From the Proc. of the 9th Inter. Workshop on Software Specification and Design, Ise-shima, Japan, April 16–18, 1998

Feature Engineering

C. Reid Turner,[†] Alfonso Fuggetta,[‡] Luigi Lavazza,[‡] and Alexander L. Wolf[†]

[†]Department of Computer Science
University of Colorado
Boulder, CO 80309 USA

[‡]Dipartimento di Elettronica e Informazione
Politecnico di Milano
20133 Milano, Italy

Abstract

The notion of feature is widely used to denote the functional structure and visible properties of a software system. More specifically, features are meant to represent a user-centric organization of a software system's functionality. Yet, other than during requirements analysis, features are seldom treated explicitly by most existing tools and methods.

This paper argues that a feature-orientation can provide benefits to software developers throughout the software life cycle. We envisage specific applications of the notion of feature that provide a powerful and unifying structure for software life cycle artifacts and activities. We discuss the problems and issues to be addressed, a brief summary of our current research work, and suggestions and directions for future research in a new area we call "feature engineering".

1 Features in Software Systems

Software engineering provides concepts and mechanisms that seek to master the complexity of software development activities. Such complexity is caused by a variety of phenomena that have been widely discussed in the literature since the 60s. They range from managerial and sociological factors to specific technical issues. One source of complexity is certainly the gulf between the user and the developer perspectives. Users are often concerned with a system's gross functionality, while developers are grounded in its implementation artifacts. Users describe their needs in term of the "features" that the software product is expected to exhibit. Developers must translate such requirements into appropriate design choices and then into software artifacts [3].

The requirements engineering community has recognized the utility of structuring the problem domain, using terms such as "high-level" requirements [6] and "requirements clusters" [5, 8]. However, while some requirements engineering efforts seek to structure a system's requirements by its gross functional entities (i.e., its features), there is little or no connection made in those terms with later stages of software development. For instance, much of the literature on the feature interaction problem is concerned only with requirements specification techniques [1, 2, 10]. Indeed, in practice, requirements documents are frequently ignored in downstream development phases [4].

We believe that by making features "first class" within and throughout software development, greater use can be made of the problem-domain structuring put in place during requirements analysis. This leads to a number of benefits, including the ability to answer such fundamental questions as: *What features are present in this system? How does the software architecture support new features? How is this feature implemented? How is this feature tested? How do these two features interact? Is this set of features consistent?*

Thus, despite the obvious relevance and wide adoption of the term "feature", we still lack concepts and techniques that make it possible to exploit the potential of features across the entire life cycle. This paper argues for making features an explicit part of the entire software development effort, from requirements engineering through system maintenance. We use the term "feature engineering" [9] to represent the various strategies for exploiting features as first class.

In the next section, we provide a definition for the term feature that makes clear the distinction between features and feature implementations. We then briefly describe a number of ways that features can be instrumental in supporting different software development activities. In the final section, we describe our current research activities and identify a number of open research issues.

2 Features and the Software Life Cycle

People who develop and use software have an intuitive understanding of the term feature as some recognizable system capability. Standardized definitions of

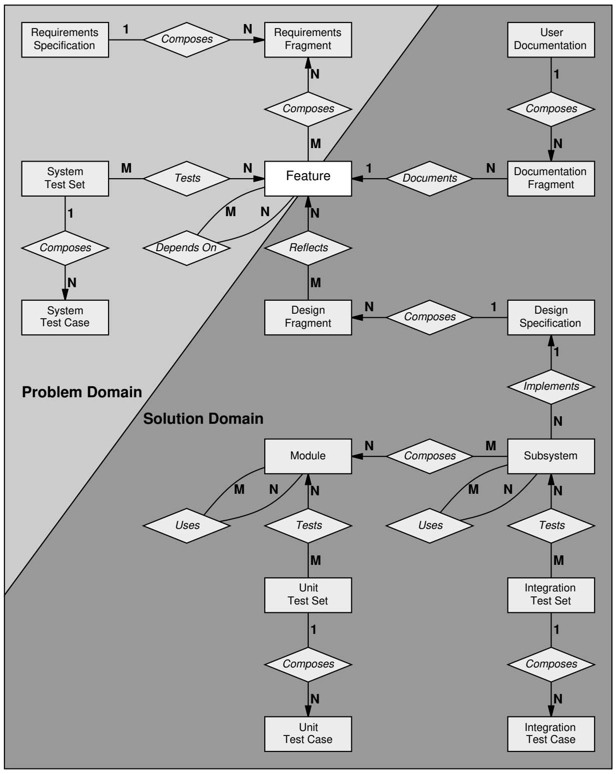

Figure 1: Life Cycle Entities and Relationships.

features (e.g., [7]) remain vague, and people often refer to features and feature implementations interchangeably. For purposes of this paper, we offer the following working definition for the term feature:

> *A feature is a clustering of individual requirements that describe a cohesive, identifiable unit of functionality.*

This definition emphasizes the origin of a feature in the problem domain. It says nothing about the manner in which a feature is implemented, and multiple implementations of the same feature are, of course, possible. Ideally, the requirements specification captures all the important behavioral characteristics of a system. Features are a natural organization of the requirements around a specific functionality. As a result, they participate in a number of important relationships with a wide range of development artifacts.

One possible model for these relationships is shown in Figure 1 as an entity-relationship diagram. The rectangles depict artifacts and the diamonds depict the relationships among them. The artifacts are divided between the problem domain and the solution domain, and features serve as a bridge between the two perspectives. Consider the user documentation for a typical system. This documentation is comprised of the entire collection of documentation fragments, which might be sentences, paragraphs, sections or chapters. Some interesting subset of these documents will be related to documenting the implementation of the system's features. The "documents" relationship identifies the association between a feature and the documentation fragments that describe its implementation in the system. Similarly, system test cases can be targeted to insure that a feature implementation is faithful to the feature's constituent requirement fragments. This figure also indicates that the mapping from feature to feature implementation is indirect and is thus often complex.

With this foundation, we can begin to explore the role that features can play across the spectrum of software development activities. Below, we identify interesting ways in which four traditional life cycle activities can be enhanced by a notion of feature.

Architectural design. A requirements specification structures the problem domain as features to be exhibited by an implementation, while the software architecture structures the solution domain as components and their connectors. Feature engineering has significant implications for software architecture. One is in relating the problem-domain structure of features to the solution-domain structure of components and connectors. Rarely is this mapping trivial. Another implication is that, from the perspective of the user, features are the elements of system configuration and modification. A high-level design that seeks to highlight and isolate features is likely to better accommodate user configuration and modification requests. Within this context, then, we see at least two mutually supportive approaches: feature tracing and feature-oriented design methods.

Testing. Feature implementations frequently cut across module boundaries and are thus candidates for module and system integration testing. Since individual requirements are associated with features, feature test designs should help pinpoint inconsistent or incomplete feature requirement sets, and systematic testing in the face of changes to various features can be improved with information about which features are implemented by which system components.

Reverse engineering. The primary influence of feature engineering on reverse engineering is to focus the analyses toward discovering connections between artifacts and features. Essentially, this means recreating relationships such as those exemplified in Figure 1.

Configuration Management. Configuration management can leverage feature information to populate workspaces with the minimum set of artifacts needed to modify a feature implementation. In addition, system builds could be specified by defining a desired set of features for the resultant system.

These observations are meant as a starting point for understanding intersections between feature engineering and traditional software development activities. In the next section, we describe some specific efforts that we have undertaken to explore and solidify a feature orientation to software development.

3 Future Work

Exploring the projection of features across development activities leads to a number of exciting opportunities for future work. We are currently carrying out three coordinated research activities.

1. We are evaluating how the concept of feature is currently exploited (implicitly or explicitly) in real software development processes. A first case has been centered on evaluating feature support in the current development practices of the UT100 telecommunications switch developed by Italtel. A second example is taken from the development process employed at the HP production site in Bergamo, Italy. Finally, two case studies are being carried out within the context of the ESPRIT project SACHER. They concern the development processes adopted by divisions of Nokia Mobile Phones and GEC Marconi Avionics.

2. Within the context of the SACHER project, we are developing semi-automatic support for impact analysis and cost evaluation of changes in feature-oriented software requirements. The expected result will be a tool able to navigate across the network of artifacts produced during development, and to generate information that can help the manager plan and control development efforts.

3. We are exploring the application of features to configuration management systems. In particular, we are studying how configuration management tasks can be raised from file-level operations to feature-level operations.

Certainly, there are a number of fundamental questions that remain to be answered before feature engineering can be fully realized. What are the characteristics that make some systems, such as telecommunication networks, a good candidate for feature engineering? What are the different relationships among features? How can we reconcile non-composable features? Can features be profitably used to reduce some of the complexity inherent in large software development projects? What is the relationship among features, software architectures, and component-based development techniques? We plan to explore these and other such questions as part of our research.

Acknowledgments. The work of A. Fuggetta was supported in part by CNR. The work of A.L. Wolf was supported in part by the Air Force Material Command, Rome Laboratory, and the Defense Advanced Research Projects Agency under Contract Number F30602-94-C-0253. The content of the information does not necessarily reflect the position or the policy of the U.S. Government and no official endorsement should be inferred.

References

[1] A.V. Aho and N. Griffeth. Feature Interaction in the Global Information Infrastructure. In *Proceedings of the Third ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 2–5. ACM SIGSOFT, October 1995.

[2] E.J. Cameron and H. Velthuijsen. Feature Interactions in Telecommunications Systems. *IEEE Communications Magazine*, 31:18–23, August 1993.

[3] M.A. Cusumano and R.W. Selby. *Microsoft Secrets*. The Free Press, New York, 1995.

[4] P. Hsia, A.M. Davis, and D.C. Kung. Status Report: Requirements Engineering. *IEEE Software*, 10(6):75–79, November 1993.

[5] P. Hsia and A. Gupta. Incremental Delivery Using Abstract Data Types and Requirements Clustering. In *Proceedings of the Second International Conference on Systems Integration*, pages 137–150. IEEE Computer Society, June 1992.

[6] J. Karlsson. A Cost-Value Approach for Prioritizing Requirements. *IEEE Software*, 14(5):67–74, September 1997.

[7] Standards Coordinating Committee of the IEEE Computer Society. IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*, December 1990.

[8] J.D. Palmer and Y. Liang. Indexing and Clustering of Software Requirements Specifications. *Information and Decision Technologies*, 18(4):283–299, 1992.

[9] C.R. Turner, A. Fuggetta, and A.L. Wolf. Toward Feature Engineering of Software Systems. Technical Report CU-CS-830-97, Department of Computer Science, University of Colorado, Boulder, Colorado, February 1997.

[10] P. Zave. Feature Interactions and Formal Specifications in Telecommunications. *Computer*, 26(8):20–29, August 1993.