CHAPTER

1

Introduction

In academic and industrial groups substantial interest has been directed toward understanding the fast-growing industry of software through the study of its products. By *product* we mean the concept, the code, the data, and the documentation that constitute a homogeneous unit and that can be sold in several copies. These kinds of studies focus on several issues, including technical, user, and commercial ones. For instance, some studies show how a product works, what benefits it delivers to the user, and how it is selling in the market. Addressing these issues for single products gives some interesting results. However, single products give partial information on the software industry. A more complete view requires that we focus on groups of products. There are visible reasons for this: software firms generally produce more than one software product and users generally use more than one software product.

These characteristics are also common to other industries—automakers produce more

than one car model and car users use more than one car model. However, they assume a higher relevance in the software industry. Producing several products rather than one is better because there are advantages to reusing experience and software across the products. Using several products rather than one is better because it is possible to assemble better and more complex software systems from different products.

This is an extreme approach that does not work. In the multitude of available software products, we may think that any product is somehow connected to any other product and that all the products must be considered together. This "group all" approach is of little use, much like the "single product" approach: It is impossible to deal with all products at once. The solution is to recognize that there are "domains" which gather software products into groups of reasonable size. The analysis of these domains yields interesting results for those who wish to understand the software industry.

In its simplest definition, a *domain* is a collection of related products. Relatedness of products is a very broad and manifold concept. Products can be related if they perform similar tasks, if they use similar technologies, or if they are used in similar contexts. *Domain analysis* is the analysis of products in a given domain. *Domain engineering* is the production of products in a given domain. The distinction between domain analysis and domain engineering is not very sharp, because analysis and production are closely related. While the distinction exists in literature, most people now refer to a single concept: domain analysis and engineering (DA&E). This book describes the theory and the application of Sherlock, a methodology for DA&E.

Traditionally, the motivation for doing DA&E has been to foster software reuse. Reuse means that software artifacts—like requirement and design documents, code, documentation, and so forth—are used in several products, which is the scope of DA&E. The reuse approach requires (1) the creation of reusable artifacts and (2) the integration of those artifacts into final systems. In current practice, firms perform both the creation and the integration. The proportion between creation and integration varies, covering the full range between pure creators (e.g., library producers) and pure integrators.

Software reuse tackles the issues of productivity in software engineering and is seen as one of the most promising approaches to software production. Its potential benefits range from decreased development time and increased product quality to improved reliability and decreased maintenance costs. As an investment, reuse works only if the software

fits a variety of situations and is actually reused many times. The larger the variety, the greater the potential for reuse and to reap the benefits of that reuse. However, dealing with variety bears some risks, like the fact that unknown or unforeseen requirements can challenge the reusability of software, or that quality can decrease as a consequence of adaptations. Variety is generally the result of variability, a leitmotiv of this book.

Variability comes from many sources: customers' unique needs, differences between versions, mutability of the environment, localization, and so on. All of these can potentially disrupt the efforts of reuse. A fundamental task of DA&E and Sherlock is to explore all the sources of variability within a domain. Sherlock is the recognition that variability is a characteristic of modern software that every software firm needs to confront. In this way, variability is no longer a source of trouble and unpleasant surprises. Rather, from a competitive viewpoint, it is a feature to exploit more effectively than your competitors.

Variability is a crucial part of decision-making in all software projects. Variability shows that a project can be undertaken in different ways–adopting one of several alternative solutions, or implementing different strategies. Decisions determine the purpose and shape of the final products, and thus determine their impact on the market. Decisions can be made on a range of possible alternatives—which derive, for instance, from the market positioning, the technology, and the software architecture. Sherlock finds the benefits and the costs of the alternatives and provides the tools to evaluate these alternatives objectively. Moreover, it finds the interactions between the various aspects of variability—market, technology, architecture, and so on—which are often critical but also very difficult to dominate. In this way, Sherlock gives the DA&E practitioner the tools and confidence to make clear-cut decisions about software product strategies.

Decisions are made at all levels. Using a top-down approach, the following is a list of areas where decisions are encountered:

1. *Strategic positioning in the market.* Decisions concern the role of the firm and its products in the market. The elements to consider are the competitive relations in the market. The alternatives to choose from are those that allow the DA&E practitioner to achieve its long-term goals.

2. *Definition of the strategic positioning in terms of product lines.* Product lines are the ultimate implementation of a strategy. A product line

is the combination of products that embeds a strategic idea. A product line has the breadth to support a strategy over a long period of time. Decisions here should determine the shape of the product lines, considering the relations between them and competing product lines. Decisions should also make clear how many product lines are necessary and, in case there is more than one, what different objectives they address.

3. *Definition of product lines in terms of products.* This is the DA&E practitioner's tactic. Here, decisions are made with regard to the articulation of each product line in terms of products. Decisions should evaluate the best options to ensure that products are cohesive in terms of the product line's strategy. Products should also support each other, exploiting the synergies (or network externalities) between them and other products. The result of these decisions can be a plan for the development and coordination of the product line's products.

4. *Development of products.* The decisions should determine which design and implementation solutions best support the development of the products. Development is conducted exploiting the benefits—like synergies and reuse—of working in the larger context of product lines.

By taming variability, Sherlock also makes it more friendly to reuse. More precisely, it formalizes variability into manageable schema through which reuse is possible. A major difficulty in the practice of software reuse is that reusability of a software artifact is difficult to assess. Reusability can only be proved in the future: A piece of software is not reusable until it is actually reused! A piece of software may seem reusable in the current situation, but, if the assumptions of the current situation change, reusability can be compromised. In stable domains, assumptions rarely change, which results in predictable levels of reusability. However, not all domains are stable.

Instability may come from several sources, such as immature technology or competition between firms pulling the domain in different directions. The goal of Sherlock is to foster software reuse in domains with high variability. On one hand, this is a matter of tracking the evolution of the domain, so that changes in the assumptions do not come as an unpleasant surprise. On the other hand, it is also a matter of influencing

the evolution of the domain according to a plan and exploiting all the strengths that a software firm has, such as network externalities, compatibility, coordination, installed bases, and so on. Of course, Sherlock cannot foresee the unpredictable. However, Sherlock is a valid guide for spotting variability, evaluating the variants, and picking the choices that are thought to be most profitable and least risky. In this sense, Sherlock improves the quality of decision-making in software development.

From a practical point of view, Sherlock delivers a set of documents describing the domain and a strategy to use to move within the domain. In addition to this, Sherlock delivers a software framework on which developers can build a variety of applications in the domain, thus exploiting reuse. Object orientation is the underlying paradigm of the documentation and the framework.

This book assumes the DA&E practitioner to be the central point of the discussion. The DA&E practitioner is surrounded by the entities that are typically present in a domain—namely products, competitors, and partners. This book teaches the DA&E practitioner how to perform certain activities, which are organized in phases. The activities may follow different paths, depending on the DA&E practitioner's decisions and the situation of the domain. We provide examples, criteria, and templates to address a large variety of situations.

Throughout the book we assume an entire software firm to be the DA&E practitioner. This is reasonable because of the multitude of the required skills and activities. For this reason, readers of this book can be more than one person in the firm and the profile of those readers is manifold. Certainly, a lot of decision-making is involved in Sherlock; competence is needed to set the strategy for the development of the firm's products. Other skills are also essential. Knowledge of the firm's products should reside with the architects and builders of the products. Mastery of the technology is usually with those who research it, or at least know its trends. Knowledge of sales and network externalities lies with the marketing experts. Potentially, any person who envisions, studies, plans, or designs the products is likely to participate in DA&E. Practice shows that these competencies and activities belong to high or medium-high roles within the firm.

Sherlock shares some aspects with other DA&E methodologies. As Arango [1] points out, a basic scheme is common to all DA&E methodologies. The differences between Sherlock and other methodologies are evident in its form and substance. The formal differences involve the use of different terminology and representation techniques. As to the

substantial differences—in the authors' opinion, most of the other DA&E approaches focus too much on modeling the domain and too little on the characterization of the products. Moreover, very few of them give guidelines as to how to move from the analysis to the development of actual software. Conversely, Sherlock precisely characterizes the relations between the products in the domain and generates a tangible asset—the domain framework—which constitutes the basis of real software products.

As mentioned, Sherlock characterizes the relations between the products in the domain. Informally, some examples of the considerations that this characterization may generate are as follows:

- Competition is not a pure marketing matter. Product features can be designed to defeat competitors.
- There is little space for a new product that does not offer radical improvements over the existing products in an established domain.
- Even a poor product produced in a timely manner can be highly profitable in a domain with high demand and little competition.
- Coordinated products can help each other penetrate hostile markets.

A clear understanding of phenomena like these sets the stage for the development of any successful software product—one who starts out on the right foot is already halfway there! Such understanding is essential in DA&E. DA&E aims at multiproduct development and, without a very clear strategy regarding what each product should do and how they should interact, coordination, synergy, and economies of scope will not happen.

Once the domain is characterized, it is necessary to select the functional and nonfunctional features the products should have. This selection process is a core part of DA&E. More precisely, for the products to be developed, the selection process determines which characteristics should be constant in all products, which should vary, and which may vary even though such variability is not implemented right away. Sherlock obtains relevant information from the study of the target firm and the market. It then builds a variability space that allows existing products to be analyzed, which leads to planning for future strategies.

The variability space visualizes the space of available choices. Products are the result of choices, so products can be positioned on the variability space. The variability space has rules to determine how products on it are related to each other in terms of users, competition, coordination, and movement. For instance, a typical problem is to determine how a product in a certain position on the variability space can be moved to another position to attract more users from a competing product. The construction of the variability space, the positioning of the products, the assessment of the rules, and the planning of the movements are all incremental activities for which Sherlock provides clear specifications. A Sherlock practitioner with a military background has compared Sherlock to military procedures in the sense that "you know what to do next."

This book contains many concepts—from market analysis to technical implementation, decision-making to object-oriented modeling. It is meant to be a self-contained package that explains the benefits of DA&E from the very beginning to the very end, and shows how to obtain those benefits. We did not want a theory book or a book only for DA&E "hackers." Moreover, Sherlock is not a "toy" methodology—it has already been applied with extreme success in several firms. In this book we have tried to provide all the material needed to replicate our experience in other contexts.

Understanding by means of examining examples is easier than learning from pure theory. The last part of this book contains four case studies obtained from the analysis of three different domains—telecommunications, neural networks, and control systems. In addition to these case studies, simpler examples accompany the presentation of the concepts, pointing out difficulties and proposing workable solutions. We want readers to be able to start practicing DA&E now!

Reference

[1] Arango, G., "Domain Analysis Methods," in *Software Reusability*, W. Schaefer, R. Prieto-Diaz, and M. Matsumoto, Eds., New York: Ellis Horwood, 1994.