Industrializing Software Production — Environments for the Domain-specific Engineering and Manufacture of Software

Grady Campbell

gradycampbell@domain-specific.com

ABSTRACT

Future software engineering research should focus on enabling the creation of automated environments for the rapid production of complex software products. Building such environments is feasible today but require significant advances in all aspects of software engineering to become cost-effective for complex high-quality products. A greater focus on understanding the nature of product families as intensionally defined sets of similar products is a related basis for motivating potential advances,

Keywords

Abstraction, Product Family, Specification, Generation, Validation, Evolution, Virtualization.

1. INTRODUCTION

The purpose of software engineering is the efficient production and sustainment of an effective solution to a customer's needs for automation. Each customer represents an enterprise whose objectives are achieved by inducing and responding effectively to real-world circumstances; the enterprise requires automated capabilities that properly align with and efficiently address its evolving objectives and challenges. The goal of software engineering research should be to formulate principles and develop techniques that enable more efficient production of more effective solutions to the needs of such enterprises.

A domain-specific approach to software engineering is an effective response to such enterprise needs but could be significantly improved through research. In particular, efforts to build environments that would support domain-specific engineering would expose and motivate numerous important topics for basic and applied research.

This paper describes a framework for constructing domain-specific environments to enable the rapid production of customized software products. This framework, in turn, provides the rationale for software engineering research needed to improve our ability to build such environments. Furthermore many of the most challenging problems for software engineering research arise in building such environments and others become more tractable when limited to a specific domain.

2. MOTIVATION

The essence of engineering is determining the nature of a problem and efficiently building an effective solution. As traditionally taught and practiced, software engineering is excessively focused on arriving at a single solution to a well-defined problem. Products are created largely through a process of trial and error. Defects are discovered through the unreliable mechanism of simulated use known as testing. Little emphasis is given to the prevalence of uncertainty and change as an attribute of real world circumstances. Analogously, development tools are conceived narrowly to support this inherently limited and inefficient approach.

In due course, this will change. Over the last 40 years, there have been numerous ideas put forward that contribute to the notion of being able to rapidly specify and build customized software. One of those ideas is build-to-order manufacturing [1], also known as mass customization or product lines. This idea, while more traditionally associated with the manufacture of physical products, applies very easily to software. The key is to recognize that similar problems ought to have similar solutions and that any differences between two products should be attributable to differences in the customers' needs that they satisfy.

When we talk of building a software product "to order" today, we may tend to assume that there should be no limitations on what a customer might request and certainly the resulting product should conform exactly to their request; after all, developers can build "anything" given adequate time and money. The developer then spends months or years working to understand the expressed need and building a product. At some point, the customer realizes that the product being built differs from their actual, possibly changed need. The customer then either faces further delay as the product is revised or must do what the buyer of a standard commercial product does, adapt their business operations to accommodate how the product works.

The build-to-order idea is that a motivated organization can create a capability to rapidly build products that can, within specified limits, be customized to a customer's specific needs. For software, this goes further, encompassing the capability to replace an existing, in-use product with a (re-customized hence modified) replacement product reflecting changes in that customer's needs. It goes further yet to support the possibility of producing software that would interpret any unresolved decisions as data to be set and used to guide it in adapting its behavior during operation. In reality, the build-to-order idea is a much more accurate characterization of what effective software organizations already do: they focus their resources on building products that leverage and enhance their existing expertise in specific business areas and enabling technology.

The build-to-order approach, to be properly realized today, applies best with a domain-specific product line focus, restricting what can be built to a set of products that are similar in their stated purpose and associated capabilities. Similarity of purpose leads to a family of products that can be designed to be similar in their construction, resulting in the ability to rapidly build different products. An assumption of similarity is key to limiting the investment required to automate support for this capability.

3. BACKGROUND

The basis for a domain-specific approach to software engineering is the concept of a product family. The essential notion of a program family was suggested by Dijkstra [2] in recognition of the fact that building a program was fundamentally a process of envisioning and choosing among a set of alternative solutions: “I prefer to regard a program ... as a member of a family of 'related programs' ... either as alternative programs for the same task or as similar programs for similar tasks.” From Parnas [3], the differences among a set of similar programs are seen to be viewed in terms of the decisions that motivate those differences. If we conceive of an envisioned set of related programs, for example corresponding to the diverse and evolving needs of a perceived market, then we can use the associated decision criteria as a discriminator to guide the manufacture of any particular product.

Domain-specific, or more narrowly product line, engineering [4] is an existing software engineering approach for the rapid production of customized solutions to an envisioned set of similar but diverse and changing problems. The focus of domain-specific engineering is the attainment of an ability to rapidly build (i.e., manufacture) instance products based on a specification of customer needs expressed in terms of a predetermined set of decisions. The definition of a domain from this perspective is the envisioned set of products that can be manufactured and the associated method and means of production. The means of production includes raw and processed materials (reusable assets), a mechanical (possibly automated) product generator, and a specialized process for specifying a product in domain-specific terminology, validating the described product to customer needs, and verifying a generated product to its specification.

The Spectrum environment [5], developed in the mid-1980’s, was an attempt to build a generic environment that would support build-to-order software production. In concept, it was influenced by much research of the time, including Lisp environments, advances in interactive user interfaces, bitmapped graphical displays, knowledge-based reasoning engines, truth maintenance, multi-paradigm programming, semantic data models, client-server computing, hypertext, and document processing. Much of this had emerged or coalesced in advanced software research efforts most notably at various universities including MIT, Stanford, and Carnegie Mellon and at the Xerox Palo Alto Research Center, but a basic construct developed for Spectrum was the ability to build reusable components that could be mechanically customized for different uses. This construct for abstraction-based “adaptable” components was the realization of the program family concept, generalized within Spectrum to the intensional representation of any family of concrete (but text-only) work products.

The basis for generating products in Spectrum was that the essential features of any product derive from a series of decisions that a developer must make to create the capabilities needed by a customer. The resulting environment was an effective proof of concept for the feasibility of fully automated product generation from a set of adaptable components guided by a specification notation that allowed resolution of a specified set of decisions. However, the combination of its implicitly broad scope of applicability with its initially limited range of implemented functionalities, showed it to be inadequate then as a replacement for conventional development methods. The key weakness was seen to be the lack of an effective basis for limiting such an environment’s scope of applicability. Recognition of this weakness led to the concept of a product line, with the realization that a focus on a coherent market (i.e., a set of customers having similar needs) provides a sound and objective basis for restricting the diversity of products that need to be built.

For the last 20 years, the capabilities for automated support of domain-specific engineering have remained extremely limited. One factor in this is that development tools have continued to be narrowly conceived on the implicit premise that a developer only needs to build a single, serially changing product. No means are provided for building a set of concurrently existing product variants from a single description that allows characterization in full detail how products in the set are alike and how they differ and that permits extraction of individual products from a set represented intensionally. Failing to provide such means, tools have not only failed to support but have also inhibited the ability to construct and concurrently maintain and evolve a set of similar products without redundant effort. The full potential benefits of a product line cannot be realized until there are proper means to automate the production and evolution of a set of similar products.

4. USING A DOMAIN-SPECIFIC SOFTWARE MANUFACTURING ENVIRONMENT

An environment for the domain-specific manufacture of software would be substantially different from current programming environments. Its capabilities would be determined through a process of domain engineering to enable the construction and evaluation of a specific family of products intended to address a targeted market; as the market and associated perceived needs change over time, domain engineering would change the specific capabilities of the environment accordingly.

In general, such an environment would provide the following types of capabilities:

- Layered descriptive specification of product capabilities (decision-based discrimination over an intensionally-defined similarity set)
- Specification and derivation of alternative products for comparative analytical and empirical evaluations of their behaviors
- Fully automated generation of candidate products, consistent with a (partial or complete) decision-based specification
- Creation of hybrid virtual/real operational environments for systemic validation of software, hardware, and processes
- Total hardware device virtualization, with deferred custom fabrication as needed for increased precision or for productization
- Product installation and use in customized, time-space virtualized operating environments
- Fully automated scenario-based product operation with a mix of simulated/real human agents, other software, and hardware
- Mechanisms to dynamically inspect, adjust, measure, and control a product and its operational environment while in use
- Static and dynamic measurement, analysis, prediction, and optimization of product quality/emergent properties and tradeoffs
- Transparent distributed computation, data sharing, and device sharing in the specification and evaluation of products

Specification of a product in this environment would entail completion of a domain-specific model representing the decisions

that distinguish among the products that can be built through this environment. Domain engineering collapses a large set of constructible products into an abstraction of the set from which instance products can be derived given resolution of a characteristic set of differentiating decisions. By having the ability to build only a subset of all conceivable products, the environment can implicitly build upon the assumptions that an expert in the domain would make, reducing the dialog for finding a solution to that of resolving only uncertainties that a knowledgeable engineer would face in that domain.

Many interesting perspectives have appeared over the years that influence our thinking on this. One is that software creates an artificial environment analogous to that of a performance of a play or the operation of a game in which users are integrally involved as actors, with outside influences seen through the agency of other actors and staging [6]. Another is that software is a medium for approximating reality as data, that is, as a model of reality, and the means to evaluate and initiate actions that modify reality to some intended purpose [7]. Finally, a third is the complexity of building increasingly integrated software-intensive cyber-physical systems and the need for models that allow us to analyze a product as a means to predict and adjust resulting system properties [8]. These, and many other perspectives, lead us to the idea that, to build increasingly complex software, we need an associated ability to construct controlled artificial environments for software validation that accurately approximate actual operational environments while giving us an effective ability to monitor, analyze, and control the behavior of candidate software solutions while in operation.

One way of envisioning how the validation aspect of such an environment would operate is to think of installing software in a real-world environment as analogous to injecting a plug-in into a running program in order to add to or modify its behavior. However, to avoid introducing unvalidated software capability into an uncontrolled real-world environment, an environment for validating domain-specific manufactured software would be a controlled hybrid of simulated and physical capabilities constructed to emulate the essential properties of the targeted real-world environment. The validation environment itself might in fact just be a product of the same or another domain-specific manufacturing environment, that is, a factory for constructing artificial environments into which software products can be injected for purposes of validation.

5. BUILDING A DOMAIN-SPECIFIC SOFTWARE MANUFACTURING ENVIRONMENT

As described, the capabilities needed in an environment for the domain-specific manufacture of software far exceed the capabilities of today's single product development environments. To construct domain-specific manufacturing environments, we need to create the following types of capabilities:

- Capability to construct an environment for performing an iterative process of domain-specific product specification, analysis, and validation
- Capability to construct and statically verify an evolving set of adaptable raw materials (architecturally-consistent component family-based generators)
- Capability to generate one or more executable product implementations from a partial (multi-valued) product specification, along with associated documentation and other supporting materials.
- Capability to specify and generate virtualizations (behavioral simulations) of hardware devices, manual processes, and natural environments
- Capability to construct an instrumented control infrastructure for observable software execution and analysis

A preliminary characterization of these capabilities, and many associated potential topics for basic and applied research, are discussed further in a draft roadmap for the Office of the Secretary of Defense on creating and transitioning improvements in software producibility into practice [9].

Although approximations of many of these capabilities could be built today, they would be limited by our lack of deep insight into the nature of software and how to predictably build a product that will exhibit desired behavior. On the other hand, attempts to create these capabilities would stimulate research and many advances in response to the real challenges encountered. A focus on understanding how small differences in similar programs affect their behaviors would provide new insights into how to construct programs to have predictably bounded behaviors. Building such environments would also drive needed advances in all of the basic methods of software and systems engineering, including management, requirements, architecture and design, implementation, and verification. The further need to transition research advances into practical application in domain-specific environments will increase the realized value of software engineering research and justify continued future investments.

6. REFERENCES

1. Holweg, M. and F.K. Pil, F. K. 2001. "Start with the Customer," MIT Sloan Management Review (Fall 2001), pp. 74-83.
2. Dijkstra, E. W. 1972. "Notes on Structured Programming: On Program Families," Structured Programming, Academic Press, London, pp. 39-41.
3. Parnas, D. L. 1976. "On the Design and Development of Program Families", IEEE Trans. Software Eng. SE-2, pp. 1-9..
4. Campbell, G. H. 2008. "Renewing the Product Line Vision," Proc. Software Product Line Conf., pp. 109-116. http://www.domain-specific.com/PDFfiles/PLVision.pdf
5. Campbell, G. H. 1988. "Abstraction-Based Environments," Software Architecture & Engineering, Inc. http://www.domain-specific.com/PDFfiles/ABE-Spectrum.pdf
6. Laurel, B. 1993. Computers as Theater, Addison-Wesley.
7. Kent, W. 1978. Data and Reality, North-Holland.
8. Broy, M. 2006. The 'Grand Challenge' in Informatics: Engineering Software-Intensive Systems," IEEE Computer 39 (10), pp. 72-82.
9. Campbell, G. H. 2007. Software-intensive Systems Producibility: A Vision and Roadmap (v 0.1) (CMU/SEI-2007-TN-017). http://www.sei.cmu.edu/library/abstracts/reports/07tn017.cfm