. In actual practice, an application engineering process can alternatively take the form of a conventional but reuse-enhanced process or be a hybrid that focuses improvements on how certain high-payoff portions of a product or particular work products are created with the remainder to be created using conventional techniques.

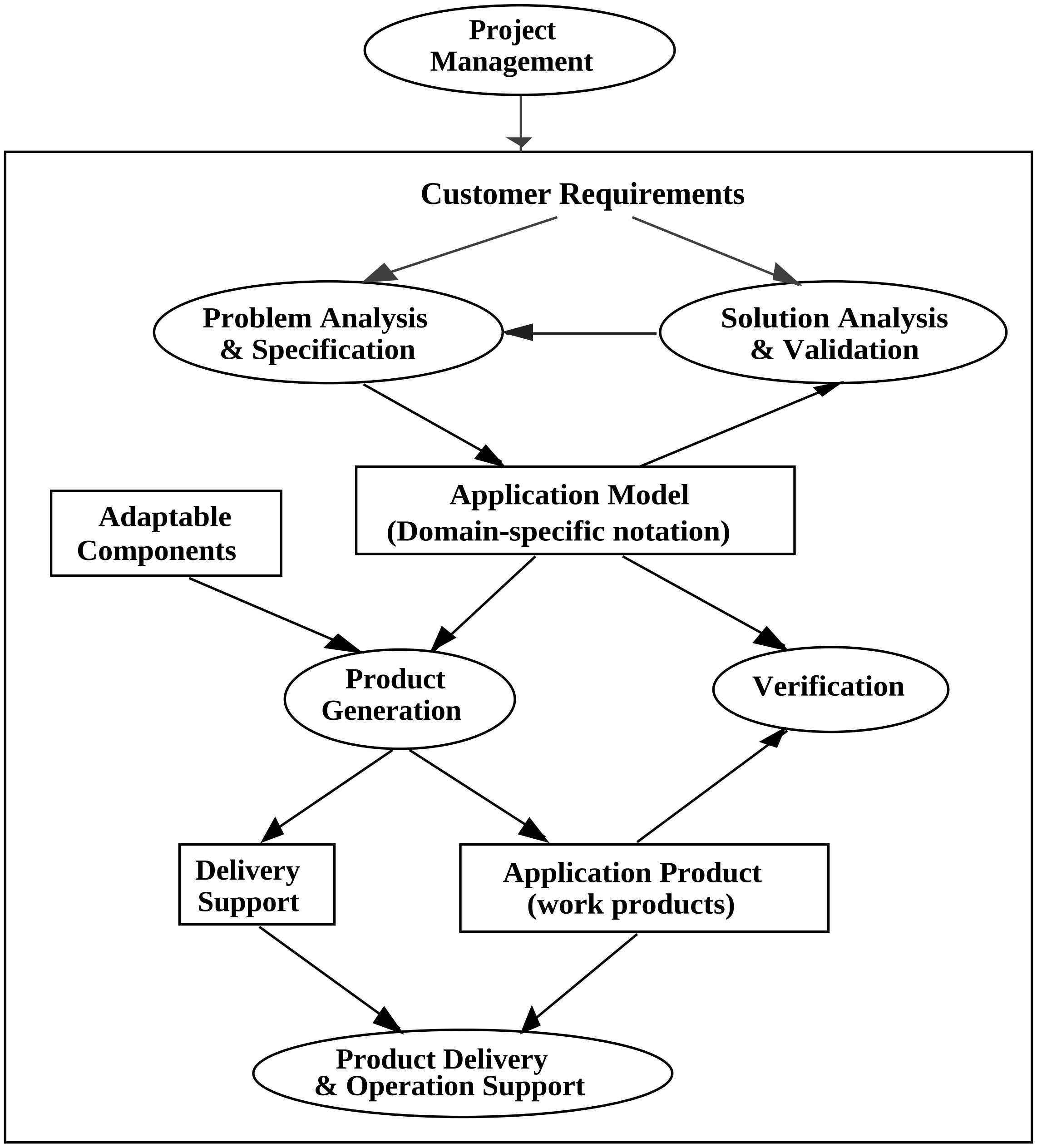
Project Management
Customer Requirements
Problem Analysis & Specification
Solution Analysis & Validation
Adaptable Components
Application Model (Domain-specific notation)
Product Generation
Verification
Delivery Support
Application Product (work products)
Product Delivery & Operation Support

Figure 5. An Advanced Application Engineering Process

PROCESS ADOPTION AND IMPROVEMENT

The actions necessary to achieve institutionalization of an effective practice of domain-specific engineering reflect the experience from other process improvement initiatives. Aspects of strategic business planning, business process reengineering, continuous quality improvement, process capability maturity [7], and technology insertion are all important in initiating and improving an effective domain-specific engineering effort. Adding the concepts of reuse capability [6] and integrating these into a coherent whole provides an effective concept of process adoption and improvement for domain-specific engineering [8]. An added facet of adopting domain-specific engineering is targeting an appropriate level of capability, described above, and tailoring it to fit the organization's business and technical needs. As a further incentive and guide to adoption, domain-specific engineering directly responds to the new Organization Product Alignment key process area at level 4 of the version 2.0 Software Capability Maturity Model (SW–CMM).

SUMMARY

For many years, the complexity of embedded software caused developers to resist the use of compiler technology. Their well-founded concern was that compilers could not produce code that was as reliable or efficient as handwritten assembly code. In contrast today at such companies, development groups not only use off-the-shelf compiler technology to build embedded software; they are following a systematic path of domain-specific engineering to better productivity and product quality. These organizations are beginning to create and use a domain-specific infrastructure which enables the rapid manufacture of customized software products to address a family of similar problems. Increasing market demands and the challenges of creating embedded software provide at the same time a strong business rationale to stop wasting effort recreating similar solutions to complex problems. Instead, these organizations are creating standardized product families and processes which they can use to produce customized solutions to better serve the needs of their markets.

REFERENCES

1. Michael Jackson, “Problems, Methods, and Specialization,” *IEEE Software* 11 (6), 1994, 57-62.

2. James H. Gilmore and B. Joseph Pine II, “The Four Faces of Mass Customization,” Harvard Business Review, January-February 1997, 91-101.

3. David L. Parnas, “On the Design and Development of Program Families,” *IEEE Trans. Software Eng.* SE-2, 1976, 1-9.

4. Software Productivity Consortium, *Reuse-driven Software Processes Guidebook*, SPC-92019-CMC, Version 2.0, Herndon, VA, 1993.

5. Grady Campbell, “Abstraction-Based Reuse Repositories,” *Proc AIAA Computers in Aerospace VII Conference,* 1989, 368-373.

6. Software Productivity Consortium, *Reuse Adoption Guidebook*, SPC-92051-CMC, Version 2.0, Herndon, VA, 1993.

7. Software Engineering Institute, *Capability Maturity Model for Software*, CMU/SEI-93-TR-024, Version 1.1, Pittsburgh, PA, 1993.

8. Grady H. Campbell, Jr., “A Unified Approach to the CMM and RCM for RSP Adopters,” Prosperity Heights Software, Annandale, VA, 1997.

BIOGRAPHY

Grady Campbell created the Reuse-driven Software Processes (AKA Synthesis) methodology at the Software Productivity Consortium. Through Prosperity Heights Software, he continues to refine, promote, and apply the ideas of domain-specific engineering for software and systems. Previously, he designed and led development of the Spectrum application generation environment and the TRF metaprogramming tool for abstraction-based reuse. A defining point of his 25 years in the software industry was his participation in the NRL Software Cost Reduction project.