# The Role of Domain Analysis In Quality Assurance

Michael F. Dunn
and
John C. Knight

CS-94-18
June, 1994

Department of Computer Science
School of Engineering and Applied Science
University of Virginia
Charlottesville, Virginia 22903

Supported in part by the National Science Foundation under grant number CCR-9213427, in part by NASA under grant number NAG1-1123-FDP, and in part by Motorola.

# The Role of Domain Analysis In Quality Assurance[1]

Michael F. Dunn
*Industrial Software Technology*
HC1, Box 93
Earlysville, VA 22936

Tel: (804) 978-7475
mfd3k@virginia.edu

John C. Knight
*University of Virginia*
Department of Computer Science
Charlottesville, VA 22903

Tel: (804) 982-2216
knight@virginia.edu

**Abstract:** Domain analysis, which refers to identifying and structuring information common to a family of software applications, is a key activity for organizations adopting reuse-oriented development methods. The benefits of domain analysis extend beyond simply supporting reuse, however. In this paper, we show how domain analysis can serve as an important technique for quality assurance by serving as the basis for software component certification.

We describe how domain analysis can be used to gather quality requirements for a family of systems. We then describe our approach to certification, ending with some preliminary results.

**Keywords and Phrases**: Reuse, domain analysis, certification.

**Michael F. Dunn** is an independent consultant specializing in domain analysis, re-engineering, and software quality assessment. He has been involved with software development, maintenance, and research projects with organizations as diverse as IBM, Sperry-Marine, Motorola, and the U.S. Army.

**John C. Knight** is a professor of computer science at the University of Virginia. His research interests are in the area of software engineering of high dependability systems.

---

1. Supported in part by the National Science Foundation under grant number CCR-9213427, in part by NASA under grant number NAG1-1123-FDP, and in part by Motorola.

# 1 Introduction

When a software engineer designs a new system, there is an overwhelming tendency to start the job with a clean slate. The field of software development is still sufficiently young that there are few references that one can use to guide the design process of most applications. As a result, applications tend to be one-of-a-kind products, with limited potential for extension or usage in other contexts. From a productivity standpoint, this means that time is wasted by constantly developing similar systems. From a quality standpoint, it means that the mistakes of the past are doomed to be repeated.

Contrast this with other engineering disciplines. For example, when a civil engineer sets out to design a new bridge, he has a body of experience to draw from that spans several thousand years. Although the location of the bridge might be unique, other crucial factors, such as material strength, maximum load, and expansion and contraction characteristics, are likely to be well understood and documented. The engineer's job is to assimilate these known physical properties and apply them to the needs of the current project. In effect, he is *reusing* a great deal of knowledge and a set of general models to aid him in the design process. The knowledge and models are associated with a collection of similar civil-engineering products, i.e., bridges.

The contrast with other engineering disciplines and the relatively low level of software productivity has fostered a great deal of interest in software reuse. Reuse of all work products, of processes, and of knowledge have been proposed as is common in other engineering disciplines. As well as productivity improvements, reuse is expected to improve software product quality also.

*Domain analysis* is a set of techniques used to gather the underlying information associated with a collection of similar software products, and is the first step undertaken by an organization in developing a reuse-based software development process. Domain analysis has been an active research topic in the software reuse community for the past several years, and researchers have proposed a number of analysis methods. While similar in their objectives, these methods differ in their complexity and focus. Although it is generally agreed that domain analysis is a necessary first step in implementing an organized software reuse process, what is often ignored is the overall quality improvements that an organization can gain from this process, with or even *without* software reuse. Specifically, domain analysis forces the organization to examine the problem area for which it is building software systems, and determine the quality criteria to which these systems must adhere in order to serve the needs of the domain.

If reuse is to be employed, however, and domain analysis is being performed as an initial step, the emphasis in existing domain analysis methods is to characterize the products being produced and to define a process to facilitate the development of such products. We claim that explicit quality criteria for products can and should be an output of domain analysis. Presently, such criteria are implicit in domain analysis work products. In addition, we claim that a critical yet usually overlooked output of domain analysis is *how* this will be achieved and what impact it has on the resulting development process and in the quality characteristics of reusable assets.

In the following sections, we explore domain analysis from the standpoint of how it can be used to support the process of assuring the quality of reusable software components. Further, we

describe a method by which one can draw inferences about the quality of a component-based system if the quality of each component has been well-established by a process known as *certification*. We begin by providing background on domain-analysis concepts, using two well-documented methods as examples. We then describe various software component certification techniques, and tie certification in with domain analysis by introducing a method called *domain-driven certification*. The concluding major section provides some detail about an on-going project being performed in cooperation with Motorola, Inc.

# 2 The Nature of Domain Analysis

Domain analysis differs from traditional systems analysis in that the intent of a systems analysis is to determine the requirements of a particular system, while the intent of a domain analysis is to determine the requirements of a *family* of related systems. Once determined, these requirements can then be used to:

- Determine variants of systems within the family.
- Determine a set of components that can be used to construct systems within this family.
- Ease future maintenance by providing a conceptual base by which to gauge the impact of modifications.

Domain analysis is a fundamental engineering activity. All branches of engineering rely on three aspects of domain analysis:

(1) Understanding the needs of the various groups of people involved in the creation and usage of the product, and how these needs might change in the future.

(2) Understanding what has been done in the past, so that previous knowledge and experience can be used in future products.

(3) Creating a development environment that enables the production of new products in the domain

Fig. 1 shows a high-level view of the domain analysis process. The input information, process, and output information are described in the following paragraphs.

*Input Information*

The information collected during domain analysis provides insight into the current state of the domain and how it is likely to change in the future. Unfortunately, this information is typically voluminous, unstructured, and difficult to assimilate. Included in this collection is undocumented expert knowledge, source code from existing systems, system documentation, and organizational strategy plans. Also, the most useful information, expert knowledge, is difficult to capture because doing so requires sophisticated knowledge acquisition techniques [BrB89]. Source code from existing systems can be used to determine the differences and commonalities between systems,

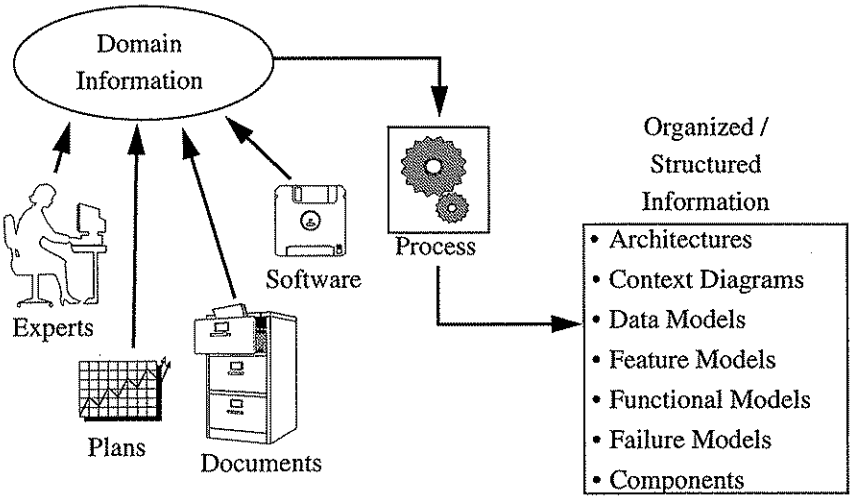


Figure 1 - High-Level Domain Analysis Process

which can suggest sets of reusable components. System documentation can be used to get an overview of how the system works, both from a user's and a programmer's perspective. Strategy plans can be used to determine future requirements. If the analyst knows that a set of systems will require a particular set of functions, he can use that information to derive a set of components to support these requirements.

*Analysis Processes*

Domain analysis is not one, but rather a set of complementary analysis processes, each one focused on a specific aspect of the domain. Commonly-used processes include:

- *Feature Analysis* for analyzing the services that the software provides from the perspective of the end user.
- *Functional Analysis* for analyzing the static and dynamic behavior of the system.
- *Architectural Analysis* for analyzing the various methods by which a set of related systems can be structured.
- *Failure Analysis* for analyzing the various failure modes of the systems and the impact that failures can have on the system's operational environment.

*Output Work Products*

The work products generated from the domain analysis process often resemble a generalized version of system specifications. Entity-relationship diagrams, structure charts, and state transition charts are all typical deliverables. These products are often parameterized, however, to factor out system-specific requirements.

These work products have two main uses. First, they provide a multi-view picture of the domain. This is analogous to a house blueprint. To keep the blueprint simple, house plans often

have separate diagrams to convey structural information, wiring, plumbing, and room layout. Combining these into one diagram results in picture that is busy and difficult to assimilate. The same principle holds in the case of domain analysis - one uses different work products to convey different information about the systems in the domain.

The second use of these work products is that they serve as generic templates to use in future system development. If the basic structure of these assets is present, much of the application-specific data can be filled in with specific requirements.

## 2.1 Domain Analysis Methods

A number of domain analysis methods have been developed in recent years. Two that have received a great deal of attention are those developed by the Software Productivity Consortium (SPC) and the Software Engineering Institute (SEI). We summarize both of these methods here and discuss them from the standpoint of how well they support the quality aspects of reuse. We introduce the general concepts associated with the enhancement of domain analysis to better serve aspects of the development process other than component development, aspects such as testing and maintenance. We refer to this enhanced process as *extended domain analysis.*

We stress that our descriptions of both the SPC and the SEI methods are necessarily very brief, and the reader is urged to consult comprehensive descriptions [SPC92, Kan90] for details.

### *Software Productivity Consortium*

The SPC's method is based on identifying the characteristics common to a family of applications within a business area. The main work products that result from this method include a set of decision models, specification templates, and generic code modules. The method also devotes a great deal of attention to the process by which a family of systems is developed. Thus, an additional work product is a specification for domain-specific development environments to facilitate building such systems. In other words, not only does the method consider systems in a domain, it also considers the infrastructure (tools, components, and environments) to facilitate the creation of those systems.

As Fig. 2 shows, there are three high-level activities performed within the domain analysis process - *domain definition*, *domain specification*, and *domain verification.*

Domain definition is the process of determining the domain's scope, and it characterizing the similarities and differences between systems in the domain. The term "scope" refers to an informal delineation of the set of systems that are included in the domain versus those that are not. The

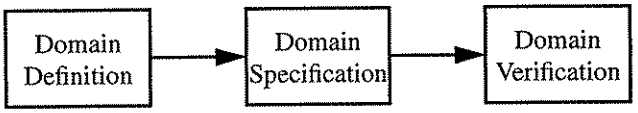


Figure 2 - SPC Process - Major Phases

deliverables resulting from this stage include four items:

- A *domain synopsis* that describes the functionality of the systems in the domain, high-level design issues, characteristics of the operating environment, and types of work products needed for applications built in the domain.
- A *domain glossary* that establishes the vocabulary necessary to discuss systems within the domain.
- A set of *domain assumptions* that establish the characteristics which are common among or variable between systems in the domain, or which would exclude systems from being included in the domain.
- A description of the *domain's status* that describes the domain's current level of technical maturity, and its potential for change in the future.

Domain specification is a set of activities for describing the work products used to build systems in the domain, and the process for using these work products effectively. It includes

- Generating *process requirements* that define the work products needed at various lifecycle phases of the development of the systems in the domain.
- Identifying *work product families* that are related applications which can use a given set of reusable parts.
- Creating a *decision model* that provide the decisions which the engineer must make when defining a particular member of a work product family.
- Creating a list of *product requirements* that specify, in abstract terms, the problems solved by families of work products within the domain.
- Developing a *product design* that describes the design of each work product family.

Domain verification is a set of activities for verifying that the implementation of the work products in the domain and the process for using these products match the requirements and designs produced during the domain specification phase.

The inputs to the domain analysis process cover three main areas:

- *Knowledge of the domain* that can be derived either from system users or developers. The information sources listed earlier, such as existing systems in the domain and system documentation, can also be used to derive this knowledge.
- *Knowledge of past and ongoing projects* that refers to understanding the subtleties of implementing systems in the domain. This includes understanding the implementation techniques used in those systems and problems that have occurred in previous development efforts.
- *A set of domain plans* that are a combination of two work products - a *domain evolution plan*, and a *domain development plan*.

  A domain evolution plan is analogous to an organizational strategy document. Its purpose is to determine the needs of near-term projects, and determine existing systems that are similar to these projects that might provide reusable components.

A domain development plan deals with organizing the resources needed to support reuse-based development in the domain. This includes determining the risks involved in each stage of the process, the objectives of each iteration of the development process, and the resources needed to accomplish each iteration.

Although the SPC method deals with quality issues in system family development, much of the relevant information is scattered across several sets of work products. For example, a number of work products provide information on component usage constraints, restrictions on allowable data values, and responses to undesired events. In addition, other quality factors, such as maintainability and portability, are captured to some degree by the decision models and variability assumptions generated by the method. However, the fact that this type of information does not reside in a centralized location can be a significant hindrance to the quality certification processes described later.

### *Software Engineering Institute*

The SEI's method is referred to as *Feature-Oriented Domain Analysis* (FODA). It is based on analyzing the similarities and differences between the various features, functions and behaviors possessed by a sample set of systems in the domain. The work products include entity-relationship diagrams, feature models, and functional models. The SEI process follows the three stages shown in Fig. 3.

The first stage, *context analysis*, is much the same as the SPC's domain definition phase. The goal is to determine the relationship between the domain under study and other domains. This can be illustrated by means of a structure diagram that is a hierarchical chart showing the domains that subsume, or are subsumed by, the domain of interest. Data flow diagrams are then created to show how information flows to and from the domain of interest. By performing a context analysis, one can begin to quantify the variability between systems in the domain.

The second stage, development of a *domain* model, is a process of analyzing the services that applications in the domain are expected to provide, and the expected run-time environments of the applications. This stage is divided into three subprocesses:

- *Feature analysis* that describes from a user's perspective what services are provided by the software.
- *Entity-relationship modeling* that describes the data used by and produced by applications in the domain, and the relationships between these data.
- *Functional analysis* that describes the static and operational aspects of the application's functionality. Static aspects can be characterized by such techniques as data flow

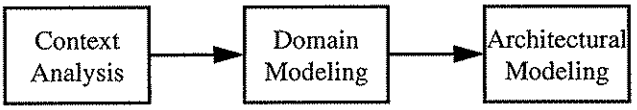


Figure 3 - SEI Process - Major Phases

diagrams. Operational aspects can be characterized by techniques such as state transition diagrams.

The third stage, development of an architectural model, is where high level designs of applications in the domain are created. The goal is to create an architectural model, which can thus be used to derive specific components.

As with the SPC method, required quality characteristics for systems in the domain are implicitly, rather than explicitly, specified. Feature modeling provides information on the services that are either required, optional, or mutually exclusive with other features in the domain. The entity relationship model gives an idea of required data and data formats. The functional model shows invalid or erroneous function states. And architectural modeling provides information on required portability characteristics. But as with the SPC model, all of this information is scattered across the set of output work products thereby complicating the certification process.

## 2.2 Extended Domain Analysis

While typical domain-analysis methods devote a great deal of attention to characterizing the functional and behavioral aspects of an application family, their focus on system quality characteristics is implicit rather than explicit. The intent of the *extended domain analysis* framework is to address this by identifying quality characteristics common across an application family and by determining the quality characteristics that must be present in a set of domain-specific components to support the system-level needs.

The term "extended" domain analysis is used not simply because it adds a new set of deliverables to the domain analysis process, but also because it uses as input the traditional set of deliverables. The process can be used on the outputs from any domain analysis method, as is shown in Fig. 4.

If we use the SEI process as an example, extended domain analysis uses the following process outputs to develop the required quality criteria:

- *Context analysis*

  Context analysis is useful in deriving maintainability, usability, and portability properties because it provides insight into the different user communities of the system and the different platforms on which applications are likely to be targeted.

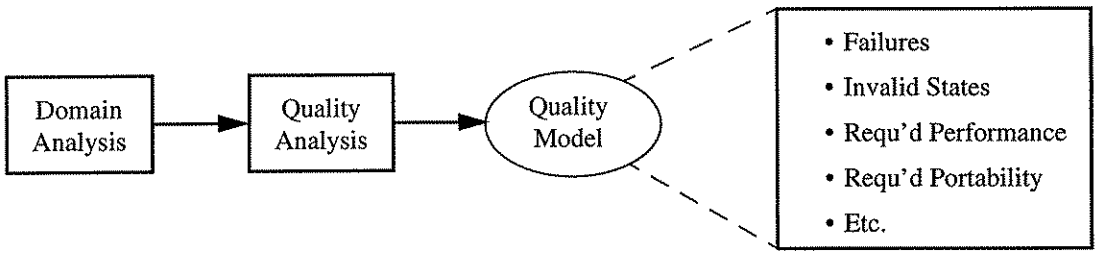


Figure 4 - Extended Domain Analysis

- *Feature analysis*

  Feature analysis enables the analyst to determine the potential usage characteristics of the application family, predict the types of faults that are likely to arise, and evaluate the impact these faults will have on the systems operational environment. Feature analysis is also important in gathering expected system performance characteristics.

- *Entity-relationship models*

  Entity-relationship models are useful in identifying the format and precision of data items shared between sets of components.

- *Functional analysis*

  Functional analysis is used to identify invalid or hazardous states of operation in the target application.

- *Architectural models*

  Architectural models enable the analyst to determine the interactions between the different components of the system, and enable him to determine how different faults in one component will ripple through and impact components in other parts of the system.

Thus, by performing an extended domain analysis, the engineer is paying attention explicitly to the quality requirements of products in the domain and generating outputs from the process that describe required characteristics such as functional correctness, portability, maintainability, and documentation in substantial detail. The first significant *additional* output generated by an extended domain analysis is, therefore, a list of the required quality criteria for products in the domain.

These criteria can be organized to show the specific relationship between the major functional areas in the domain as generated by the SEI's functional analysis process, and a categorized set of quality criteria. As an example, consider the domain of product shipping scheduling systems. Suppose that we have discerned the following major functional areas:

- *Customer management* that includes the entry of customer address and order information.
- *Transportation management* that includes information on the locations of warehouses, airports, and railway and truck terminals. It might also include information on transit times, costs, and schedules.
- *Schedule optimization* that takes customer order information and transportation information, and produces an optimal delivery route.
- *Report generation* that allows the user to generate reports and perhaps do "what-ifs".

For each of these functional areas, one would use the information contained in the domain analysis to produce the quality criteria to which each function must adhere. These criteria can be organized according to the quality classification scheme developed under ISO Standard 9126 that includes the categories of functionality, reliability, usability, efficiency, maintainability, and portability [Vol93].

# 3 Quality Properties and Certification

If a set of related systems in a domain is required to possesses a common set of quality characteristics, it is reasonable to expect that the reuse-based process for developing these products should help to ensure that they have the required properties. If this were possible, it would constitute a second area of productivity gain attributable to reuse. Rather than reuse just enhancing productivity through faster material generation of products, it would help reduce costs in areas such as testing.

How might this be achieved? In general, the approaches advocated for achieving this goal are based on the use of a set of components to support development of these systems that adhere to their own specific set of quality characteristics. Further, if it can be shown in a rigorous manner that system quality properties hold as a result of using components with their own specific properties, then we have taken an important step towards establishing the quality of the set of systems that will use these components. The process of establishing the component quality characteristics is called *certification*, and it has become a major issue in reusable software development in recent years for two reasons:

- The increasing acceptance of the idea, if not the practice, of reuse as a development method, and the growing number of tools and libraries to support it.
- The unevenness of the quality of components currently available to the reuser.

The availability of a set of trustworthy "plug and play" components will help to make reuse a standard practice. The purpose of certification is to provide a consistent set of criteria by which such components can be judged. More importantly, once the engineer knows that a component possesses a particular set of characteristics, it might be possible for him to draw inferences about the systems that use these components.

In this section we discuss three certification schemes, *levelled*, *scored*, and *statistical* that have appeared in the literature, and introduce the *domain-driven* approach developed by the authors.

## *Levelled Schemes*

In a levelled scheme, components are assigned quality "levels" according to the amount or type of quality assurance processing that has been applied to them. Components assigned to low certification levels have had little or no quality assurance applied to them, while those assigned to a high level have typically undergone some form of prescribed testing, inspection, and/or analysis, sometimes by a third party. There is no upper limit on the number of levels that can be used in such a scheme. However, most schemes currently in use include only four or five levels.

In a variation on levelled schemes, each level is divided into sublevels. The levels are often referred to as "primary levels" and the sublevels are referred to as "secondary levels". As in the basic levelled scheme, different primary levels correspond to the different cumulative quality criteria being used. Secondary levels refer to the amount of confidence the reuser should have that the component has achieved the primary level, based on the quality assurance processes that have

been applied to the component. Additional information on levelled certification schemes is available [PPS92, REW93].

### *Scored Schemes*

In a scored scheme, scores are assigned to a component based on a weighted set of quality criteria. Criteria can be determined by the organization based on desired domain properties. Many variations on this scheme focus on reusability-enhancing characteristics such as module complexity or number of parameters in each function. However, other quantifiable properties, such as average execution time over a given input sample, can also be used as criteria. A component's overall score is a composite of its scores in each quality area. The goal is to enable the user, if he is presented with a set of similar assets, to choose the asset with the highest overall rank.

### *Statistical Schemes*

Statistical schemes address only one major aspect of system quality: overall reliability. The goal is to determine the reliability of each component, and then infer the reliability of the system based on a composite of each component's reliability.

Statistical schemes are based on an understanding of system usage characteristics. The certifier creates a model of system execution characteristics that enable him to make an educated guess about which parts of the system are most likely to be executed and the likelihood that execution of one part of the system will be followed by execution of another. By understanding the frequency of execution of the different parts of the system, and the transitions between these parts, the certifier can begin to create a Markov model that captures these transitions. Using this model the engineer can assess the reliability required in each part of the system in order to meet an overall quality goal for the entire system.

The main certification criteria is mean time to failure. There are two ways of looking at this value. The first way is time-based, and is defined as the average length of time between manifestations of system faults. The second way is event based, and is defined as the probability of a fault occurring while processing a given input value.

### *Domain-Driven Scheme*

The methods described above each give a measure of confidence that a set of components has undergone some standard process of quality assurance. But each suffers from the same set of weaknesses:

- *Lack of tailorability* - The schemes are rigid in their definitions. A levelled scheme, for example, defines what the levels will mean and libraries certified to a given level must comply with that definition. This approach fails to address the needs of different organizations or domains.
- *Loss of information* - In each method, a large volume of information is compressed into a single number, making it difficult to determine what factors went into the quality

assessment process.

- *Lack of ability to draw quality inferences* - As noted above, given that we have a set of components whose quality properties are well-defined, we would like to be able to use this information to understand the quality properties of the systems using the components. The methods described above do not support this ability.

We summarize domain-driven certification in this section but omit substantial detail for brevity. The interested reader is referred to the authors previous publications on the subject [Kni92, DuK93].

*Domain-driven certification* addresses the three problem areas outlined above by making explicit the criteria by which a set of components is evaluated. In fact, this is the definition of component certification:

**Definition: A reusable component is defined to be certified if it possesses a prescribed set of properties.**

Notice that this definition is arbitrarily flexible. Different definitions can be established for different organizations and domains. Notice also, that the quality criteria possessed by a component are listed explicitly thus permitting engineers to understand completely what qualities a component possesses. Finally, notice that the properties can and *will* be tailored to permit their exploitation in the demonstration of domain quality properties in systems built with the certified components.

The ability to exploit component properties is an extremely important point. The set of properties that are included in a certification definition is defined to be that set of qualities that, if present, would facilitate the demonstration of required domain quality properties in systems - nothing else. Thus, only component properties that can be exploited to do something useful, i.e., show that a system possesses domain quality properties, are included in the certification definition.

Once the certification definition is in place and libraries of components have been shown to meet the criteria, it is possible to develop systems using those parts and establish domain properties relatively easily by depending on the component properties.

The domain-driven certification process can be summarized as follows:

- Perform an extended domain analysis and, using its output, determine the necessary quality criteria for products in the domain.
- From the *domain* quality properties determine the set of qualities that should be held by *components* in libraries used to build products in that domain.
- Develop systems in the domain of interest using a reuse-base development process and exploit the component certification properties to establish domain properties of the systems.

Finally, it is clear that domain analysis in its extended form as we define it is playing a critical role in the whole certification process and is in fact an essential contributor to system quality.

Fig. 5 shows how domain analysis and certification fit together in this scheme.

# 4 Experience

For the past two years, we have been experimenting with these ideas on certification in cooperation with the Cellular Tools department of Motorola, Inc. In this section, we provide some preliminary results of this work. A more detailed explanation can be found elsewhere [Dun93].

## 4.1 Problem Domain

The Cellular Tools department is responsible for developing software for simulating and testing the digital switches Motorola manufactures for routing and managing cellular phone calls. This domain is useful for validating the concepts of domain-driven certification because it possesses two important characteristics:

- The software created in the domain is intended to automate expensive processes. The switches under test are quite costly, and making the test process more efficient means possibly reduced development resources.

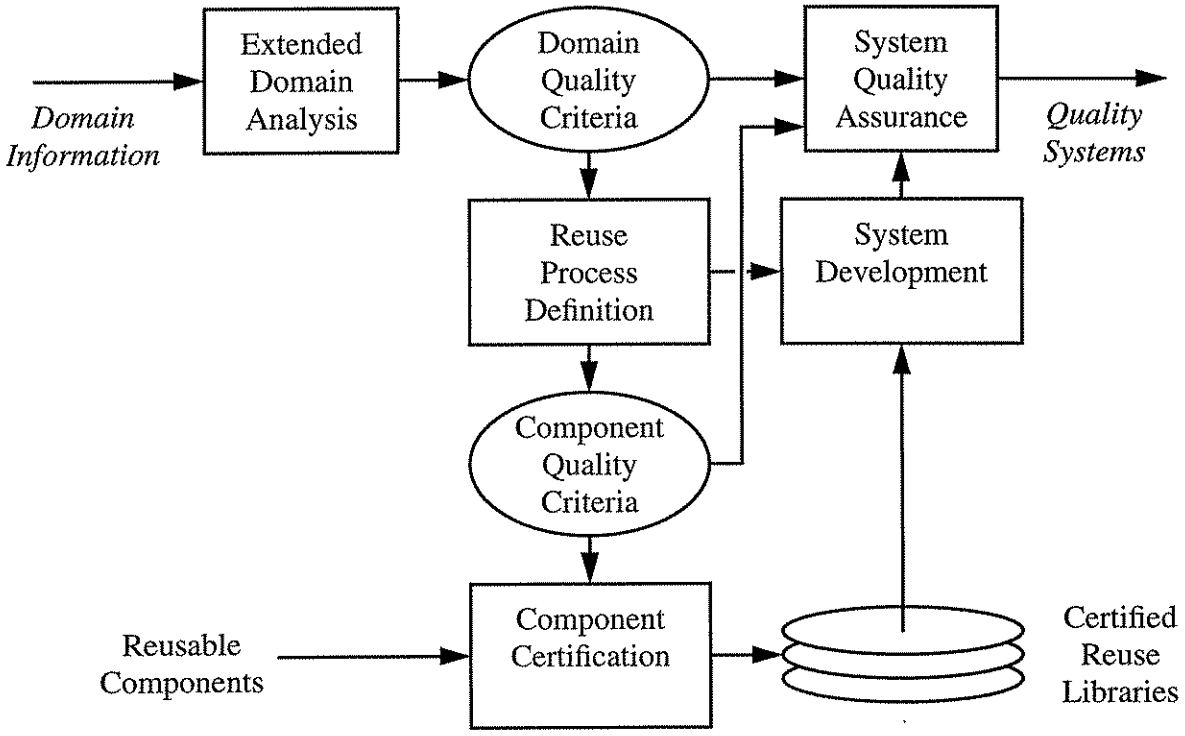


Figure 5 - Domain Analysis and Certification

- The domain itself requires high dependability. As telephone switches have become controlled by software, software flaws have become the most likely source of failures. A single failure in a single switch can mean that calls cannot be placed, which might cause significant and serious inconvenience. Further, faults can sometimes lead to a cascade effect, causing an entire switching network to fail. The software thus must be submitted to rigorous testing procedures. This includes testing each feature of the switch, testing an integrated set of features, and testing how the switch reacts to different combinations and load-levels of calls.

## 4.2 Analysis Process

Fig. 6 shows the process we used for performing our extended domain analysis. The process consisted primarily of meetings with domain experts to determine the scope and content of the testing domain. It also included analyzing system source code, requirements and designs documents, and development strategy documents. Also included were these activities:

- *Analysis of the organization's mission.*

  This yielded a description of the services that the organization is expected to provide to its customer set. Included is a set of goals, and for each goal, sets of assets and processes required to achieve that goal are described.

- *Analysis of the organization's strategy.*

  This yielded a description of how the organization needs to prepare for future changes to its product line, supporting technology, and operational environment. Associated with each change is a set of required assets and processes.

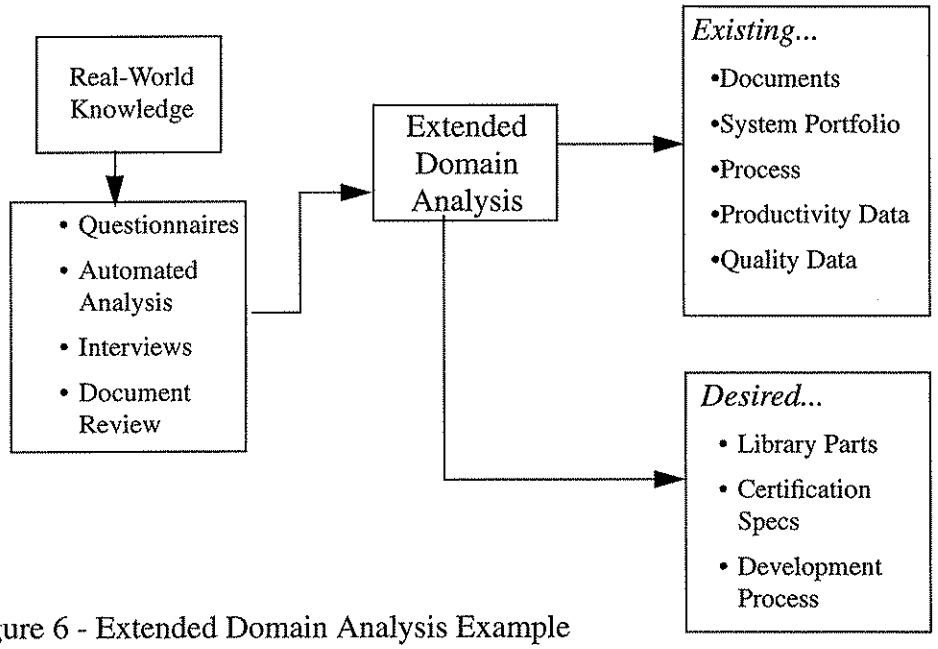


Figure 6 - Extended Domain Analysis Example

- *Analysis of production goals.*

  This yielded a description of the organization's production goals, the means by which these goals should be measured, and the assets and processes required to support them.

- *Analysis of the technological infrastructure.*

  This yielded a mapping between lifecycle phases, lifecycle work products, and development tools. The purpose is to assess how well the organization's current technology base supports its development process.

- *Analysis of subdomains.*

  This yielded a breakdown of the domain into its subdomains. For each subdomain, we identified required functionality and implementation models. For each implementation model, we identified design trade-off's, example systems, possible failures, and supporting sets of assets.

## 4.3 Initial Results

The main result of the work to date is a preliminary set of quality certification criteria for six subdomains within the message protocol verification domain. For each of these subdomains, we derived properties according to the six-category ISO 9126 quality classification scheme: functionality, portability, maintainability, performance, dependability and usability

As an example, one of the organization's goals was to make a subset of their systems cross-compatible between their Sun platform running Unix, and their PC platform running DOS. This had obvious ramifications on the portability characteristics of the components, since it prohibited them from using platform-specific constructs without adequate parameterization.

A side-effect of performing this process was that it forced the engineering team to identify and prioritize the organization's quality and productivity goals. This is a direct result of identifying which components would have the greatest impact on the development of new systems in the domain and determining what their quality attributes should be.

Additionally, the process caused the engineering staff to come to an agreement about the meaning of certain commonly-used terms within the domain. Engineers will often use the same word or phrase to mean different things, and, clearly, this can inhibit using work products across different projects.

# 5 Conclusions

We have presented an overview of domain analysis and concluded the traditional definition of domain analysis does not highlight domain quality considerations to the degree needed by a rigorous certification process. We consider this to be essential if reuse is to pay off as a productivity vehicle and we have defined the concept of extended domain analysis as a result.

To support the goal of facilitating the development of systems possessing significant quality characteristics, we have established a precise meaning for component certification and shown how the definition and extended domain analysis work together to provide a framework for developing high quality systems. We also reported our preliminary experience with extended domain analysis.

As often happens, more questions were raised than answered during the course of the project. Among the questions raised are

- How can the knowledge acquisition process be streamlined?
- How can the domain-driven certification process best be used by organizations at different levels of process maturity?
- What types of tool support will best facilitate the domain analysis and certification process?
- How should the certification properties be organized?
- How do the various quality standards proposed by such organizations as IEEE or ISO fit into our framework?

## Acknowledgments

It is a pleasure to acknowledge many useful discussions about the subject matter of this paper with Allan Willey of Motorola. This work was sponsored in part by the National Science Foundation under grant number CCR-9213427, in part by NASA under grant number NAG1-1123-FDP, and in part by Motorola.

# References

[BrB89] Brule, J.F., and A. Blount., *Knowledge Acquisition*, McGraw-Hill, Inc. New York, NY, 1989.

[Dun93] Dunn, M.F., *Cellular Tools Extended Domain Analysis*, Univ. of Virginia CS Tech. Report CS-93-50, August, 1993.

[DuK93] Dunn, M.F. and J.C. Knight, *Certification of Reusable Software Parts*, Technical Report CS-93-41, Department of Computer Science, University of Virginia, 1993.

[Kan90] Kang, K.C., S.G. Cohen, J.A. Hess, W.E. Novak, and A.S. Peterson, *Feature-Oriented Domain Analysis Feasibility Study: Interim Report*, SEI Tech. Report CMU/SEI-90-TR-21, August 1990.

[Kni92] Knight, J.C., *Definition and Framework for the Certification of Reusable Parts*, Proceedings International Software Quality Exchange '92, San Francisco, CA 1992.

[PPS92] Poore, J.H., T. Pepin, M. Sitaraman, F. L. Van Scoy, *Criteria and Implementation Procedures for Evaluation of Reusable Software Engineering Assets*, CDRL S45.11, March 28, 1992.

[REW93] *Proceedings of the Second Annual West Virginia Reuse Education and Training Workshop*. October, 1993.

[SPC92] *Domain Engineering Guidebook*, SPC Tech. Report SPC-92019-CMC, December 1992.

[Vol93] Vollman, T.E., *Software Quality Assessment and Standards*, IEEE Computer, Vol. 26, No. 6, June 1993.