HP Domain Analysis: Producing Useful Models for Reusable Software

Early software reuse efforts focused on libraries of general-purpose routines or functions. These fine-grained assets did not produce the hoped-for quality and productivity improvements. Recent software reuse efforts have shown that architecture-based, domain-specific reuse can yield greater quality and productivity improvements.

by Patricia Collins Cornwell

A software domain is a set of systems or applications that share some common functionality. This common functionality is typically embodied in various software components.* Domain analysis is a software engineering process that produces a characterization of a software domain to support the reuse of the software components. The HP domain analysis method produces a set of models that guide the design of reusable software.

While a few papers and books have been published on aspects of domain analysis,[1,2,3,4,5] very little has been published on practical domain analysis methods. HP has developed and refined a practical domain analysis method which has been used in several reuse projects. The method has proven to be an effective and efficient way to get the information needed for the design of domain-specific software. The focus of each domain analysis is guided by the business priorities and anticipated uses of the domain models.

This article describes a reuse process framework and the essential activities, deliverables, and typical uses of the results from the HP domain analysis method. Because the HP domain analysis method is designed to be tailored to the strategies of HP's business organizations, sections of this article will describe the business contexts for reuse strategies and special reuse roles in the organization.

Reuse Process Framework

Early reuse efforts focused on libraries of general-purpose routines or functions. These small-grained assets did not produce the productivity and quality improvements hoped for because so much engineering effort had to go into integrating these assets to produce a useful product. More recent efforts have shown that architecture-based, domain-specific reuse with larger assets can provide significant productivity and quality improvements. For the past five years, practical engineering experience in adapting reuse to meet HP's business needs has confirmed that the biggest return on investment comes from reuse that is based on domain-specific components that work in a flexible, but well-defined architecture. Reuse-oriented engineering addresses how these software assets are produced, supported, and used.

Because each organization has its own variations on processes and often quite distinct choices of specific methods, the U.S. Department of Defense's Software Technology for Adaptable, Reliable Systems (STARS) program sponsored a project to develop a conceptual framework for reuse processes.[3] This conceptual framework helps organizations understand the relationships among their software asset production, support, and utilization processes. HP participated heavily in the definition of the reuse process framework and provided some of the earliest experiences in its application. The usefulness of this reuse process was validated with organizations adopting reuse-oriented software engineering practices. This article's discussion of major reuse processes blends the knowledge gained from the STARS Conceptual Framework for Reuse Processes (CFRP)[6] with subsequent experience in reuse adoption at HP.

Fig. 1 shows the fundamental relationships among the reuse engineering processes: produce assets, support assets, and utilize assets. All three processes are guided by the results of one or more domain analyses performed in the Analyze Domain process, which analyzes and models a domain for architecture-based, domain-specific reuse efforts. Essential assets of software reuse-oriented engineering include a domain architecture and reusable software components.** Domain analysis produces domain models and other domain information used by the other three reuse-oriented engineering processes. The domain models and information are valuable assets, capturing the organization's knowledge about its product line capabilities and how those capabilities can work together in a range of competitive products.

* A complete software component includes both object code and all related information needed to use it. This related information includes parameterization information, source code if not proprietary, test information, design information, evaluation results, and other descriptive information.

** Some reusable software assets include software generators rather than components that are based on reusable code.

Fig. 1. *The domain analysis software engineering process.*

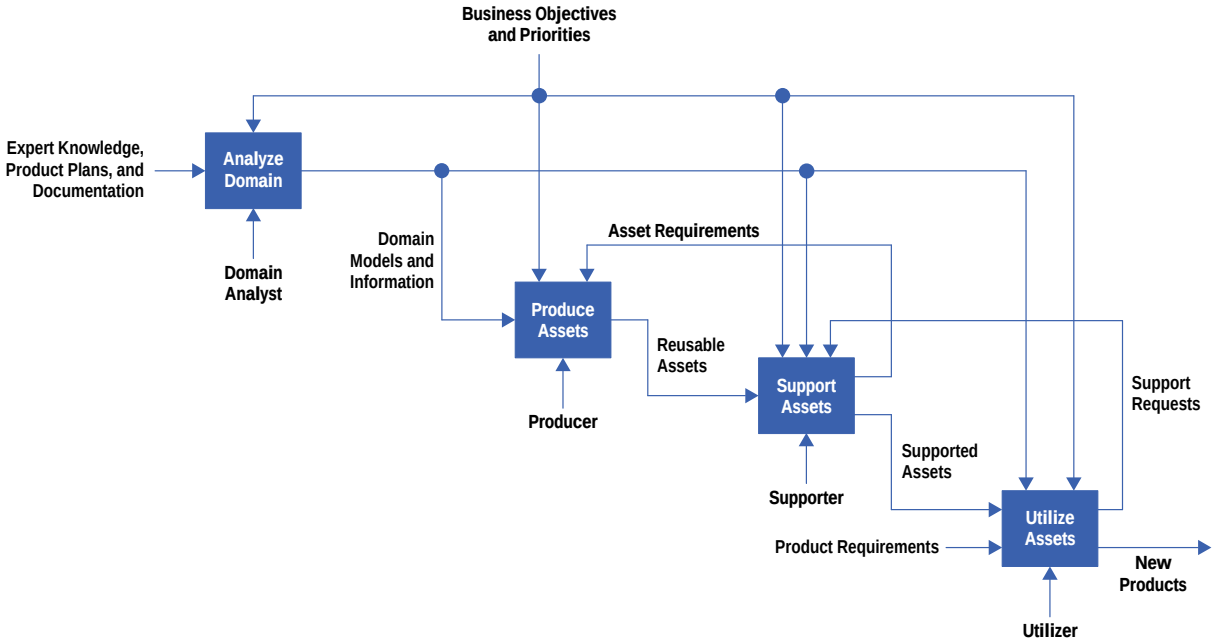
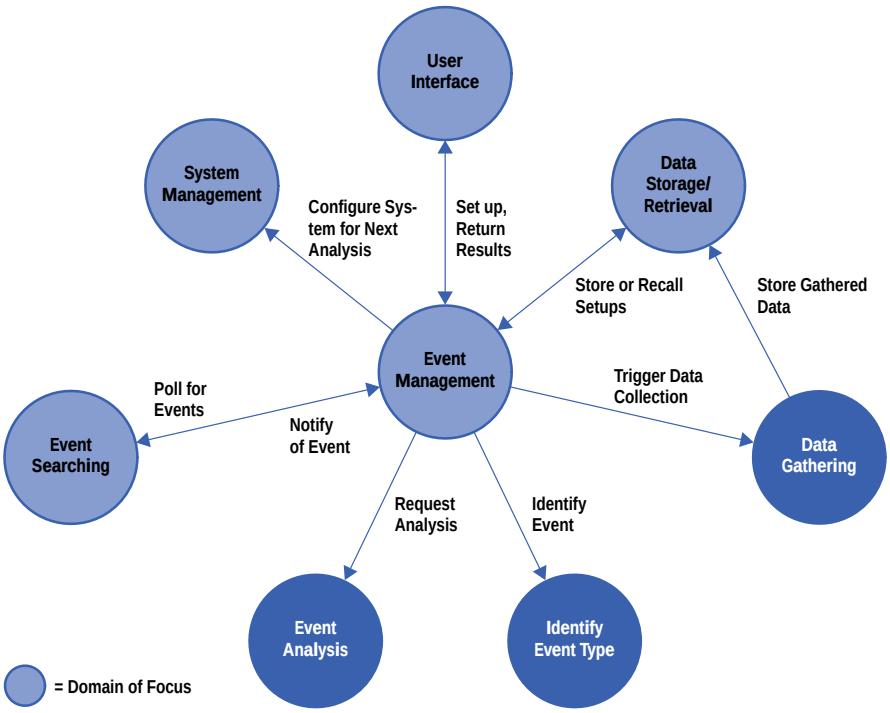


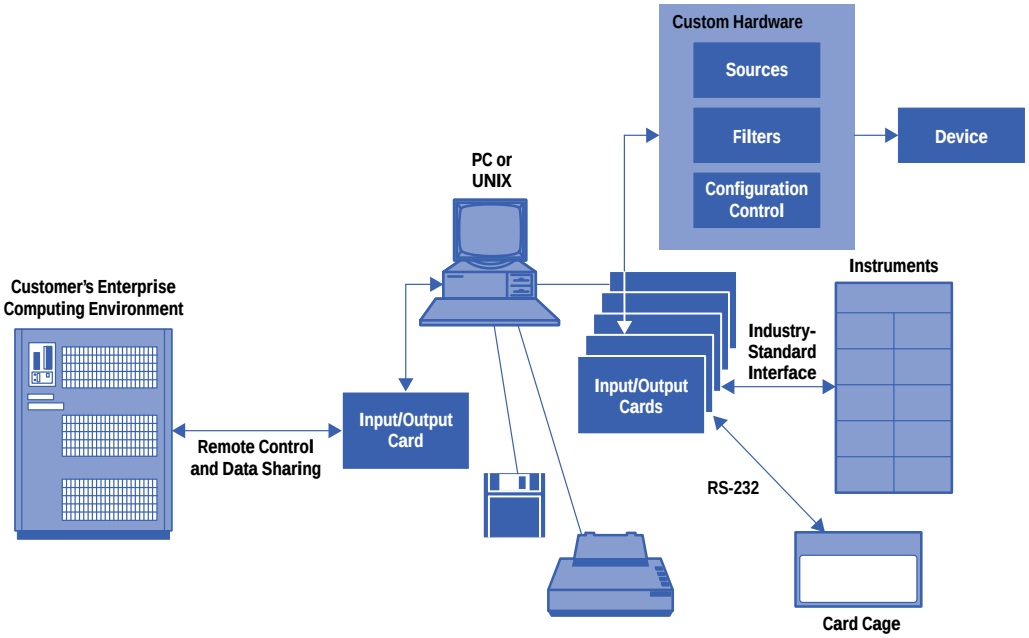
Fig. 2 shows a conceptual model for a data analysis domain, and Fig. 3 shows a physical environment model for a device measurement domain.

Fig. 2. *Conceptual model for a data analysis domain.*



Domain analysis is an essential part of any reuse effort. However, the domain analysis methods being used in industry range from an informal and quick expert prediction of what the domain should cover to a highly structured, exhaustive analysis and modeling effort with hundreds of pages of documentation. For different business situations, either of these extremes or more moderate alternatives may be appropriate. Across software engineering businesses, it is not possible to define one domain analysis method that meets all needs. In fact, even the higher-level descriptions of an overall reuse process may differ significantly from organization to organization. Rather than defining a domain analysis method that would work in only one business context, the HP domain analysis method is adaptable to a wide range of businesses that generate products that include software or firmware.

Fig. 3. *Physical environment model for a device measurement domain.*



The Produce Assets process shown in Fig. 1 uses domain models to develop reusable assets for use in portfolios of products within the domain. The Support Assets process includes managing the collection of assets and assisting the users in understanding how to take best advantage of the assets. Asset support also includes serving as a users' advocate with producers, integrating the needs of many user groups and assessing the relative benefits of producing or reengineering particular assets. The Utilize Assets process constructs new products with supported assets.

Business Contexts for Reuse Strategies

The first step in any domain analysis is to understand the business priorities and constraints where reuse is going to be used. There are numerous business circumstances that demand an improvement in the way software is developed and maintained. Reuse-oriented software engineering is often used to address business pressures for reducing product cycle time, increasing product quality, escalating the rate of introduction of new product features, and improving employee job satisfaction. However, software reuse is not always an appropriate way to accomplish business goals. Like so many software engineering methods, software reuse has become the goal for some organizations, rather than a means to accomplishing an organization's business goals. To ensure that reuse serves the business needs, we recommend that the management of a reuse effort begin by explicitly identifying the business priorities and analyzing what kind of reuse strategy (if any) will best support achieving those priorities.

Some organizations have launched reuse initiatives only to find that their products do not lend themselves to productive, cost-effective use of the software assets they develop. The two aspects of business context that most influence the decision to employ a reuse approach are the business goals and the product portfolio characteristics.

Businesses need to be more productive than ever to be competitive. Software reuse is rarely a short-term solution for meeting these increased productivity pressures. However, with a managed investment in adopting reuse, the benefits can be measured within the first few uses of the software assets.

Product Cycle Time. In many commercial businesses, the strongest competitors are those who dramatically reduce the time between introduction of a product and the introduction of its successor. To determine the potential impact of software reuse in meeting shortened time-to-market goals, we first assess whether software engineering (development and quality assurance) has critical path activities for the overall product development and release process. Managers who are being encouraged to reduce product cycle time need to focus on engineering activities that produce more and do it faster, possibly with a smaller team. They cannot afford to make organizational or process changes to noncritical path efforts if those changes won't substantially improve the product cycle time. Adopting software reuse involves an up-front investment in new skills and, often, in additional engineering effort. Therefore, investing in software reuse is appropriate when there is a predictable long-term benefit from the up-front investment, and the short-term costs of the investment are tolerable.

Product Quality. To determine the potential impact of software reuse in meeting product quality goals, we first assess how much of a product's quality is determined by the software or firmware. Software reuse is valuable in improving product quality when a significant amount of the functionality is consistent from product to product, so that as that software is tested and in use, more defects are found and fixed. For HP products, the question is often not so much of ensuring the highest

quality (which is a must) as it is of providing that quality with less testing time. Nevertheless, as the reusable software "ages" and fewer defects are found in successive uses, customers tend to experience higher quality in the products.

Rate of Innovation. Many businesses make ongoing trade-offs between the rate of introduction of new products and the number of innovations that are new to each successive product. The rate of innovation in products can be increased when the product is based on a stable software platform. A reusable software or firmware platform provides that base functionality without the effort to design and implement the same functionality for each product, freeing the product team to focus on innovations for each new product.

Employee Job Satisfaction. Improved employee job satisfaction has become a very important business goal in some organizations, especially those where intense pressures for meeting deadlines have resulted in employee burnout. We have worked with organizations where the move to reuse was motivated as much by the desire to provide a better work environment for sustainable employee job satisfaction and work-life balance as it was to meet marketplace pressures. For an organization that already has the market share it needs and has a competitive product line, the goal may be to release those future products with a team that is energized by the software engineering effort rather than exhausted by it.

Note that software reuse often makes it imperative for an organization to adopt architecture, design, implementation, and quality practices that would be a significant benefit to their software engineering projects even if reuse were not accomplished. Many software engineers welcome the move to software engineering methods that make them more productive, which contributes to their job satisfaction.

Product Portfolio Characteristics. An increasing number of HP divisions have business plans for a portfolio of products. The products may be tailored to address different market segments, from personal uses to enterprise uses, or tailored to meet specific industry needs, like automotive or banking businesses. We can look at this portfolio as a snapshot-in-time of the set of products a business wants to deliver. In addition, the product portfolio must be managed over time, with the introduction of additional features in successive versions of products. The business's vintage chart anticipates the desired product evolution and provides essential information for assessing the potential for reuse.

The characteristics of a product portfolio that can improve the prospects for software reuse rely mostly on the stability of the feature set in the product portfolio. There must be some significant set of base functionality that the set of products have in common to make it profitable to invest in reuse. This common functionality may constitute as little as 10% of a product's software and still be worth implementing with reuse in mind.

To reduce the risk of adopting reuse (any change involves risk, as does no change), those chartered with producing the reusable software rely on access to experts in the kinds of functionality for the products that will be produced. These people are often referred to as domain experts. These experts must be made available to the domain analysis effort as part of management support for the reuse strategy.

We use a rule of thumb in which the asset designers must get access to at least three existing examples of products that have the kind of functionality they want to provide in a reusable form. They also need characterizations of at least three intended future products that would also have that kind of functionality. The three examples give concrete information about what functionality is common (an existence proof). The three projected uses suggest that the investment will be amortized adequately to realize the benefits of designing, developing, and maintaining the software assets. The future uses also suggest the range of variation the assets must support.

HP Domain Analysis

The HP domain analysis method was developed by HP's software initiative (see ***Sidebar*** in Article 4). The HP domain analysis method supports analysis and modeling of capabilities (functionality or services) provided by domains such as microwave frequency measurement modules, crosscorrelation analysis algorithms, or report generation routines. The method explicitly addresses gathering the constraints and requirements of producers, supporters, and anticipated users of the domain analysis and related assets. A reuse strategy for software engineering is most successful when producers, supporters, and utilizers are full, active contributors to the domain analyses they will later use. The roles of the producers, supporters, and utilizers are described in the sidebar "***Reuse Roles***".

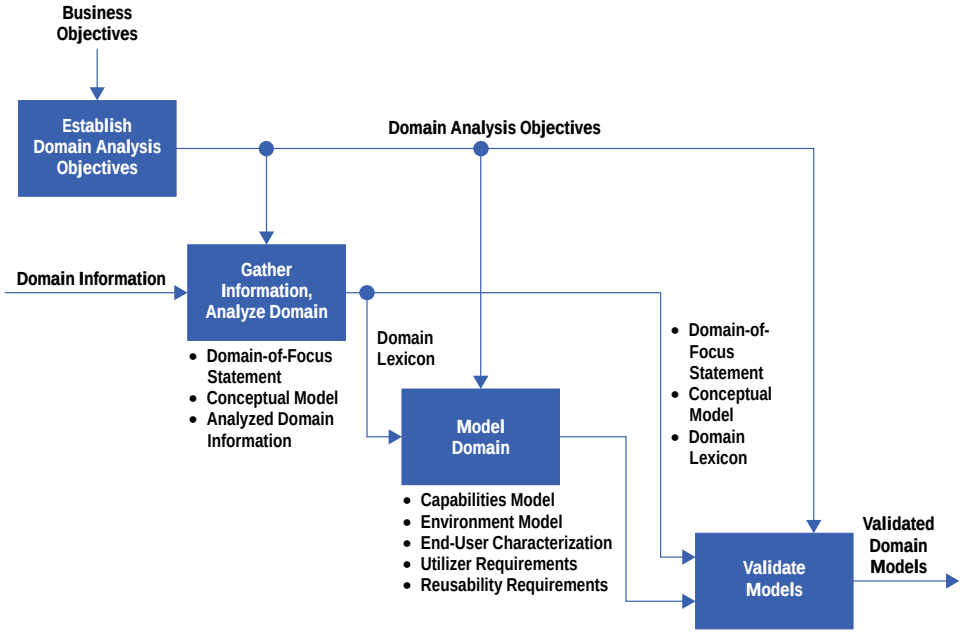
The Analyze Domain process shown in Fig. 1 produces a kind of reusable asset in the HP domain analysis method and is, therefore, conceptually a kind of Produce Assets process. Nevertheless, Fig. 1 shows the Analyze Domain process separately to emphasize that HP domain analysis guides and is guided by all three fundamental reuse engineering processes.

Basics of HP Domain Analysis

The HP domain analysis method includes domain analysis and modeling. The analysis identifies capabilities of systems in a domain of focus (i.e., the set of systems being analyzed), and classifies the common capabilities and the range of variation across systems that are anticipated in the future. The modeling captures the relationships among critical capabilities in the domain and creates models of the capabilities and their relationships without imposing a particular implementation solution.

In HP domain analysis, the term "domain" refers to any set of implementations (systems or subsystems, for example) in which the implementations have some common capabilities. Most of the time we define the domain by the set of common

Fig. 4. *The activities and deliverables that make up the domain analysis process.*



capabilities, rather than by listing all the potential implementations that could fall in the domain. For example, for a domain like a microwave measurement test system, there are endless possible products. However, most implementations of microwave measurement test systems include capabilities like test management, data management, report generation, signal measurement, and so on.

There are no common rules about what makes a set of capabilities the right size and complexity to be called a domain. If the domain of focus represents a consistent set of capabilities in a larger context (for example, in the context of a product portfolio), the most useful scope for such a domain is one in which there is a high degree of cohesiveness among the capabilities within the domain, and a limited coupling to other domains with which the domain of focus might be combined to produce products. For example, a domain like a graphics editor has significant complexity within it. However, in a larger context like document publishing, the graphics editor might have connections to other domains like text editing and document printing, which have a well-defined and comparatively simple interface.

Intuitively, domain assets are much more likely to be reusable if they provide a coherent “chunk” of desired capabilities and if the assets are easy to integrate into a complete solution. Typically, ease of integration is accomplished with a simple interface between the assets and the rest of the product software or firmware.

The Method

The HP domain analysis method usually involves three cycles through a set of well-defined activities. Fig. 4 shows these activities and the deliverables they produce, and Fig. 5 shows a typical level of effort expended on each deliverable during each of the three cycles through the domain analysis process. At each step through these cycles the analysis and models are refined and deepened and go through the same basic activities. The first cycle usually needs to focus on the context in which the domain will function because the team is still analyzing just what part of the overall product portfolio the domain must cover. The second and third cycles refine the scope of the domain and fill in details on domain capabilities and their relationships.

The following sections describe the activities shown in Fig. 4 and the deliverables produced by each activity.

Establish Domain Analysis Objectives. Use business goals and constraints to produce a clear statement of the purpose for the domain analysis. Identify those who have a stake in the domain analysis. Typically stakeholders include managers in the product development organization, those who will design and implement the domain assets, those who will support those assets, and those who are targeted to utilize the assets. Get agreement from these people on the objectives.

The deliverables from this activity include a statement of desired objectives for this domain analysis and a set of metrics that will determine the progress and success of the analysis. The documentation will also show how the domain analysis objectives are aligned with the business goals. Fig. 6 shows an example of some of the objectives and metrics for a domain analysis project.

The value of this activity is that it ensures that the domain analysis meets business needs, enables the domain analysis team to manage its investment of time and effort, and establishes a partnership with stakeholders to ensure that the domain analysis results meet their needs.

Fig. 5. *The level of development effort expended during each cycle of the domain analysis process.*

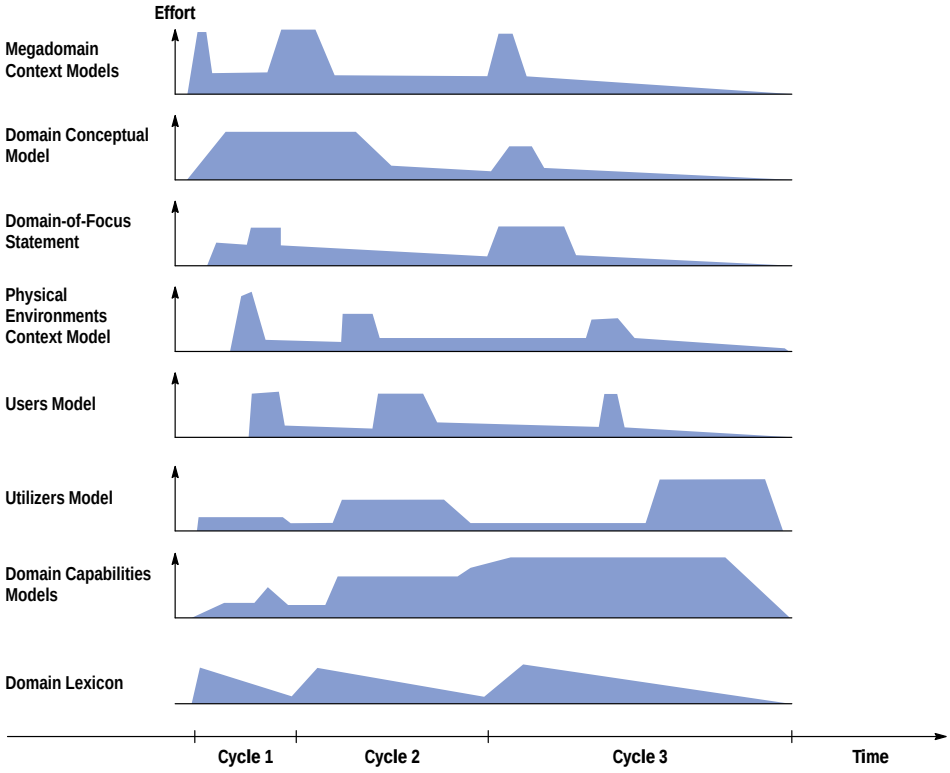


Fig. 6. *An example of some of the objectives and metrics for a domain analysis project.*

Objective:	The domain analysis will assess the value of building products based on reusable software, where that value is based on potential competitive advantage.
Metrics:	• Count the number of competitive capabilities in the domain capability models. • Name the clusters of capabilities that marketing identifies as a competitive advantage.
Objective:	Determine what software would be highly reusable as a software platform in our products targeted for the years 1997 to 2001.
Metrics:	• Does the domain context model clearly identify which parts of the domain constitute the platform domain? • Does the platform domain's conceptual model capture the capabilities needed for targeted products in the years 1997 to 2001? Are those capabilities highly similar across the targeted products?

Develop a Domain Context Model. Identify examples of existing systems that include capabilities of the domain you have identified. Use the list of targeted uses of the domain from the domain-of-focus statement to identify examples of future systems that include the domain. Develop a top-level model of the major elements of the systems. This is called the *megadomain*. Normally this model contains five to eight elements, with labeled relationships shown on the interconnections between the elements. Fig. 2 shows a conceptual model of a megadomain. This model serves as the organization's domain context model, since the parts of the megadomain that are in the domain-of-focus statement can be highlighted and the relationship to the overall megadomain can be identified.

This activity identifies a top-level interface between the domain and the rest of a system. The focus of the activity is in identifying what parts of a system are within the domain. The domain boundary decision is refined over the three cycles of the domain analysis.

Identify the Domain of Focus. Determine what domain will be the focus of the domain analysis and produce a *domain-of-focus statement*. The domain-of-focus statement is a descriptive statement (about three to five pages long) of what characterizes the domain, in terms of its anticipated uses and the scope of its capabilities. This statement complements the conceptual model (described below) and provides a basis for shared understanding about the domain, without needing to understand

the detailed relationships of elements in the domain. Because it describes the scope of the domain, it guides the more detailed modeling efforts. Fig. 7 shows an excerpt from a domain-of-focus statement for a printer command handling domain.

Fig. 7. *A portion of a domain-of-focus statement for a printer command handling domain.*

Domain of Focus Statement

The Rainbow domain [domain common name] is the system firmware that supports the set of all color printing solutions products [targeted products] anticipated in 1994 – 1997 (see Rainbow genealogy chart). [utilization time frame] There are three classes of end users for the Rainbow domain: people who send print jobs to the printer to obtain printouts, third-party application developers of applications that have printing capabilities (i.e., applications that interface to Rainbow-compatible printer drivers), and the person who detects and reports problems with the printer behavior, typically a systems administrator or product field service technician. [targeted end users]

The Rainbow domain supports people who send print jobs to the printer through its ability to detect and report (to the application from which the print request was made) any malfunctions and other status of the printer, to receive, interpret, and dispatch print commands (also, from the application from which the print request was made), and to control the actual printing, paper handling, and front panel displays (if any). [externally observable capabilities]

Be sure the terminology used is documented in the *domain lexicon* (described below). As the description of the domain evolves, be sure to keep the domain-of-focus statement consistent with the domain models, especially the conceptual model. Fig. 8 shows a portion of the lexicon for an I/O bus project.

This activity provides a succinct statement of the essential characteristics of the domain for use with stakeholders. It also serves as a reference for decisions about the scope of the domain.

Fig. 8. *An example of definitions that might appear in a domain lexicon.*

Access Functions: Functions that read or write data within the domain database.

Arbitration Block: A software component that monitors and controls arbitration for a bus.

Arbitration: The act of controlling access to a shared bus.

.
.
.

Bus Driver: The component that controls a given set of signals on the bus. Note that there is only one bus driver for a given set of signals.

Bus Signal: This term has two different meanings: hardware bus signal and software bus signal. A hardware bus signal is a voltage on a wire that connects architecture components together. A software bus signal is a variable that models the hardware voltage on a wire.

.
.
.

I/O Bus: The set of signals needed for communication between I/O devices and the processor bus.

I/O Controller: Hardware or HDL model that handles all I/O transfers from the processor bus to the I/O bus.

.
.
.

Processor Bus: The set of signals needed for communication between processors, I/O controllers, and memory.

.
.
.

Transaction: A series of events on a bus that accomplish a certain task. A transaction transfers data, inquires about or change state, or provides the system with information.

Gather Domain Information. Look at existing systems that include the domain. Also look at information on future products that might be able to use reusable designs and implementations in the same domain. Interview domain experts for their knowledge about trends in technology or capabilities associated with the domain. Identify the externally observable capabilities of the domain.

This activity provides the technical information for analyzing and modeling the domain, and ensures that the domain characterization is accurate and adequately complete.

Develop a Conceptual Model. Create a graphical depiction of the domain's primary elements and their relationships. This model may not be a technically accurate, top-level representation of the eventual design, but rather is an easy way to understand the "big picture" of the domain. It complements the domain-of-focus statement. Often the domain-of-focus statement and the conceptual model are used in management briefings and as part of the support documentation for engineers who need a general understanding of the domain.

There is no need to be fancy with the conceptual model. In fact, an intuitive model is preferred. Arcs between elements of the model must be labeled and the intended interpretation of the connections between those elements must be documented. The conceptual model should use the terminology defined in the lexicon and must be consistent with the domain description in the domain-of-focus statement. Usability of the conceptual model is enhanced by using a single modeling paradigm such as process flow, data flow, or entity-relationship diagrams.

Define the Domain Lexicon. This is an ongoing activity in which terms used to discuss the domain are defined, with pointers to related terms. The lexicon is used as a reference document for the terminology of choice for communicating about the domain. The domain lexicon includes every term in the domain models. It is invaluable in saving time and minimizing misunderstandings among the asset producers or between asset producers, supporters, and utilizers because it enables them to interpret the domain models consistently. The domain lexicon also allows a new member of a team to study the domain models and gain an initial understanding of the models without needing full-time assistance.

This effort will create a common understanding of concepts related to the domain, capture decisions about preferred terminology, and support efficient learning about the domain.

Model Domain Capabilities. Develop a model or set of models that capture the relationships among externally observable capabilities of the domain. This model is based on the gathered domain information, uses the terminology of the lexicon, and is consistent with the domain-of-focus statement and the conceptual model. The capabilities model is the primary reference used by the domain architect and asset designers. Thus, it must contain a characterization of all essential, externally observable capabilities of the domain. Also, because the capabilities model is concerned with the externally observable capabilities of the domain, it can be a valuable document for those needing to understand how to maintain or use the domain.

Later, the architecture and designers (who are members of the producer team) will transform the capabilities model into a chosen engineering solution. For organizations that have not done formal architecture and design, the capabilities model may serve as the design, supplemented by the other domain models. As the organization's skill in software design increases, the capabilities model will be the primary source of information for guiding the more detailed and formal software design. The domain analyst documents links between gathered domain information and decisions about capabilities and their relationships.

The most practical approach to capabilities modeling is to use the same paradigm as will be used in the architecture and design of the domain assets. Feature-based models, entity-relationship-attribute models, object-services models, and logic-rules models can be employed. However, the domain models are not meant to reflect implementation decisions. Furthermore, the capabilities models show only what is observable to utilizers and end users of the domain capabilities.

Most capabilities modeling requires a combination of aggregation, abstraction, and decomposition approaches to identify the top two or three layers of externally observable capabilities. Since these layers may not be strictly hierarchical, the models must capture the kinds of relationships that exist among the capabilities. Capability models identify each of the capabilities as required, or optionally, identify sets of alternative capabilities and note other capability interdependencies.

The most practical way to capture the range of variation across intended uses of the domain is through use scenarios. One very useful kind of model shows stimulus-response relationships among capabilities (i.e., what services or actions transpire as a result of what events) for different scenarios.

Modeling the domain capabilities provides domain architects and asset designers with a characterization of the domain's externally observable capabilities in sufficient detail for architecture and external design needs.

Model Physical Environments. Characterize the physical environments in which the software for the domain needs to work. This may mean describing the processors in an instrument where firmware runs, or the various heterogeneous enterprise environments where application software will run. Also, if there are diverse interface standards to be met, those are captured in this model. Fig. 3 shows an example of a simplified physical environment context model.

This activity ensures that asset designers have the information needed to accommodate computing environment constraints, like parallel or distributed processing, and refines the scope of the intended use of the domain assets.

Model End Users' Needs. Capture the characteristics of the end users that could influence the design or the implementation of domain assets. Each targeted development project may provide a characterization of their end users that includes skill level, understanding of how the product works, expectations, and a mental model of the user interface. The models of end users translate the end-users' usability requirements into a model of the capabilities the domain provides to meet those requirements.

This activity ensures end-user usability of domain assets and often influences the set of capabilities provided.

Model Utilizer Needs. Use identified utilizer needs for usability of the domain assets. This list usually includes the utilizer's constraints with respect to development environment (programming tools, version control and configuration management expectations, etc.), usage support requirements, and skill level. This information may influence the reusability requirement decisions and the domain architecture or asset design. The utilizer model translates the utilizer's usability requirements into a model of the capabilities the domain's components provide to meet those requirements. This model is typically quite different from the end-user model because it shows what will need to be provided in the product development phase, rather than the functionality needed in the delivered product.

Reusability Requirements. Develop a clear statement of how the domain will interpret such reusability characteristics as portability, modularity, scalability, extensibility, tailorability, interoperability, plug compatibility, and standards conformance. This statement defines how the team will know when they have achieved adequate reusability in their domain design and implementation.

Validate Models. Ensure consistency, completeness, and usability of the domain analysis and models. This activity is best supported with regular and explicit quality assurance activities, like minireviews or checkoffs against objectives and measures defined in establishing domain analysis objectives. This activity will ensure quality and provide an assessment of when the domain analysis is adequately complete.

How Much

The knowledge captured during a domain analysis is essential for success in reuse. Therefore, domain analysis is not overhead but rather a low-risk, efficient way of gathering and developing domain knowledge in forms that are readily accessible to those in the organization who:

- Develop reusable software or firmware
- Develop products that use the reusable assets
- Support the assets.

The larger the domain, the more people are typically involved in the domain analysis. Nevertheless, planning for it to take about six weeks of full-time effort from the start of the analysis and modeling until the stakeholders have what they need to delve into architecture and asset design is reasonable. Normally, the stakeholders are the first people to use the domain analysis results. For a team new to domain analysis, productivity will be greatly enhanced by having an experienced domain analyst (even one from a very different domain) available to guide the team through the method.

As mentioned earlier, the HP domain analysis method has three cycles of well-defined activities. For a typical domain, the first cycle generally takes about a week of focused effort for the domain analyst, who is leading the domain analysis effort, plus 10 to 30 hours for each of the domain experts providing information and assisting in the analysis. The second cycle takes about two weeks of information gathering, analysis, and modeling for the domain analyst and those who will be involved in the design and implementation of the domain assets. The users are consulted during this cycle, but the time commitment may only need to be a few hours for review and suggestions. The final cycle takes about three weeks for refining and validating the analysis and models. The domain analyst is involved full time, while the domain architect and domain asset designers may begin sketching out their first asset design ideas in parallel with the third domain analysis cycle. This parallel approach can ensure that the refining and validating activities meet the designers' needs. Domain analysis typically ends as an explicit activity when the design team has the information it needs to develop or reengineer an architecture and reusable software. However, it is important to maintain consistency between requirements, domain models, domain architecture, and asset design.

Using HP Domain Analysis Results

The deliverables from an HP domain analysis are designed to be useful in other, specific reuse activities. In architecting the domain and designing the reusable assets (Produce Assets in Fig. 1), all of the deliverables are used. The capability models and domain characterizations are primarily targeted to be used to guide these activities.

In supporting asset utilization, the lexicon is indispensable. The conceptual model and domain-of-focus statement are especially useful in acquainting developers with the domain assets (a form of asset support). The capability models and domain characterizations provide useful details for the utilizer, who is trying to understand how the assets might best provide the capabilities needed in the product (another form of asset support). The lexicon and capability model also support asset management through library classification, asset access, and configuration management.

In using assets, the utilizer will likely reference most of the domain analysis deliverables, initially relying on the conceptual model and domain-of-focus statement. Part of using assets is taking the initiative to identify asset requirements to the

producers, who will translate requirements requests into refinements of the impacted deliverables. For example, the need for a new capability is reflected in the capability models and lexicon.

Generally, managers rely most heavily on the conceptual model and domain-of-focus statement, with the lexicon as background material. See the sidebar: "***Management***".

Conclusion

The HP domain analysis method provides a simple and effective way of getting information needed to be successful in domain-specific, architecture-based reuse. By providing a method with a clear set of deliverables that have well-defined uses, we improve the efficiency and effectiveness of the domain analysis. Following the HP domain analysis method can substantially reduce the risks in reuse-oriented software engineering, risks that arise when the assets produced and supported do not adequately meet utilizers' or end-users' needs.

In most cases, we find the best results are obtained working with an experienced domain analyst the first time a team goes through the cycles to do their first domain analysis. Over time, that team's experience in domain analysis can increase to a level of domain analysis expertise that can be spread throughout the organization.

Acknowledgments

The HP domain analysis method was initially captured as a domain analysis workbook. Its primary author was Mark Simos of Organon Motives, who was a consultant to HP's software initiative program. Modifications to the workbook and the underlying process model were influenced by the work of Ruben Prieto-Diaz and Guillermo Arango.[4,5] A completely new workbook was developed after the modified workbook was used with HP product development teams in the computer peripherals, computer systems, medical, and test and measurement businesses. Mike Ogush and Tom Van Slack from the software initiative program have provided valuable suggestions for making the method match the needs of HP software development organizations.

References

1. J. W. Hooper and R. O. Chester, *Software Reuse: Guidelines and Methods*, Plenum Press, New York,1991, pp. 51-66.
2. *Symposium on Software Reusability*, ACM SIGSOFT, Seattle, Washington, April 28-30, 1995.
3. T. J. Biggerstaff and A. J. Perlis, *Software Reusability*, Vols. 1 and 2, ACM Press, New York, 1989.
4. G, Arango, "Domain Analysis : From Art Form to Engineering Discipline," *Proceedings of the Fifth International Workshop on Software Specifications and Design*, 1989, pp. 152-159.
5. R. Prieto-Diaz, "Domain Analysis of Reusability," *IEEE Proceedings of COMPSAC '87*, 1987, pp. 23-29.
6. *STARS Conceptual Framework for Reuse Processes (CFRP), Volume I: Definition*, STARS-VC-A018/001/00, September 30, 1993, and *STARS Conceptual Framework for Reuse Processes (CFRP), Volume II: Application*, STARS-VC-A018/002/00, September 30, 1993.

▶ Go to Article 6
▶ Go to Table of Contents
▶ Go to Journal Home Page