

Software Asset Management and Domain Engineering

Satish Subramanian
Guidant Corporation,
Cardiac Pacemakers Inc. (CPI),
St. Paul, MN 55 112
Satish.Subramanian@guidant.com

Promoting reusability by managing software assets can greatly benefit companies that develop a family of similar products, where products are evolving from one another. One of the main goals of domain engineering is to identify and document the commonalities across the various products in a particular domain. Managing these common products or assets will keep the development cycle short for future products and help tame the large divergence seen in the development of different products.

Guidant Corporation has been involved in the development of medical devices, such as cardiac rhythm management systems at CPI, which are complex real-time safety-critical systems. Cardiac rhythm management systems include implantable medical devices used to monitor human hearts and to provide appropriate therapy when needed [1, 2, 3]. These devices and related products are constantly evolving as technology and market needs change. The systems being developed at CPI are thus a family of products and share many functionalities among them. There are several issues that need to be addressed in the domain analysis of a family of products.

Constant Evolution. Identifying commonality among the products and creating reusable and modifiable domain models can help handle the constant evolution of products that results from changes in software requirements, technology and hardware architectures. Future products can then be developed from these domain models. The variations and changes in the newer products should be fed back and incorporated into the domain models.

Feature Comparison. Requirements and design specification of products in a family could take on different forms and structure. This is especially true if these products did not evolve from the same starting specification or from the same team of engineers. The specifications differ greatly in their form but are similar in content as they describe the equivalent features of independent products. During domain analysis these products have to be compared to extract the common features among them. If the feature description format across products is not uniform and does not have the same level of detail then the process of comparison becomes

difficult. If a standard format is adopted within a company the feature comparison and domain engineering process will be less difficult.

Different Audiences. Experts from different domains work on the development of one product. These include physicians, software engineers, electrical engineers, industrial and manufacturing engineers. Documents intended for various audiences are created in the development of a product. Software artifacts that need to be created for reuse in such an environment are not just software architecture or source code. The domain models that are created with just one particular audience in mind will undergo significant changes in the future when other related documents change.

Further, the actual user requirements (the problem being solved) and the solution requirements (requirements imposed by the solution to the problem) not distinguished in the requirements specifications [7]. Documenting user requirements separately will aid in the comparison of solution requirements across products. The user requirements have to be abstracted out and documented separately for domain experts, whereas the solution requirements are intended for the engineers. Thus the domain models created should encompass information needed by experts from different domains. Automatic generation of various documents for different audiences from a single domain model is highly desirable and will ease the burden of maintaining the domain models and related documents.

Common Vocabulary. The involvement of people from different domains and experts in different products creates the need for a common vocabulary. The software engineer must have some knowledge of the domain to understand the requirements, and the domain experts must have some system knowledge to shape the requirements. But the domain experts do not know everything about the software architecture and software vocabulary, and a software engineer does not completely know the domain and related terminology. To facilitate the communication among these two groups it is desirable that the domain experts and the software engineers have a common

domain vocabulary and domain models that both can understand and use. This will also benefit software engineers who have worked on different products, as they tend to use different terminology in different projects.

Scenario based validation. Domain models generated can be validated and checked for completeness using various use-case scenarios. In addition to these scenarios, the software architectures can be evaluated against various change-scenarios (configurations) to assess their ability to withstand future changes. These scenarios also help in integrating domain models from different products as they help define interfaces between the models. So it is important to identify domain-related scenarios in various categories and exercise them on the models.

Software Verification and Validation. The products being developed at CPI are mission-critical systems and so they undergo a rigorous multi-stage testing and revalidation. These systems consists of a variety of software, hardware and firmware components, and need to be rigorously tested for safety and reliability. In testing a family of products, one can reuse test cases, test scenarios, testing techniques and, testing processes across products. But if these test artifacts are not flexible then they have to be recreated or redesigned each time there is a requirements change. To accommodate changing system requirements it is beneficial to create test architectures which facilitate reuse of system-level test scenarios across projects [4].

Hardware Abstraction. We found that one of the main impediments to software reuse is the tight coupling that exists between software and hardware. Most of CPI's embedded systems consist of both software and hardware components tightly coupled with one another. Software is constrained by the hardware; the size of the software, the memory and, the programming language used depends on the hardware platform. Consider a change in a product requirement that dictates a decrease in the size and weight of a cardiac rhythm management device. This change may necessitate the use of smaller (and maybe different) processing units in the device. In such a case, it would be difficult to reuse any software artifact which is highly dependent on the previous hardware architecture. To avoid rework the hardware dependencies must be kept to the minimum by designing hardware abstraction layers in the domain model and thus allowing the software and hardware components to evolve independently.

Software Safety Analysis. Software safety analysis involves identifying safety faults in software and mitigating their causes. At CPI, identification and removal of faults from devices is of utmost importance due to the safety-critical nature of the products [2,3]. The cost and effort required to perform safety analysis on new

products can be leveraged by reusing the safety analysis information and data available from the earlier processes applied on existing products. The hazard data, safety analysis processes, and fault mitigation techniques used for fault identification and mitigation in current products should be documented and integrated into the domain models for future reuse [5,6].

Repository. Storing and disseminating information is very important in any reusability project. The design of the repository will depend on the domain models created. For our purposes, the repository is required to store and maintain traceability among artifacts such as domain models, the domain architecture, source code related to various features in domain model, use-case scenarios, test scenarios, change scenarios, safety fault trees and mitigation techniques used.

Guidant Corp. is currently addressing several of the above issues in domain analysis, specification reuse, reusable architectures, verification and validation processes, and safety analysis in its Software Asset Management department. Creating reusable artifacts generated from existing products with qualities of modifiability and maintainability is one of the primary goals of this department. Various pilot studies have been performed at CPI to understand the issues related to feasibility and benefits of large-scale domain analysis and asset management endeavors.

References

- [1] L. Elliott, R. Mojdehbakhsh, W. T. Tsai, *A Process for Developing Safe Software*, Proc. of the 7th Annual IEEE Symposium on Computer-Based Medical Systems, 1994.
- [2] R. Mojdehbakhsh, S. Subramanian, R. Vishnuvajjala, W. T. Tsai, L. Elliott, *A Process for Software Requirements Safety Analysis*, Proc. of International Symposium on Software Reliability Engineering, 1994, pp. 45-54.
- [3] R. Mojdehbakhsh, W. T. Tsai, S. Kirani, L. Elliott, *Retrofitting Software Safety in an Implantable Medical Device*, IEEE Software, Jan. 1994, pp. 41-50.
- [4] M. Poonawala, S. Subramanian, W. T. Tsai, R. Vishnuvajjala, R. Mojdehbakhsh, L. Elliott, *Testing Safety-Critical Systems - A Reuse-Oriented Approach*, Software Engineering and Knowledge Engineering 1997, (to appear).
- [5] S. Subramanian, R. Vishnuvajjala, R. Mojdehbakhsh, W. T. Tsai, L. Elliott, *Framework for Designing Safe Software Systems*, The 19th Annual International Computer Software and Applications Conference, 1995, pp. 409- 414.
- [6] S. Subramanian, L. Elliott, R. Vishnuvajjala, R. Mojdehbakhsh, W. T. Tsai, *Fault Mitigation in Safety-Critical Software Systems*, IEEE 9th Symposium on Computer-Based Medical Systems, June 1996, pp. 12-17.
- [7] D. Volovik, R. Mojdehbakhsh, W. T. Tsai, *What software engineering can learn from practitioners*, Proc. of Software Engineering and Knowledge Engineering '90, pp. 216-221.