Renewing the Product Line Vision

Grady H. Campbell, Jr.
Prosperity Heights Software; Software Engineering Institute (ASP)
gradycampbell@domain-specific.com

Abstract

Twenty years ago, an effort to build a reuse-based software generator led to the realization that a domain-specific focus was essential to achieving effective software reuse. This realization became the product line vision. However, reuse was never the primary focus of this vision but only a means to an end: achieving the ability to rapidly produce and evolve high-quality software. Although the spirit of this vision guides us still, much of current product line work assumes a formulation that is limited from the perspective of the original vision. A look back at neglected aspects of that vision will suggest opportunities for greater progress. Beyond that, consideration of an emerging "producibility" vision will provide a broader perspective for framing future product line efforts.

1. The Vision and its Origins

In the early 1990's, a project of the Software Productivity Consortium created the first comprehensive software product line methodology. The resulting Reuse-driven Software Processes (RSP) Guidebook [1] defined a product line as "a collection of (existing and potential) products that address a designated business area." This definition was introduced to help people understand why the traditional conception of software development as building one-of-a-kind and one-size-fits-all products might not in fact be the best fit to the actual need in all cases. If an organization was going to build a set of similar products, why did it make sense to build them all as if they were unique as the traditional process suggests? Many developers had long held a view that there was substantial redundancy and reinvention in many development efforts and that the traditional paradigm of handcrafting software to satisfy seemingly unique requirements obscured this and resulted in wasted effort. Having the concept of a product line raised the possibility that there was a need to be able to produce distinct but similar products that could satisfy differing or changing customer needs.

Product lines were already a familiar concept in manufacturing and marketing and represented the reality that distinct products are often meant to satisfy similar needs even if built independently. RSP introduced the concept of a product family to represent the idea that the products of a product line ought to be alike in their construction, differing only as needed to satisfy differing needs (as suggested by Dijkstra [2] and Parnas [3]). As explained by Dijkstra: "I prefer to regard a program not so much as an isolated object, but rather as a member of a family of 'related programs'.... We can think about related programs either as alternative programs for the same task or as similar programs for similar tasks." By assuming that products would be similar, they would be expected not only to be similar in structure and composition but also to be built using a streamlined development process reflecting the elimination of redundant efforts.

The RSP guidebook described its focus as "a methodology for the construction of software [products] as instances of a family of [products] that have similar descriptions." The idea was that quality customized products for a product line business could be built much more rapidly if, as Dijkstra argued, they were conceived of as instances of a family of similar products. This perspective had come from experience five years before in which an attempt to build a general-purpose software generator had been ground down under the burden of indefinite scope and the resulting constant need to add new capabilities in order to build new products [7]. The resulting observation was that a narrow focus on the future needs of a coherently defined business domain (as suggested by Mark Simos) would limit the scope and hence bound the cost of creating a viable software generator. In addition, the knowledge and expertise needed to build such a generator could be realistically scoped to reflect the resources of a manageable organization. Such a focus also promised to provide a basis for significantly raising the level at which problems could be specified, approximating the way that customers for such products described their needs [8].

In the years following completion of the RSP Guidebook, the Domain-specific Engineering (DsE) methodology was developed to refine and extend the RSP approach [5]. A comprehensive course, tutorials addressing specialized methods, tools for adaptable software, and various topical reports, all publicly available, documented the DsE methodology for adopting and implementing a market-focused product line approach (Figure 1).

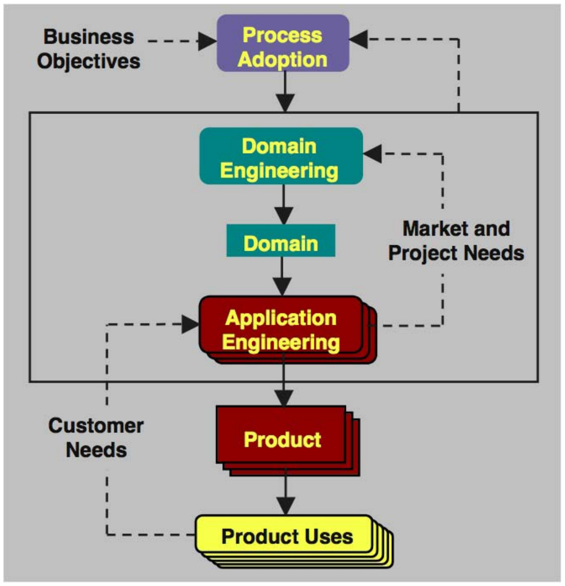

Figure 1. Domain-specific Engineering

What is discovered with a retrospective look at these materials is that several aspects of the original product line vision are not well addressed in much of the discussion of product line concepts today. Of the several books on software product lines written in the last 10 years (most notably including [11], [12], and [13]), all provide good introductions, guidance, and foundations for a product line approach but all present a simplified view of the product line concept by neglecting some important elements encompassed by the original vision. Publications concerning current research in product lines (notably [14]) provide a good sense of where the community is focused and what are considered the key issues and promising avenues for progress. A better understanding of missing aspects of the original vision and the motivations behind them may suggest opportunities for changes in emphasis that will provide a faster path to achieving the potential of the product line concept.

2. Neglected Aspects of the Vision

The product line vision was that a domain could be conceived based on a business need, formulated as a product family and associated production process, and used by engineers to rapidly build and evolve customized products for use by customers as their needs changed. The RSP guidebook prescribed a methodology for how to implement this vision. It described this methodology as having four distinguishing features:

- A focus on a domain represented as a family of products, all being similar but differing in well-defined ways
- Product building reduced to the resolution of decisions that corresponded entirely to the ways in which products could differ
- Dependence on the mechanical derivation of tailored components from an adaptable form of reusable assets for the construction of all work products
- The use of model-based analyses of the structure and composition of products as a guide to identifying and evaluating alternatives

Although the spirit of this vision is fundamental to product line efforts today, most approaches today are much weaker in their capabilities than this vision suggests. The RSP guidebook described a hierarchy of capability that a product line process might exhibit, essentially characterizing four canonical levels, labeled as opportunistic, integrated, leveraged, and anticipating. Most efforts today focus on the opportunistic and integrated levels of capability without apparent appreciation of the potential offered at the higher levels.

The following discussion identifies six aspects of the original product line vision that seem neglected today. The objective of this characterization is not to critique the potential value of other current efforts nor to suggest that all of the described aspects have to be accepted as essential. However, it should be appreciated that these aspects were part of the original vision and that the community may benefit from weighing their value in pursuing future advances.

2.1. Aspect 1: A Decision Model Formulation of Commonality and Variability

Commonality and variability is a central focus of product line discussions. Similarly, the idea of variation points in concrete work products as a localized realization of variability is familiar. What is missing is the link between these two ideas. RSP defined the concept of a "decision model". The decision model was a critical organizing element of RSP. It was a canonical formalization of the assumptions of commonality and variability upon which a domain is formulated as a product line business focus. The premise was that variabilities are a higher conception that often are not localized within a product's concrete artifacts but rather may be dispersed across and pervade the form and content of the product as a whole. The purpose of the decision model was to provide an overarching framework for identifying the sources of variability throughout a product regardless of where within the product the implications of that variability might be found.

The decision model was conceived to define only the things about similar products that differ. Factors that correspond to common features of similar products are not explicitly represented in the decision model. More to the point, the decision model identifies the decisions that application engineers need customers to make so they know which of a specified family of products is to be built. RSP introduced the concept of an "application model" as being a particular resolution of decision model factors that distinguishes one application product from all other instances of the product family. The premise of a product family is that, by constructing an application model that resolves the decisions specified in the decision model by domain engineering, the production of a specific product is mechanical – no other information is needed to build a particular product that is in some way distinguishable from every other instance of the product family. By resolving decisions differently, application engineers have the means to create multiple products that may satisfy a customer's needs in different ways, reflecting different function-cost-quality tradeoffs.

Without a decision model, the development of a product family and associated components becomes a bottom-up domain engineering exercise of conjecturing all conceivable manner of detailed variability that may or may not be of actual value to customers. The decision model provides a controlling mechanism through which options for customization are constrained as justified by targeted market needs, rather than maximized by engineers' imaginations to the detriment of cost and schedule.

2.2. Aspect 2: Adaptable Components

The popular conception of software reuse is that someone will create libraries of fixed components that can then be used to rapidly compose products of any sort. Libraries of this sort have seen some success, as with the use of off-the-shelf commercial and free products, when customers' and engineers have light or malleable needs that can be adjusted to fit the capabilities and qualities that available components provide. When customers are more demanding, the use of fixed assets has not been so easy or successful. Clearly, if a single product meets a given need for everyone, there is no need for multiple products; however, in reality, differing needs are inherent in the tradeoffs that businesses must make in pursuing their varied endeavors.

The weakness of fixed components becomes evident in practice when developers are allowed to adapt such components to a customer's specific needs and then add the modified component back as a reusable asset. The result, lacking the discipline that comes with looking at every asset as an instance of a family, is a proliferation of similar components whose similarities and differences become obscured over time. Even if the product line as a whole has well-defined commonalities and variabilities, the components upon which the products are built reflect no such logic but only the arbitrary instances created to meet past needs and little consideration of future needs that characterize a product line. To adhere to product line principles, products need to be built from components that have been specifically engineered (or reengineered) to reflect the decisions that define the domain.

RSP proposed the use of adaptable rather than fixed assets (Figure 2). Whereas fixed components may be similar, that similarity is not explicit to their representation and easily lost; conversely, an adaptable component is a single unified representation of all the members of a family. The idea of adaptable components was a direct realization of the idea of program families, that components could be conceived as being instances of a family and that any differences among similar components could be explained as representing different resolutions of engineering tradeoffs among customer needs and constraints. Although within the framework of a product line the adaptability of components can arise from detailed insights about the capabilities needed, the decision model provides a focusing mechanism for limiting the diversity of instances actually needed as well as the effort required to create a component with sufficient tailorability.

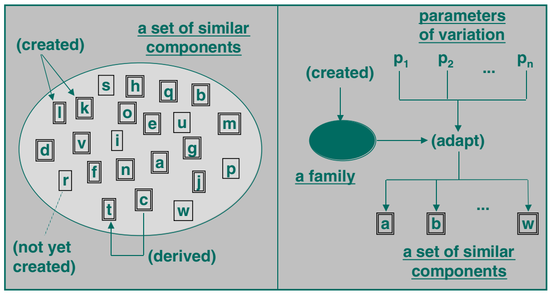

Figure 2. Two views of a component family

The adaptability of a component is represented by an associated set of parameters of variation that control the mechanical derivation of tailored instances of the component family. Adaptability to specified parameters is integral to the design and implementation of the component family as a domain artifact. In RSP, the relationship between the variabilities expressed in the decision model and the adaptability parameters of each component is an arbitrarily complex mapping: parameters may be derived from multiple decisions and decisions may factor in the resolution of many different adaptable components of a product family.

2.3. Aspect 3: A Domain-Specific Process

One of several primary differences between weaker and stronger forms of product line approach is a willingness to depart from the traditional model of a phased, work-product-focused software process. Like the typical conventional point-solution process, the application engineering process in the typical product line approach depicts a series of activities for requirements, design, implementation, and testing.

The RSP guidebook provided a framework for progressively streamlining the software process but a fundamental discontinuity occurs when the focus changes from producing a progression of work products to producing a product as a whole. From this point, a work-product-focused phased process impedes progress. Instead a whole-product application engineering process is non-linear, eliminating the detailed step-by-step effort to produce individual work products and introducing the means to produce any and all work products at any time (Figure 3).

To account for the view that the application engineering process could take different forms, the RSP domain engineering process included activities for both product family engineering and process engineering (Figure 4). Analogously, DsE defined a domain as “the knowledge (product family) and expertise (process) required to build a particular type of product.”

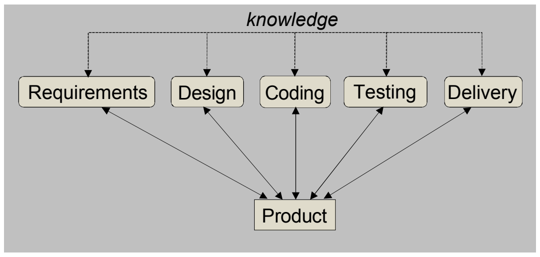

Figure 3. A non-linear software process

Product family engineering encompassed the requirements, design, implementation, and testing both of an architecturally integrated set of adaptable components for all categories of work product and of generators that application engineers could use to compose application work products from those components. Process engineering defined how application engineers were to work and constructed the corresponding mechanisms that application engineers would be provided as their means to create and evaluate an application model, identifying a needed product in terms specified by the decision model and then using the mechanisms provided by product family engineering to generate the corresponding work products for delivery into customer use.

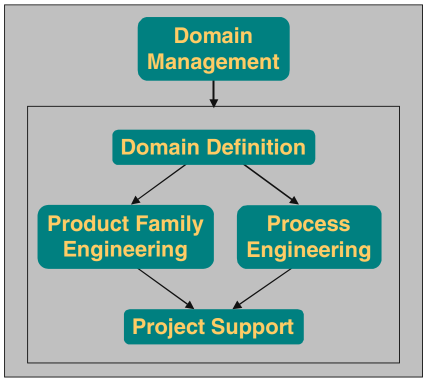

Figure 4. DsE domain engineering process

2.4. Aspect 4: Total Product Generation

In an advanced application engineering process (Figure 5), the application product as a whole is represented by an application model whose content is specified by the decision model. The product consists of a set of work products, each specifying for example requirements, design, implementation, test materials, management materials, or delivery materials and each being derived from a set of adaptable components guided by the decisions expressed in the application model. Adaptable components are the raw material for generating the work products comprising an application product.

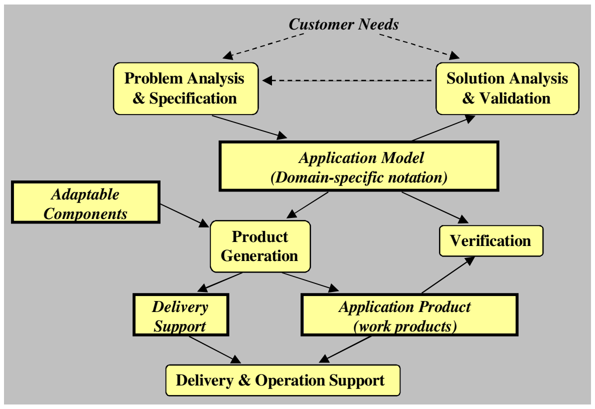
Customer Needs
Problem Analysis & Specification
Solution Analysis & Validation
Application Model (Domain-specific notation)
Adaptable Components
Product Generation
Verification
Delivery Support
Application Product (work products)
Delivery & Operation Support

Figure 5. A DsE application engineering process

In a product line context, there is no inherent need to produce work products individually in some arbitrary phased or "top down" order. Any work product can be produced at any time, being as correct and complete as the underlying application model and enabling domain capabilities permit. It is conceivable that for purposes of engineering or customer review and approval that it makes sense to produce only selected work products at a given point but this is not dictated by the capabilities of a product line approach. In a mature product line effort, it is imposed only if domain engineering institutes a process in which such selective generation is the proper approach for application engineering.

A motivation for having this perspective is the inherent co-dependence between a problem and its solution(s). In practice, problems are never fully understood nor completely communicated by customers. Furthermore, as developers propose solutions, customers often gain new insights into their problems, leading them to change how they characterize them. With a product line approach, such changes are easily accommodated when these correspond to changes in decisions identified in the decision model. There is then no reason to generate only a subset of the product's constituent work products. Alternatively, when a problem requires a solution that is not possible with existing product line capabilities, it prompts further domain engineering to extend the decision model and product family accordingly. This feedback from customers to application engineers and then to domain engineers is a principal means, along with market and technology projections, of guiding the productive evolution of a domain as market needs change.

A principal benefit of this perspective is that it provides customers with multiple compatible views of their problem and possible solutions. This enables producing multiple candidate solutions that allow the customer to understand the tradeoffs involved and allows them to choose a solution that best satisfies subjective as well as objective criteria of best fit to their needs.

2.5. Aspect 5: Model-based Validation and Verification

In a product line context, the concepts of validation and verification occur in reverse order from that of a traditional process. In the traditional process, there is no basis for validation until a complete product has been produced and verification is a highly subjective and time-consuming evaluation of whether dependent work products are consistent in content. With the DsE model of application engineering (Figure 5 above), validation is the evaluation of the application model as a restatement of the customer's understanding of the problem and needed solution. This model is meant to allow for both static and dynamic means of evaluation, including execution of the implied product within a realistically simulated environment or the application of formal methods to derived models of the product's properties. Analogously, verification is the evaluation of whether the end product has been generated consistently with the application model. Any failure of verification reflects defects of domain engineering, which have been seen to decrease rapidly as the products of a domain are used and improved.

Model-based analyses are one aspect of the RSP approach that has never been fully developed but, being one of the four principles of RSP, it was nevertheless viewed as an integral element. This neglect was a continuing issue of the lack of fundamental science and technology for the specification, measurement, and analysis of the properties of software behavior in a system. However, the product line context provides a great potential for enabling such analyses in that it leverages effort across a set of similar products. As with other facets of a product family, there are common properties that will be true of all products in the family and properties that differ across the family corresponding to the implications of its variabilities. Analogously, adaptable components provide a basis for the direct application of formal methods, leveraging the associated effort across all instances of the component that are ever produced, with a component's parameters of variation providing a basis for generalizing formal properties beyond the specifics of individual instance components.

2.6. Aspect 6: A Systematic Adoption-Improvement Framework

The RSP Guidebook effort was complemented by an effort to define a Reuse Adoption method [4]. Because this method did not assume a product line approach to reuse, it did not fully anticipate the needs of a product line effort but it did provide some essential insights for understanding how to adopt a product line approach. Specifically, it defined an adoption process that recognized and prescribed attention to many of the challenges inherent to the required organizational transition. It also defined a model for evaluating an organization's maturity that projected its readiness for instituting reuse practices.

Tailoring the Reuse Adoption method for a product line approach resulted in a method for Reuse-driven Process Improvement (PI_r) [5]. PI_r defined an enhanced adoption-improvement process (Figure 6) and defined four models to guide the performance of that process:

- Domain viability, for determining an organization's viable product line market focus
- Process maturity, for improving an organization's degree of engineering discipline
- Process capability, for targeting an organization's level of manufacturing discipline
- Product line strategy, for defining an organization's tailored product line approach

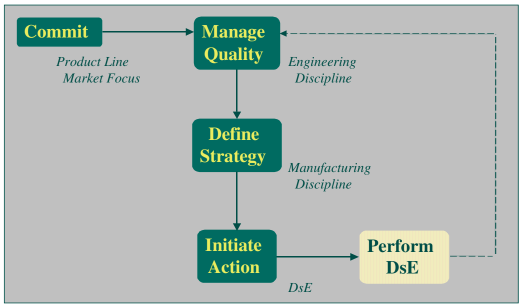

Figure 6. PI_r process

The use by Thomson-CSF (Thales) of the PI_r method was found to be broadly instructive into how a large organization could effectively institute reuse [6]. PI_r introduced several improvements over a general reuse adoption approach:

- It defined a broader, more comprehensive process for adoption and improvement specifically of a product line approach by an organization, including the continuous refinement of that approach as its needs change.
- In its process maturity model, it distinguished elements of organizational maturity that are addressed by general process improvement methods such as the SEI's Capability Maturity Model Integrated® from elements that are particular to product lines.
- It introduced a more coherent model of domain viability based on a better understanding of the factors that determine the success of a product line effort.
- Recognizing that product line approaches themselves constitute a family, it created a process capability model that reflected relevant factors for deciding among the levels of process capability so that an organization could derive a product line approach tailored to its own needs and capabilities.
- It refined the reuse strategy to be a product line strategy reflecting the market, business, organizational, technology, and transition contexts that determine how the organization should go about instituting and evolving its product line approach.
- It defined a rudimentary economic model that starts with an organization's historical measures of productivity and quality, recognizing that an organization's ability to produce software differs due to its maturity, its goals, its capabilities, and the inherent complexity differences among domains.

3. A Producibility Vision

As the science and technology of software engineering have advanced, a new vision of "producibility" is gaining viability. Happily, it encompasses and expands on the product line vision, but also takes us back to being able to reformulate the concept of a general approach not only for software product generation but also to encompass the totality of systems production. As defined in a preliminary roadmap for a program of research and transition [9], producibility is "the ability to deliver needed capability in a timely, cost-effective, and predictable manner."

From the roadmap, producibility has three dimensions that suggest the capabilities that need to be improved:

- Developer productivity (the efficiency and effectiveness of developers in creating and evolving a product)
- Product value (the utility and quality of each product that results)
- Acquirer acuity (the insight and foresight that acquirers have in delineating current and future capabilities needed)

This restates the original motivation for the product line concept, which was pragmatically limited to producing a set of similar products corresponding to the scope of a business enterprise. With this new vision, we can encompass product lines while also acknowledging the potential for other bases for limiting the scope of applicability of the production capability that results from domain engineering like activities. This particularly applies to product families that span business areas by providing solutions for broadly acknowledged needs with capabilities that are not themselves complete products but that serve as components of other business-directed application products. Similarly it provides a framework for the development and evolution of products that are long-lived and supportive of changing needs.

The producibility vision is of a capability for the computer-aided design and manufacture (CAD/CAM) of software-intensive systems (SiS). Taking a broad view of the meaning of CAD/CAM in industry, CAD is the conception, design, and analysis of a problem and solution in model form while CAM is the manufacture from raw and processed materials of a product that conforms to that model. The roadmap identifies five principles that characterize this vision:

- Model-centric (All problem-solution information is expressed in a comprehensive multi-faceted model of a product and its envisioned context of use.)
- Virtualized (A system is defined by building, pre-deploying, and validating in a software form within a hardware/software/user virtual environment.)
- Predictable (Software and dependent system properties of interest are able to be accurately predicted and mutually optimized as a product model evolves.)
- Decision-focused (Multiple alternative solutions are modeled, produced, and empirically evaluated based on identified customer and engineering decisions.)
- Evolvable (The problem-solution is continuously evolved to create variant products that satisfy anticipated differing or changing needs.)

This is a broadening of the same conception upon which the original product line vision was based, with the enabling constraint of a focus on a family of similar products. To realize this vision beyond the product line context still requires significant advances in our understanding of software as an artificial construct that must both correctly sense and represent the world in which it operates and also act effectively within it. It also goes beyond a narrow conception of product lines that focuses only on software to encompass systems engineering and customized hardware manufacturing as elements of a complete product that are interdependent and based on a shared view of customer needs and related engineering tradeoffs.

The roadmap identifies five areas of research focus required to achieve this vision:

- Model-based development (Bridging the conceptual gap between customers and product developers to rapidly formulate, build, and evaluate alternative solutions to evolving needs)
- Predictable software attributes (Measuring, predicting, and controlling SiS software properties and tradeoffs)
- System virtualization (Creating virtualized environments for realistically evaluating solutions)
- Disciplined methods (Applying effective methods for engineering discipline in the development of software within systems)
- Infrastructure and emerging technology (Exploiting changing infrastructure and computing technology capabilities for enhanced producibility)

For each of these five, the roadmap begins to define goals whose attainment will provide the technological capabilities (tools and methods) needed to implement the producibility vision as a systematic approach for the production of software-intensive systems. In the interim, much of this vision can be implemented today within a product line context. While we may lack the generally applicable scientific insights to apply this vision to build an arbitrary system, the limiting assumptions that underlie a product line offer a context in which more limited methods are sufficient. In addition, since many of the needed advances are relevant topics for product line research, such implementations will gain improved capabilities as the research challenges of producibility are addressed.

References

[1] *Reuse-driven Software Processes (RSP) Guidebook* (SPC-92019-CMC), Software Productivity Consortium, Nov 1993. <http://www.domain-specific.com/RSPgb>

[2] E.W. Dijkstra, "Notes on Structured Programming: On Program Families", *Structured Programming*, Academic Press, London, 1972, pp. 39-41.

[3] D.L. Parnas, "On the Design and Development of Program Families", *IEEE Trans. Software Eng.* SE-2 (1976), pp. 1-9.

[4] *Reuse Adoption Guidebook* (SPC-93051-CMC), Software Productivity Consortium, Nov 1993.

[5] *Software Product Lines Through Domain-specific Engineering*, Prosperity Heights Software, 2002. <http://www.domain-specific.com/>

[6] M. Ezran, M. Morisio, and C. Tully, *Practical Software Reuse*, Springer, 2002.

[7] G.H. Campbell, "Abstraction-Based Environments", Software Architecture & Engineering, Inc., 1988. <http://www.domain-specific.com/PDFfiles/ ABE-Spectrum.pdf>

[8] G.H. Campbell, S.R. Faulk, and D.M. Weiss, "Introduction to Synthesis", Software Productivity Consortium, June 1990.

[9] G. Campbell, "Software-intensive Systems Producibility: A Vision and Roadmap (v 0.1)" (CMU/SEI-2007-TN-017), Software Engineering Institute, 2007. <http://www.sei.cmu.edu/publications/ documents/07.reports/07tn017.html>

[10] D.M. Weiss and C.T.R. Lai, *Software Product-Line Engineering*, Addison Wesley, 1999.

[11] P. Clements and L. Northrop, *Software Product Lines*, Addison Wesley, 2002.

[12] K. Pohl, G. Böckle, F. van der Linden, *Software Product Line Engineering*, Springer, 2005.

[13] T. Käkölä and J.C. Dueñas, *Software Product Lines*, Springer, 2006.