Domain Engineering:
An Empirical Study

William Frakes
C. David Harris, Jr.

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

December 21, 2006

Domain Engineering: An Empirical Study

William Frakes
Virginia Tech

C. David Harris, Jr.
Virginia Tech

Domain Engineering: An Empirical Study

William Frakes and C. David Harris Jr.
Virginia Tech

Abstract: This paper presents a summary and analysis of data gathered from thirteen domain engineering projects, participant surveys, and demographic information. Taking a failure modes approach, project data is compared to an ideal model of the DARE methodology, revealing valuable insights into points of failure in the domain engineering process. This study suggests that success is a function of the domain analyst's command of a specific set of domain engineering concepts and skills, the time invested in the process, and persistence in difficult areas. We conclude by presenting strategies to avoid points of failure in future domain engineering projects.

1. Introduction

Domain engineering has emerged as an important topic in software engineering research and practice. While many methods have been developed to support domain engineering [2], there has been little empirical analysis of the domain engineering process. This paper conducts an empirical analysis of thirteen domain engineering projects in a university setting. Taking a failure modes approach, this report analyzed the collected project data and various project outcomes to identify points of improvement. Sources of information include the DARE books, participants' demographic information, and survey data taken during and after the domain engineering exercises. These artifacts were produced over the course of three years as part of a graduate-level, advanced software engineering course at Virginia Tech.

These resources present important opportunities to discover individual strategies for domain engineering. The data alone presents a contribution to domain engineering research; however, this report's analysis and conclusions likewise provide valuable insights for improving the craft of domain engineering.

2. Domain Engineering

2.1. Context

Software developers have always done software reuse. Operating system libraries and programming language syntactic elements are designed to be reused and have been successful in standardizing computer programming languages and methods.

There are many types of reuse. [Frakes and Terry 96] For example a programmer may reuse software by selecting lines of code from preexisting projects and copying into new applications. He/she may also reuse software functions or libraries. However, developing a framework to systematically engineer domain specific reusable software components requires an entirely different process and methodology.

The term *software engineering* was first popularized in a 1968 NATO conference with the hopes of applying structured engineering principles to the process of creating software [9]. As a discipline, software engineering is an evolutionary leap from the craft of computer programming. Traditionally, successful development is the result of intuition, hard work, and individual talent.

But as commercial demands for software increase, it becomes impossible to continue traditional ad hoc, single system development strategies [10]. Current software engineering research aspires to apply a more formal methodology to software development.

Software engineering models strive to manage the software development process. Improved requirements gathering, design, quality, cost, and time estimation are all goals of the various software engineering processes. The numerous software engineering processes - waterfall, iterative, capability maturity model, etc- all have the common goal of improving software development. An emerging goal of software engineering is to design software assets so that the software can be reused easily. Creating reusable software components in a particular domain is a goal of domain engineering.

2.2. Design for Reuse

Like software engineering, the field of software reuse has its origins in the late sixties. During this time, Doug Mcilroy of Bell Laboratories proposed software reuse as an area of study and recommended basing the software industry upon reusable components [7]. Other research contributions include Parnas' notion of programming families and Neighbors' idea of domain analysis [2]. Since this time, software reuse has been an active area of research.

Domain engineering recognizes that software development communities create software in an application area or a set of systems that share design decisions [2]. Commercial groups create software products in a given market area. Problem domains such a software metrics or vocabulary analysis inspire research, publications, and software from multiple sources.

The process of domain engineering has two phases. Domain analysis systematically analyzes preexisting software, architectures, and documents to capture their commonalities and variabilities to develop a generic architecture and templates. Domain implementation develops software artifacts that can be produced new systems within a domain.

2.3. Benefits

A core benefit of domain engineering is that development with reusable software assets – libraries, code generators, domain specific languages, contributes to improved software quality and productivity. Software that is reused is often tested more frequently than software that has a single use or function. Using reusable assets typically decreases development costs, for the reuse requires less time, less testing, and less new documentation. In addition, reusable assets often improve the accuracy of initial development estimates of time and cost [7].

2.4. **Literature Review**

Several Domain engineering methodologies exist, including Family-Oriented Abstraction Specification and Translation [8], Feature-Oriented Domain Analysis (FODA) [9], Organization Model Modeling (ODM), and Domain Engineering Method and Reusable Algorithmic Libraries (DEMRAL). This report uses the Domain Analysis and Reuse Environment methodology [6].

2.5. **Domain Analysis Reuse Environment**

The Domain Analysis and Reuse Environment (DARE) methodology is a multidisciplinary approach that has two phases: domain analysis and domain implementation.

Domain Analysis

In domain analysis, a domain analyst, with the help of a domain expert, collects domain source code, documents, and architectures. Through a systematic study of these domain sources, the domain analyst acquires an understanding of the commonalities and variabilities of domain systems. Specific activities include the identification and organization of important vocabularies, the development of feature tables, and the creation of facet tables and generic architectures. These products of analysis provide the material for domain implementation.

Domain Implementation

Software reuse can be divided into two types: *parts-based* and *formal language* based.
Parts-based reuse is better known. It typically introduces reusable software "parts" into a language. Examples of parts-based reuse include the reuse of programming functions, classes, and libraries. Java, for example, provides a large collection of reusable java libraries in the form of Java Application Resource (JAR) packages. Parts based reuse can also include language frameworks - interacting collections of objects, such as the C++ Standard Template Library, or the Microsoft .NET Framework.

The second type of reuse, formal language based, involves creating a general, formal language that has the reusable information embedded into it [8]. Formal language based reuse can be further divided into two types: programming language based and generative.

Domain specific languages that are created for a small problem area are called *Little Languages*.[Bentley] R, for example, is a statistics language that readily processes data sets and produces summary statistics and graphs with a few commands. Others examples of little languages include: Lex (Lexical Analysis), Yacc (Yet Another Compiler Compiler), Csound for dealing with sound, Make for managing software project compilation, Extensible Stylesheet Language Transformations (XSLT) for translating XML documents, and Ant (Anther Neat Tool) for managing Java projects.[20,21]

In the generative approach, domain knowledge is built into the generator such that based upon a few specifications, an application can be generated. AndroMDA, for example, is a model-driven architecture that translates UML into software that integrates with several common architectures, such as Spring, EJB, Struts, Hibernate, JSF, and Java. Other examples of language specific application generators are Sun's Netbeans and Microsoft Visual Studio. Both of these programming user interfaces contain a collection of default project types that can be generated automatically. They also facilitate adding commonly used application components and provide ready access to software libraries.

Systematic Reuse and Product Line Engineering

The idea of applying a product line approach to systematic software reuse has its origins in manufacturing, where products are produced in a structured way. This formal creation takes full advantage of the knowledge of the commonalities and variabilities of each release. Software Engineering Institute defines a software product line as "… a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way."

2.6. A Unique Opportunity

Measures of software reuse are difficult to estimate. In the past, individual organizations like NASA and Motorola have instigated reuse programs and reported improved reuse rates from ~20% to 79% [26], and 14% to 85% [27]. An industry-wide perspective, however, is a difficult statistic to come by. Research indicates that there are great differences between reuse practices among different industries, and the factors that influence the success and failures of software reuse practices are complex [29].

This study explored this complex failure space by capturing the efforts of thirteen individuals' first domain engineering exercises. Surveys of the participants in this study indicate they reuse software on average 30% across all the lifecycle objects in their organizations and 28% of the lifecycle objects they personally create. In addition, none had a work program for software reuse education or a means to measure levels of reuse. [Appendix C].

From a failure modes perspective, this study offers insight into the factors that contribute to both success and failure in domain engineering. The collection of data alone provides valuable information, but it is hoped that its analysis and the conclusions of this report provide helpful guidance for future domain engineering efforts, and will contribute more reuse rates in the workplace.

3. The Study

3.1. Demographics

In this study, thirteen subjects completed domain engineering in several domains using DARE-COTS [1] as part of an advanced graduate course in software engineering. Projects were completed in a 14-week period from January to April of 2005 and January to April 2006. None of the subjects had prior knowledge of domain engineering. Demographic information on the subjects is reported in table 3,1.

Demographic information includes the subject's degree level, the subject's degree area he/she was pursing (computer science or information systems), the subject's years of industrial experience, the subject's primary programming language, the subject's chosen domain, and the subject's language used in the project domain implementation.

Subject	Level	Degree	Experience	Languages in order of familiarity	Domain	Implementation Language
1	M.S.	MIS	0	NA	Conflation	Java
2	M.S.	CS	9.5	VB.NET, C#, VB6, PL/SQL, Java C++	AHLTA Longitudinal Domain Book	NA
3	M.S.	CS	5	Java, PL/SQL, Perl, C	Open source java metrics	Java
4	M.S.	CS	3	C++	Symmetric encryption	C
5	M.S.	CS	15	C++,C, Fortran, Assembler, Java	Simple metrics	Perl
6	M.S.	CS	10	C++, C#, Java, Perl	Sentence Alignment Systems	C#
7	Ph.D.	CS	1	Java, C++, C	Conflation	Java
8		CS	13	C++, C	Conflation	Perl
9	MS	CS	3	Java, Ruby, C++	Blog domain	NA
10	M.S.	CS	7	Visual Basic, C++, C	Personal information management systems	Visual Basic
11	M.S.	MIS			Conflation	NA
12	M.S.	CS	1	C++, Pascal, C	Static code metrics	C
13	M.S.	CS	6	Java, SQL, C#	Object Oriented Software Metric Tools	Java

Table 1: Participant Demographics

3.2. **Assignment**

Students were given the following semester-long assignment:

- Domain engineering using DARE for one of the following domains: software metrics, conflation algorithms, or one of your choice.
- Collect process metrics for your project such as:
    - Time for each step
    - Log of what you did
    - Produce size/ complexity metrics
- Create one or more reusable components for your domain and/or Create a little language for your domain and/or Create an application generator for your domain

3.3. **DARE Book Structure**

The completed DARE book contained the following sections:

DARE

Definition: (Domain Analysis Reuse Environment) [1]

DARE Book provides a detailed specifications of the domain

Domain Sources

Vocabulary Analysis

Architecture Analysis, code analysis

Summary information, glossary, bibliography index, and appendix

DARE Book

Domain sources

Source documentation

Source code

Source notes

System description

System architectures

System feature tables

Vocabulary

Basic vocabulary, facets, synonyms, templates, thesaurus

Architecture

Generic architecture

Generic features

Code structure

Reusable Components

Summary

Domain Glossary

Bibliography

Index

Appendix

3.4. The Paper Outline

The remainder of this paper is as follows: Section 4 defines an idealized model or the DARE methodology for domain engineering. Data collected from the thirteen projects is presented in section 5, and then data is analyzed for points of failure in section 6. These observations are summarized in section 7's themes of failure, followed by section 8's principles of success. Section 9, 10, and 11 then suggest practical implications of these observations for future DARE projects endeavors and future research.

4. DARE Methodology Model

4.1. Domain Analysis

Domain engineering's fist phase, domain analysis, is the process of analyzing a given domain to derive domain models. Sources of this analysis include exemplar code, exemplar text and domain expert information. Examples of models include generic architectures, facet tables, feature tables and templates.

4.1.1. Scoping the Domain

Domain analysis begins by selecting a domain that can be bounded and scoped to a manageable level. With the help of a domain expert, the domain analyst scopes a domain and creates a formal domain model. Scoping the domain involves gathering suitable domain exemplars, documents, and notes, then describing the domain verbally. Set notation may be necessary to state clearly what is in and what is not in the domain [25]. The success of this is dependent, in part, on the degree to which a chosen domain is suitable for analysis.

4.1.2. Domain Suitability

The suitability of a domain for analysis is a function of several factors. First among these is the availability of at least three exemplar systems. The suitability of exemplar systems for domain analysis is a function of system complexity, stability, size, and formality. System complexity increases with the number of system types, and the complexity of the system source code.
System stability is related to the quality of the project. Elements of stability of an exemplar system include: code quality, maintenance support, release history, experience of authors, comment to code ratio, coding style, and complexity metric.

The size of a system can be a determining factor in whether the system can be scoped to a manageable size for domain analysis. For instance, the domain of software metrics might better be scoped to static complexity analysis. A system with high formality will contain documentation, architectures, and ample software comments. An exemplar system produced in a formal methodology contributes to stability.

4.1.3. Vocabulary Analysis

The vocabulary analysis steps include gathering the domain exemplar code, exemplar text and domain expert information, and compiling an initial word set. Then using automated processes and domain knowledge, this set of words is reduced to a manageable set of key words. Diagram 1 represents the vocabulary analysis methodology. Various vocabulary analysis methods include the use of stop lists, stemming, conflation, clustering, and frequency.

Cluster tables group words together around a point of commonality. The points of commonality become the column titles of the facet table with the cluster words becoming the rows or points of variability. The facet table maps directly to a template whereby descriptive words describe the generic system with the facet table's columns mapping to variables. This template then is used in the creation of the reusable asset.

Diagram 1: Vocabulary Analysis

The cluster tables, facet table, and template are all derived from the key word set. At each step of the process, domain analyst system knowledge plays a key role in refining the tables.

4.1.4. Architectural Analysis

The domain analyst, along with the domain expert, creates a system feature table for each exemplar system. System feature tables describe the attributes of each of the systems.
System architectures are architectures available in the documentation of the exemplar systems.
A generic architecture is formed, in large part, by an analysis of the set of available system architectures and the template produced by the vocabulary analysis.

4.1.5. Software Analysis

The exemplar code is analyzed using a variety of techniques to understand how it can be incorporated in a reusable asset. Software components, such as classes, functions, and libraries, can be identified and incorporated into a parts-based reusable asset. Software metrics can be used to identify areas of complexity or maintainability that can assist in a redesigned, reusable component. This process is aided by analysts that possess domain expertise, programming language knowledge, and an understanding of software analysis metrics.

4.2. **Domain Implementation**

Reusable assets are derived by domain implementation based on domain models Assets can be parts based components, domain specific languages, or application generators. In a parts-based implementation, domain knowledge gained from software analysis can contribute to creating reusable language components, such as classes and libraries. These parts can be incorporated and reused in future software applications. In a formal language implementation, the template and generic architectures provide frameworks for creating domain specific languages or application generators.

5. **Project Data**

The following section presents data collected from thirteen DARE projects. Data considerations are organized by measures of time, measures of size, domain scope, vocabulary analysis, architectural analysis, and dare book tables., The Discussions of their implications can be found in section 6.

5.1. **Activities Time Log**

Participants were instructed to provide a log of time spent on each stage of the process.
All times are given in hours except where noted.

Table 2. Activities Time Log Entries

Project Number	1	2	3	4	5	6	7	8	9	10	11	12	13
Book Creation		12		20				2					
Finding good tools		16.25	50										
gathering source information	8	2		6			1 ,,, 2 weeks	13				38	
Documents											9		2
source code											4		0.5
system descriptions								0.5					
system architectures							2 … 3 weeks	8			8	16	
system feature tables						2	3 days	0.5				20	3
source notes													1
expert systems						12		5					0.5
study of domain				46				3			9		
domain scope	1						3 weeks	3			2		
vocabulary analysis	4	21	20	12	30	13	2 weeks	0	0	0	3		
basic vocabulary					1								1
frequency analysis						4							
cluster analysis						8							2
facet table					1	2	4 days	2				20	0.5
synonym table					1	0.5							0.5
template					1	1							0.5
thesaurus					1	0.5							0.5
vocabulary notes					1			1					
code analysis	30		3	2	2	4	5 days	1			3		
source notes								3					
Arch analysis	5	14	60	10	2								
generic architecture					2	4		4			6	16	0.5
generic feature table					2		1.5 days	1			2	16	
architecture notes					2		1.5 days	3					
implementation/ reusable component	5		55	80	4	24	5 weeks	5		55	3		0.5
reusable algorithm								62					
glossary								5					2
testing										15			0.5

5.1.1. Number of time log entries for each participant.

This table considers the number of time log entries made by each participant. The variability of these measures may play a role in understanding different project outcomes.

Table 3 Number of Time Log Entries												
3	5	5	6	6	7	10	10	12	13	14	19	3

Statistical Summary 1 - Number of time Log Entries			
Sample Size, n:	13	Range	16
Mean	8.69	Minimum	3
Median	7	1st Quartile	5
Midrange	11	2nd Quartile	7
Variance	23.06	3rd Quartile	12
Standard Deviation	4.80	Maximum	19

5.1.2. Total time creating the DARE project

This table considers the total time invested in DARE project. Time investment may be an indicator of project success.

Table 4. Total Time of DARE project												
53	65.25	188	176	65	75	NA	1222	NA	70	49	126	13

Statistical Summary 2, Total Book Time Statistical Summary			
Sample Size, n:	11	Range	175
Mean	91.11	Minimum	13
Median	70	1st Quartile	53
Midrange	100.5	2nd Quartile	70
Variance	3014.79	3rd Quartile	126
Standard Deviation	54.9	Maximum	188

5.1.3. Time Preparing the Domain

For purposes of analysis the report groups together the time log steps of table 2 from “finding good tools” to “domain scope”. These steps comprise an important phase of the DARE book, referred to in this paper as “preparing the domain”.

Table 5. Total time preparing the domain												
9	18.25	50	52	15	14	NA	33	NA	NA	32	74	7

Statistical Summary 3, Domain Preparation Time			
Sample Size, n:	10	Range	67
Mean	30.22	Minimum	7
Median	25.125	1st Quartile	12
Midrange	40.5	2nd Quartile	25.12
Variance	498.83	3rd Quartile	50
Standard Deviation	22.3	Maximum	74

5.1.4. Time invested in vocabulary analysis

In this report, vocabulary analysis encompasses the creation time log steps, table 2, from “vocabulary analysis” to “vocabulary notes”. Diagram 1 of section 4.1 graphically shows the collection of steps involved in vocabulary analysis.

Table 6. Total time spent on vocabulary analysis												
4	21	20	12	36	29	NA	3	NA	NA	3	20	5

Statistical Summary 4 - Vocabulary Analysis Time			
Sample Size, n:	10	Range	33
Mean	15	Minimum	3
Median	16	1^{st} Quartile	4
Midrange	19.2	2^{nd} Quartile	16
Variance	129.55	3^{rd} Quartile	21
Standard Deviation	11.38	Maximum	36

5.1.5. Time Implementing Reusable Assets:

The time for implementing reusable assets is derived from the creating time log entries, table 2, “implementing reusable component”, “reusable algorithm” and “testing”

Table 7. Time for Domain Implementation													
implementation/ reusable component	5		55	80	4	24	5 weeks	5		55	3		0.5
reusable algorithm								62					
Testing										15			
Total	**5**		**55**	**80**	**4**	**24**		**67**		**70**	**3**		**0.5**

Statistical Summary 5, Time Implementing Reusable Assets			
Sample Size, n:	9	Range	79.5
Mean	34.2	Minimum	0.5
Median	24	1st Quartile	4
Midrange	40.25	2nd Quartile	24
Variance	1108.19	3rd Quartile	67
Standard Deviation	33.28	Maximum	80

5.1.6. Time Groups

In the category of implementing reusable assets, there were to two distinct groups: those who spent 50 hours or more, and those who spent 5 or less hours. The data points break down as follows:

Table 8. Time spent implementing Reusable Assets		
<= 5	0.5, 3, 4, 5	
>=50	55, 67, 70,80	

5.1.7. Time Outliers

Outliers of the time entry log, table 2, are listed below. Although there is a good deal of variation of the time entry log, outliers in the following categories are noteworthy. The following table lists areas of the project where students experienced difficulty completing the task.

Time outliers	
Finding good tools	50
Gathering Source Information	38
Study of domain	46
Architectural Analysis	60

5.2. Book size

Book size was measured by the number of words, number of lines, and number of pages.
There was considerable variation in book size due to the different individual approaches to book construction. Book style ranged from those who included domain sources with their projects - the system exemplar code, vocabulary analysis results, code analysis, etc., to those whose books were little more than an outline with references.

Two areas that greatly increased the size of books were the inclusion of system source code and intermediary vocabulary analysis tables. In one project, for example, a 422 page book was reduced to 20 pages after removing 376 exemplar source code and 26 pages of vocabulary analysis.

Book Size	1	2	3	4	5	6	7	8	9	10	11	12	13
Words	12966	6448	3101	6334	3218	3350	10396	1416	2962	74939	28068	3396	1083
Lines	4021	4757	457	3047	1202	683	3,636	613	428	26648	6628	1076	227
Pages	81	68	17	60	17	19	65	32	19	422	125	20	8
Words - source code	10,060	6,185	3101	6334	3218	3350	3282	1416	2962	9,104	17314	3396	1083
Lines - source code	3,103	4,494	457	3047	1202	683	1518	613	428	6653	3658	1076	227
Pages - source code	62	62	17	60	17	19	28	32	19	46	78	20	8
Words - vocabulary frequency analysis	10,060	3,060	3101	6334	3218	3350	10396	1416	2962	68869	24008	3396	1083
Lines - vocabulary frequency analysis	3,103	1,257	457	3047	1202	683	3,636	613	428	20598	4598	1076	227
Pages - vocabulary frequency analysis	62	41	17	60	17	19	65	32	19	379	91	20	8
Words - source and vocabulary	1,060	2,787	3101	6334	3218	3350	3282	1416	2962	3036	13252	3396	1083
Lines -source and vocabulary	3,103	993	457	3047	1202	683	1518	613	428	604	1627	1076	227
Pages - source and vocabulary	62	35	17	60	17	19	28	32	19	11	44	20	8

The following charts display DARE book size by word count, line count, and page count.

5.2.1. Book Word Count

Chart 6 shows the total word count for DARE books. These measurements do not remove exemplar source code or vocabulary analysis.

Table 6, Word Count in DARE book.												
12966	6448	3101	6334	3218	3350	10396	1416	2962	74939	28068	3396	1083

Statistical Summary 6 : DARE Book Word Count			
Sample Size, n:	13	Range	73856
Mean	12129	Minimum	1083
Median	3396	1st Quartile	3101
Midrange	38011	2nd Quartile	3396
Variance	4.086476e+8	3rd Quartile	10396
Standard Deviation	20215.03	Maximum	74939

5.2.2. Word Count without exemplar code and vocabulary analysis

Here shows the total word count for all projects with exemplar source code and vocabulary analysis removed. These measurements have less variance with source code and vocabulary analysis removed.

Table 7, DARE book word count with exemplary code and vocabulary frequency analysis removed.												
1060	2787	3101	6334	3218	3350	3282	1416	2962	3036	1924	3396	1083

Statistical Summary 7: Book Words minus source and vocabulary			
Sample Size, n:	13	Range	5274
Mean	2842.23	Minimum	1060
Median	3036	1st Quartile	1924
Midrange	3697	2nd Quartile	3036
Variance	1.863497e+6	3rd Quartile	3282
Standard Deviation	1365.1	Maximum	6334

5.2.3. Line Count – code and vocabulary

Chart 8 displays the total line count for all projects with exemplar source code and vocabulary analysis removed.

Table 8, DARE book line count without exemplary source code and vocabulary frequency analysis												
3103	993	457	3047	1202	683	1518	613	428	604	438	1076	227

Statistical Summary 8: Line Count minus code and vocabulary			
Sample Size, n:	13	Range	2876
Mean	1106.84	Minimum	227
Median	683	1st Quartile	457
Midrange	1665	2nd Quartile	683
Variance	893486.8	3rd Quartile	1202
Standard Deviation	945.24	Maximum	3103

5.2.4. Page Count – code and vocabulary

Here shows the total page count for all projects with exemplar source code and vocabulary analysis removed

Table 9, DARE book page count without exemplary source code and vocabulary frequency analysis												
62	35	17	60	17	19	28	32	19	11	17	20	8

Statistical Summary 9: Page Count minus code and vocabulary			
Sample Size, n:	13	Range	54
Mean	26.53	Minimum	8
Median	19	1st Quartile	17
Midrange	35	2nd Quartile	19
Variance	291.26	3rd Quartile	32
Standard Deviation	17.06	Maximum	62

5.3. **Domain Scope**

Domain analysts are to describe their domains verbally, describing what is in and what is not in the domain, using set notation if necessary. These measurements capture this domain scoping statement, by measuring the number of words and sets in the statement**.**

5.3.1. **Number of Words**

Chart 10 displays the number of words present in each projects domain scope statement

Table 10.a , Domain Scope Word Count												
64	NA	178	1075	141	688	160	229	NA	255	NA	NA	18

Statistical Summary, word count in the domain scope section

Statistical Summary 10: Domain Scope Word Count			
Sample Size, n:	9	Range	1057
Mean	312	Minimum	18
Median	178	1st Quartile	141
Midrange	546.5	2nd Quartile	178
Variance	118990.5	3rd Quartile	255
Standard Deviation	344.95	Maximum	1075

5.3.2. **Mathematical Model using Set Notation**

For those projects that had domain statements, table 10.b notates if the project used set notation.

Table 10.b Presence of set notation.												
1	NA	0	1	0	0	0	1	NA	0	NA	NA	0

Three of the nine projects that had a section devoted to scoping the domain, three or 33%, used a set based mathematical model to describe their domains.

Two individuals who did not use set notation used different notation. One participant used pseudo code in the scope statement, and the other used a top-down function and connector architecture.

5.3.3. **Number of sets**

Table 10.c displays for those projects that used set notation, how many sets were used in their model.

Table 10.c, Number of sets in domain scope												
4	NA	NA	20	NA	NA	NA	8	NA	NA	NA	NA	NA

Of the three projects that used set notation to scope their domains, the number of sets used in the model was 4, 8, and 20.

5.3.4. **Domain Sources**

A key component to domain analysis is the selection of domain sources. These exemplars include source code, documents, and architectures.

5.3.4.1. **Count of Exemplar Sources**

Chart 11 shows the number of number of examples of domain source code used for each project's domain analysis.

Table 11, Number of Exemplar sources												
3	7	3	3	3	4	3	8	4	3	3	3	3

Statistical Summary, Exemplar Count

Statistical Summary 11: Exemplar Count			
Sample Size, n:	13	Range	5
Mean	3.86	Minimum	3
Median	3	1st Quartile	3
Midrange	5.5	2nd Quartile	3
Variance	2.80	3rd Quartile	4
Standard Deviation	1.67	Maximum	8

5.3.4.2. Count of Exemplar Documents

Chart 12 displays the number of domain documents chosen for the DARE book analysis.

Table 12a Number of Domain Documents												
5	3	3	7	3	4	4	19	3	3	7	3	3

Statistical Summary 12a, Exemplar Document Count			
Sample Size, n:	13	Range	16
Mean	5.15	Minimum	3
Median	3	1st Quartile	3
Midrange	11	2nd Quartile	3
Variance	19.47	3rd Quartile	5
Standard Deviation	4.41	Maximum	19

5.3.5. **System Descriptions Count**

Each project contained system descriptions. This table shows the number of system descriptions.

Table 12.b ,System Descriptions Number and Type												
3 fictitious	NA	3	5	3	2	3	6	3 paragraphs	3 paragraphs	3 paragraphs	3 paragraphs	3

The majority of the projects contained system descriptions. A system template is presented in Appendix D. Of this set, three projects did not follow the system template; rather, these projects described the system verbally in a paragraph or less. 92% of the projects contained system descriptions.

5.4. Architectural Analysis

As part of the domain analysis, participants undertook an architectural analysis of their domains. There were three points of data taken with their analysis: the number of system architectures in each DARE book, the types of architectures in this set, and the word count of the architectural analysis section of the domain book.

5.4.1. Number of system architecture

This table contains the number of architectural images contained in each individual's DARE book.

Table 13a, Number of System Architectures												
9	14	3	26	4	3	3	6	0	3	5	6	3

Statistical Summary 13: System Architecture Count			
Sample Size, n:	13	Range	26
Mean	6.53	Minimum	0
Median	4	1st Quartile	3
Midrange	13	2nd Quartile	4
Variance	46.26	3rd Quartile	6
Standard Deviation	6.80	Maximum	26

5.4.2. Types of Architecture

This table lists the architectural type for each architectural image in the DARE book.

Table 13.b, Types of Architecture						
1	**2**	**3**	**4**	**5**	**6**	**7**
3 flow chart 3 top down, 3 use case	7 activity, 7 module	Object relationships. Class diagram,	Functional flow and their actions	functions and connectors	Pseudo code (1), function and connectors (3)	function and connectors
8	**9**	**10**	**11**	**12**	**13**	
Flow (functions, connectors, data)	class diagrams	flow (functions and connectors)	flow (functions, connectors)	data flow, data flow, data flow, top down, batch sequential	class diagrams	

5.4.3. **Word Count**

This table shows the number of words for each projects architectural analysis section.

Table 14.a, Word count architectural analysis												
852	115	0	1767	179	0	0	0	0	0	30	126	0

Chart 14

Statistical Summary 14: Architectural Analysis word count			
Sample Size, n:	13	Range	1767
Mean	236.07	Minimum	0
Median	0	1st Quartile	0
Midrange	883.5	2nd Quartile	0
Variance	265476.2	3rd Quartile	126
Standard Deviation	515.24	Maximum	1767

Seven projects included only the architectures diagram but contained no descriptive text.
Six projects included descriptive text. The number of words ranged from 30 to 1767.

5.4.4. **System Feature Tables**

Each analyst was to create feature tables of his/her domain system exemplars. The table below shows the number of tables included in his/her system feature table section.

5.4.4.1. **Number of Tables**

Table 14.b Number of System Feature Tables

1	1	3	3	3	1	3	6	1	1	1 (3 combined)	3	1 (3)

Number of Features

This table shows the number of features in each feature table.

Table 14.c, Number of Features in System Feature Tables											
4	11	5,11,5	16x3,17x3,20x3	11,11,6	9x4,	10,10,10	6 (6x1 tables)	12	38x3	8	13,14,18	7

There was a great variety of interpretations of what constituted a feature table.

5.5. **Vocabulary Analysis**

The log of activity included the vocabulary analysis activities. There was a great deal of variety of interpretation as to what labels to use and what constituted vocabulary analysis. Therefore it is helpful to redisplay those elements that constitute the vocabulary analysis phase. Due to the variety, discussing them individually is impractical, but looking at the total time spent on vocabulary analysis is.

5.5.1. Time invested in vocabulary analysis

Table 4, Total Time Spent on Vocabulary Analysis												
4	21	56	12	36	26	NA	3.00	NA	NA	3	20	5

Statistical Summary 4: Vocabulary Analysis Time			
Sample Size, n:	10	Range	33
Mean	15	Minimum	3
Median	16	1st Quartile	4
Midrange	19.2	2nd Quartile	16
Variance	129.55	3rd Quartile	21
Standard Deviation	11.38	Maximum	36

Of this group there is one outlier, whereby an individual spent 56 hours on vocabulary analysis.

These times can be viewed a two groups – those who spent 12 hours or more and those who spent five hours or less

Table 4.b, Total time spent on vocabulary analysis	
<=5	3,3,4,5
>=12	12, 20,21,36

5.5.2. **Manual or Automatic**

Vocabulary analysis consists of a combination of automated and manual processes. The following table displays whether an analyst uses automatic, manual, or a combination of both methods.

Table 14.c, Vocabulary Analysis: manual, automatic, both												
Manual	Automatic	automatic	both	Automated	both	NA	both	both	both	both	both	manual

5.5.3. **Original Set**

Analysts begin with a raw set of domain words to be analyzed. For those who reported their original set, the figures are recorded below.

Table 14.c, Original Set Word Count												
		thousands	thousands	thousands	thousands	thousands				10790		

5.5.4. **Number of words key word set**

Following a series of automatic and manual vocabulary analysis methods, analysts derive a key word set with which to create facet tables. The number of each projects key word set is shown in the following table.

Table 15, Number of word in keyword set												
39	939	650	34	4681	39	48	65	14	4117	32	NA	188

Chart 15

Statistical Summary15: Key Word Set Size			
Sample Size, n:	12	Range	4667
Mean	903.83	Minimum	14
Median	56.6	1st Quartile	36.6
Midrange	2347.5	2nd Quartile	56.5
Variance	2.764544e+6	3rd Quartile	794.5
Standard Deviation	1662.692	Maximum	4681

5.5.5. **Key word set, minus outliers**

Four individuals' vocabulary key word sets contained a key word set greater than 600. This group skewed the group that derived manageable set of vocabulary. This set is listed below.

Table 16.a, Key word set												
39	OUTLIER	OUTLIER	34	OUTLIER	39	48	65	14	OUTLIER	32	NA	188

Chart 16

Statistical Summary 16: Log Entry Count			
Sample Size, n:	8	Range	174
Mean	57.375	Minimum	13
Median	39	1st Quartile	33
Midrange	101	2nd Quartile	39
Variance	2993.696	3rd Quartile	56.5
Standard Deviation	54.71	Maximum	188

5.5.6. Presence of Word Clusters

Table 16.b displays the number of projects that used word clusters in their vocabulary analysis**.**

Table 16.b, Presence of word clusters												
0	0	0	0	0	1	1	0	0	0	1	0	0

Out of the 13 projects, 3 or 23% arranged keywords in clusters.

5.5.7. Facet Table

Vocabulary analysis involves organizing the domain's key words into facet of commonality and variability. This is displayed with a facet table. This section shows measurements of the facet table.

5.5.7.1. Number of facets

The key word set, ideally, is used to create a single facet table; this word set represents the commonalities and variabilities of the set of domain systems being analyzed. Below is the number of facets in the feature tables.

Table 17.a, Number of Facets												
7	8	3	4	3	7	10	6	6	6	4	NA	4

Chart 17

Statistical Summary 17: Facet table facet count			
Sample Size, n:	12	Range	7
Mean	5.66	Minimum	3

Median	6	1st Quartile	4
Midrange	6.5	2nd Quartile	6
Variance	4.60	3rd Quartile	7
Standard Deviation	2.14	Maximum	10

5.5.7.2. **Variability**

A facet table's variability is the number of columns in a facet table. The following table shows the number of variable elements in each domain's facet table.

Table 17.b, Facet Table variable count for each column						
1	**2**	**3**	**4**	**5**	**6**	**7**
8.2. variability	NA	3,3,6,6,8,6,4	7,2,3,2,2,2,2,2	6,14,3	3,3,4	3,3,3,4,4,3,3
8	**9**	**10**	**11**	**12**	**13**	
3,3,3,4,4,3,3	2,2,4,3,6	23,23,9,12,10,18	2,3,2,11,4,3	3,5,7,3,2,3	2,2,5,2	

5.5.7.3. **Facet table complexity - Product**

A facet table's commonalities and variabilities are a representation of the number of feature choices available for a given generic system. This complexity can be measured by the product of each facet table's variable elements.

Table 18.a, Facet Table Complexity (Product)												
62208	1344	252	9216	36	3888	288	10283760	1584	1890	40	NA	504

Table 18.a contains two outliers in project one and eight. Table 18.b and chart 18, below, display the facet complexity with these two outliers removed.

Table 18.b Facet Table Complexity (Product)												
OUTLIER	1344	252	9216	36	3888	288	OUTLIER	1584	1890	40	NA	504

Statistical Summary 18: Facet Complexity by Product			
Sample Size, n:	10	Range	9180
Mean	1904.2	Minimum	36
Median	924	1st Quartile	252
Midrange	4626	2nd Quartile	924
Variance	8.009142e+6	3rd Quartile	1890
Standard Deviation	2830.043	Maximum	9216

5.5.7.4. **Facet table complexity - Sum**

A second way to capture each facet table's complexity is to study the sum of each column's variability count.

Table 19, Facet Table Complexity (Sum)												
36	22	23	38	10	23	17	95	25	23	11	NA	20

Chart 19

Statistical Summary 19: Facet Table Complexity (Sum)			
Sample Size, n:	12	Range	85
Mean	28.58	Minimum	10
Median	23	1st Quartile	18.5
Midrange	52.5	2nd Quartile	23
Variance	506.08	3rd Quartile	30.5
Standard Deviation	22.49	Maximum	95

5.5.8. Template

The template is derived, ideally, from the facet table. The template is comprised of descriptive works and variables.

5.5.8.1. Words

Chart 20**,** below displays the number of words in each template description

Table 20, Word Count of the Template Description												
52	0	66	95	29	224	104	32	55	88	15	NA	83

Statistical Summary 20: Number of Template Words			
Sample Size, n:	12	Range	224
Mean	70.25	Minimum	0
Median	60.5	1st Quartile	30.5
Midrange	112	2nd Quartile	60.5
Variance	3418.56	3rd Quartile	91.5
Standard Deviation	58.46	Maximum	224

5.5.8.2. Template Variable Word Count

This section considers the number or variables in the template description.

Table 21, Variable count of template												
7	0	12	12	3	32	10	6	6	20	3	NA	16

Chart 21

Statistical Summary 21: Template Variable Count			
Sample Size, n:	12	Range	32
Mean	10.58	Minimum	0
Median	8.5	1st Quartile	4.5
Midrange	16	2nd Quartile	8.5
Variance	78.44	3rd Quartile	14
Standard Deviation	8.85	Maximum	32

5.6. Code Analysis

Of those projects that had software analysis, the following sets of data describe the time invested and the methods used.

5.6.1. Time Log Entries for Software Analysis

Table 21.b redisplays the projects' time log entries for code analysis.

Table 21.b, Time Log Entry of Code Analysis												
30	NA	3	2	2	4	5 days	1	NA	NA	3	NA	NA

5.6.2. Presence of Source Analysis

The following table contains a 1 for those whose project contained vocabulary analysis and 0 for those who did not.

Table 21.c Was their code analysis?												
1	0	0	1	0	1	1	1	0	1	0	0	0

- Six of thirteen, 46%, had some code analysis.
- One said it did not contribute
- One who had no code analysis reported "Due to the differing intents of the authors, their differing Java skill levels, and their differing use of program and GUI generators, we find it difficult to perform a direct code analysis of the three programs and wouldn't use any of them in a real-world application"
- Project 3 and 11 both indicated three hours of code analysis, but did not contain any artifacts of that analysis in the project. Project 3 discussed the benefits of he analysis. Project 11 contained no code analysis entry in the DARE project.

5.6.3. **Method**

This table indicates if the code analysis was performed manually or with some automated tools.

Table 21.d Vocabulary analysis: manual or automated?												
M	NA	M	A	NA	A	A	A	NA	A	NA	NA	NA

- Three of seven, 42%, used all manual methods
- Four of seven, 57%, used automated tools in whole or in part.
- One used completely manual inspection of code
- One listed all the functions of the system.
- One who used automated tools expressed the following frustration trying to find a tool to analyze Perl: "For Vanilla and Align CMU, I attempted to use the suggested cia and cflow tools. Cia seemed to be decommissioned (I could not find a functioning copy). There was a subsequent release called Acacia, but I also couldn't find a copy of this. Therefore, the analysis used only the cflow tool. After many fruitless searches, gave up on analyzing Perl. Apparently, there are no static code analysis tools for Perl. The main Windows implementation of Perl, ActiveState Perl, did not have a static analysis tool, though it had such tools for other languages. In any case, Moore's implementation does not use subroutines or classes, so the static analysis would've been useless

anyways. Also, when I was contemplating using more systems, I found a nice tool for Java called DependencyFinder. Even though I eventually decided to remove GMA NYU from later analyses, I include the static code analysis results for completeness”

- One project who invested two hours in software analysis, contained the following book entry: “Due to the differing intents of the authors, their differing Java skill levels, and their differing use of program and GUI generators, we find it difficult to perform a direct code analysis of the three programs and wouldn't use any of them in a real-world application.”

5.7. Generic Architectures

From the collection of system architectures, part of the DARE process is to derive a small number of generic architectures, representing the domain.

5.7.1. Number of words

This table contains the number of words in the DARE book devoted to the generic architecture section.

Table 22 Number of words in Generic Architecture section												
349	0	403	91	0	0	136	0	300	24	463	75	0

Chart 22

Statistical Summary 22: Word Count Generic of Architecture			
Sample Size, n:	13	Range	688
Mean	155	Minimum	0
Median	77	1st Quartile	0
Midrange	344	2nd Quartile	77
Variance	42133	3rd Quartile	171
Standard Deviation	205.26	Maximum	688

5.7.2. **Diagram count**

This diagram contains the number of generic architectures in the DARE book representing the domain.

Table 23.a, Architectural Diagram Count												
1	4	1	2	1	1	1	1	2	1	2	2	1

- 100% of the projects contained a Generic Architecture diagram
- Eight of thirteen projects contained one generic architecture diagram.
- Three projects contained two generic architectures.
- One project contained four generic architectures.

Statistical Summary 23: Number of Generic Architectures			
Sample Size, n:	13	Range	3
Mean	1.53	Minimum	1
Median	1	1^{st} Quartile	1
Midrange	2.5	2^{nd} Quartile	1
Variance	0.76	3^{rd} Quartile	2
Standard Deviation	0.87	Maximum	4

5.7.3. **Type of Architecture**

This table shows the architectural type in each project's Generic Architecture section

Table 23.b, Type of Generic Architecture						
1	**2**	**3**	**4**	**5**	**6**	**7**
functions and connectors … 13 functions (depth 4)	3 module, 1 activity	api , state and function	See note	functions and connectors (12 functions depth 3)	functions and connectors	Flow (function connectors data)
8	**9**	**10**	**11**	**12**	**13**	
flow (function connectors data)	abstract flow	function and connectors	NA	top down	NA	

5.8. Generic Feature Table

The generic feature table combines the system feature tables into one table. Columns represent the systems, and rows enumerate the system features. Each feature contains points of variability. The following three tables consider these measurements for project generic feature tables.

5.8.1. Presence of a generic feature table.

This table shows which projects contains a generic feature table. Nine of the thirteen projects, or 69% contained a generic feature table.

Table 23.c, The number of tables of the generic feature table.												
1	0	1	0	0	1	1	1	1	1	1	1	0

5.8.2. Number of features

This table shows the number of features contained in each project's feature table.

Table 23.d, number of features												
3	0	18	3	3	9	8	5	4	8	8	25	NA

- Project 1 displayed a single table with verbal descriptions of each generic feature.
- Project 6 displayed the same table for both system feature table and generic feature table
- Projects 3 and 12 have 18 and 25 features respectively. These may be too many features.

5.8.3. Points of variability

The generic feature table is comprised of points of variability for each feature. Where possible a discrete number is displayed. Other projects use verbal descriptions.

Table 23.3, Generic feature table points of variability						
1	**2**	**3**	**4**	**5**	**6**	**7**
Variability described as "characteristics" difficult to measure.	NA	yes/no	No table just a sentence, The Fiestal Network pattern can be generalized to allow for variable block size, key size, number of rounds, and the particular actions in the function f. Variability is hard to measure as the function f is not tightly defined	verbal description of each feature, not able to measure	4	3
8	**9**	**10**	**11**	**12**	**13**	
yes/no	yes/no	2 … 8	yes/no	yes/no	NA	

5.9. Domain Implementation

Phase two of domain engineering is domain implementation. The following tables consider the number of projects that had a domain implementation section and the types of reusable assets created.

5.9.1. Presence of domain implementation.

This table places a 1 where a reusable component was developed and a 0 where it was not.

Table 24.a ,Was it developed												
1	0	1	1	1	1	1	1	0	1	0	1	1

Ten of the thirteen, or 76% of the projects, contained a reusable asset of some kind**.**

5.9.1.1. Parts Based Reusable Asset(s)

This table places a 1 where a parts based reusable asset was developed and a 0 where it was not.

Table 24.b, Reusable Asset(s)												
1	NA	1	1	1	1	1	1	NA	1	NA	1	1

Ten of the thirteen project, or 69% of the projects, implemented a parts based reusable component.

5.9.1.2. Code Generator

This table places a 1 where a code generator was developed and a 0 where it was not.

Table 24.c, Code Generator												
0	NA	0	1	0	0	1	1	NA	1	NA	0	0

Four of the 10 projects that had a section from domain implementation created a domain specific code generator

5.9.1.3. Little Language

This table places a 1 where a little language was developed and a 0 where it was not.

Table 24.d, Domain Specific Formal Language (Little Language)												
1	NA	0	0	0	0	1	1	NA	0	NA	0	0

Three of the ten project or 33% developed a little language.

5.9.1.4. Language

This table contains the language of the projects reusable asset.

Table 24.e, Language												
java	NA	java	C	perl	.NET	java	perl	java	vb	NA	NA	java

5.10. Tables

Each DARE book contains glossary, synonym, thesaurus, bibliography and index tables. The following section considers the size of each of these tables and provides statistical summaries.

5.10.1. Glossary Word Count

Table 25.a, Glossary – Number of Elements												
20	117	10	30	0	9	0	0	0	4	0	5	0

Percent Contained a Glossary: 54%

Statistical Summary 25.a : Glossary Elements Count			
Sample Size, n:	7	Range	113
Mean	27.85	Minimum	4
Median	10	1st Quartile	5
Midrange	60.5	2nd Quartile	10
Variance	1629.81	3rd Quartile	30
Standard Deviation	40.37	Maximum	117

5.10.2. Synonym Word Count

Table 25.b, Synonym Table – Number of Elements												
11	0	3	0	3	13	0	4	0	4	0	0	8

Percent Contained a Synonym Table: 54%

Of those who had synonym tables:

Statistical Summary 15.b : Synonym Table			
Sample Size, n:	7	Range	10
Mean	6.5	Minimum	3
Median	4	1st Quartile	3
Midrange	8	2nd Quartile	4
Variance	16.95	3rd Quartile	11
Standard Deviation	4.11	Maximum	13

5.10.3. Thesaurus Word Count

Table 25.c, Thesaurus – Number of Elements												
12	0	12	0	0	14	0	22	0	0	2	0	8

Percent contained a thesaurus: 46%

Statistical Summary 25.c: Thesaurus Elements Count			
Sample Size, n:	6	Range	20
Mean	11.66	Minimum	2

Median	12	1st Quartile	8
Midrange	12	2nd Quartile	12
Variance	43.86	3rd Quartile	22
Standard Deviation	6.62	Maximum	22

5.10.4. Bibliography Element Count

Table 25.d, Bibliography – Number of Elements												
11	0	7	7	0	6	6	20		3	9	4	0

Percent Contained a Bibliography: 69%

Statistical Summary 25.d: Bibliography Elements Count			
Sample Size, n:	9	Range	17
Mean	8.11	Minimum	3
Median	7	1st Quartile	6
Midrange	11.5	2nd Quartile	7
Variance	25.61	3rd Quartile	8
Standard Deviation	5.06	Maximum	20

5.10.5. Index Word Count

Table 25.r, Index entry count												
11	0	37	0	0	55	35	6	0	0	0	0	0

Eight of the thirteen projects or 61% had an index in the book.

6. Failure Point Observations

The following discussion highlights failure points observed in the domain engineering projects. These comments are based on the data of section 5, the surveys, and participant demographic information. Individual project examples are given throughout to provide a broader understanding of the challenges faced by the domain analysts.

6.1. Variation

As noted in section 3 all participants of this study undertook domain engineering and the creation of a DARE book for the first time. With this consideration, there was understandably a good deal of variety of approaches to the project. Quantitative variations include the size of the DARE book and the time spent on each section of the process.

6.1.1. Book Size

Measuring the size of the DARE book proved difficult, as there was no consensus as to whether the source code and/or data of the vocabulary analysis phase was expected to be in the book. The two extremes were the all inclusive DARE book, where all data relative to the project went in the book, and the outline form, where all of the book's sections contained references to artifacts and data in outside locations. In one case, a DARE book measuring 422 pages was reduced to 47 after removing the source code and the vocabulary analysis of the exemplar system. The book was further reduced to 20 pages after removing a source code analysis section that listed all the exemplar systems classes and functions. This does not necessarily have any implications for the overall success of the project, but in terms of establishing a reusable methodology, the variation of size makes comparable measurements of size difficult. Comparing and contrasting book size as an indicator of success would be an appropriate area for further study.

6.1.2. Activity Log

There was much variety of interpretation of recorded activities and what titles should be given to those activities. One individual recorded his/her time in days rather than hours. Two individuals omitted time recording all together. The number of elements entered varied from 3 to 19, with a median of 8.5.

Finding general themes is difficult if one looks only at individual time recordings.
It is valuable, however, to look at the total time spent on the DARE project. Grouping subsections together also proved helpful. These include:

1. Domain Preparation: selecting a domain, gathering sources, and scoping.
2. Vocabulary analysis: gathering all domain sources to construct a general vocabulary, conducting analysis to derive a key word set, deriving cluster tables, facet tables, and template.
3. Domain implementation: creating a reusable software component or generator.

6.1.3. Omission of Book Sections

As highlighted in the various tables of section 5, there was a considerable variety to the completeness of each DARE book.

6.2. Time invested in the project

Those individuals who spent less than 70 hours on their projects achieved less with their projects. Those that spent less than 70 hours had either poorly scoped domains or their domains were too small. Those who spent less than 5 hours creating reusable assets did not create any truly useful product. In addition, those who spent five hours or less on vocabulary analysis either did not use any automatic tools and did not select vocabulary based upon system knowledge, or they only used automated tools; therefore, they did not create a set that was useful.

Total time spent on DARE project										
13	49	53	65	65.25	70	75	122	126	176	188

- Little time invested time in vocabulary analysis

Total time spent on vocabulary analysis	
<=5	3,3,4,5
>=12	12, 20,21,36

- Little time invested less time developing reusable assets

Time spent implementing Reusable Assets	
<= 5	5,4,5,5,3,0.5
>=50	55, 55, 67,80

6.3. Domain Preparation

Those projects that had difficulty scoping the domain, creating feature tables, and deriving a template through vocabulary analysis tended to choose domains that were either too big or too small. Many of those projects failed to clearly scope the domain, which resulted in a “vague” domain.

6.3.1. Domain Choice and Scope

- Domain is too small

Small domains have few facets and points of variability. This leads to an inability to construct generic architectures with adequate features, resulting in trivially small reusable components.

One project focused on a large software system of a chosen domain. Actually in one large program, systems exemplars were modules of the application. Domain scope was therefore artificial. This small domain choice constricted the exercises processes for the rest of the project. System architectures, feature tables, and facet tables all had very little variability. The result was a project with a generic architecture with very little utility.

- Domain is too big

A domain that is too big will have too many facets and points of variability. There will also be many more possible exemplar systems to include in the analysis. Large domains may also be very complex as well, making the analysis phases more difficult.

Subjectively, none of the projects sampled chose domains that were too big to be scoped. Some projects, however, did have difficulty scoping the domain to a manageable level.

One project, refined the domain’s scope iteratively, during the analysis phase. Beginning with the domain, “Symmetric Encryption Algorithms”, the author considered the complexity of the subject along with the set of more than 30 algorithms, and refined the domain to “Fiestal Block” symmetric algorithms. This scope was later further restricted by discarding consideration of the cryptographic strength of the algorithm, and focusing on capturing the commonalities and variabilities of the Fiestal Block architecture.

- Domain is not scoped well.

If a domain is not clearly scoped, it can affect the rest of the domain analysis process.
Four of the thirteen projects had no domain scope statement. Of the Nine that did have domain sections, only 3 or 33% used set notation to describe their domains. Others relied on verbal descriptions. One used verbal descriptions and pseudo code.

Those projects that had no domain scope section tended to be less successful in deriving facets and tables in the vocabulary analysis. Their reusable assets also tended to be smaller or non existent.

One project chose the domain of web based logging software or the "Blog Domain". Although there were exemplar systems to analyze, the project contained no domain scope section. Later in the system analysis section the author commented, the domain is not very complex but rather young and undiscovered. Since the domain is young, a lot of the components developed by applications have not been completed and released which will limit the understanding of the domain…" Whether the result of poor domain scope, or a domain that is too new to capture; the end result is a domain that is difficult analyze.

6.3.2. Exemplar Choices

6.3.2.1. Finding Exemplars

One problem observed by several subjects was that they were unable to locate exemplar systems. Reasons for this included trouble searching the web and lack of domain familiarity.

One individual said, "It took me a long time to decide on which systems to analyze because most of the open source systems I found on Internet do not have proper design and architecture information. After performing me research for several weeks, I went to Dr. Frakes for help with picking the systems, and he pointed out the ccount system, which has proper documentations and architecture design description. I picked the other two systems based on their implementation language, C. I have spent about 35-40 s in researching and reading over the materials and debating on whether or not to choose the systems for domain analysis."

This individual recorded 38 hours gathering exemplar sources. With a total project time of 126 hours, this means this project invested 30% of its time finding exemplar sources.

Class surveys reveal that this is a recurring theme in the work place as well. In response to the question, "What problems have you had when trying to reuse software?" Responses are as follows:

- "Haven't established a well organized reuse library. Hard to find codes to reuse, need experienced programmer to point out the existence of similar code."
- "Finding it""
- "I have not reused software before."
- "Not knowing where to find reusable components"
- "No pre-existing components"

- Poor Exemplars

Domain analysis is based on the extraction of information from previously built systems. If these exemplars are inadequate, then the resultant domain models and implementations will be poor.

One of the projects in the software metrics domain was based on three open source systems found at Sourceforge. The domain analyst reported that these systems were not good exemplars because they were poorly coded and had no maintenance support. The comments found in the following project notations reveal much about the quality of these projects, and hold good lessons for future analysis.

- Quality of project

The author of one project "…explicitly disclaims future support saying that (the project) was merely an artifact or byproduct of his academic research. "

Another stated the project "started as a VC++ implementation produced in my spare time over a 2 week period …". Incidentally, this project was removed from SourceForge sometime during the 14 week period after it was downloaded for the project.

- Age of Project

With regard to selecting these sources, the domain analyst reported: "Three software metrics projects were selected from SourceForge http://sourceforge.net/ as candidates. Selection criteria included their self-declaration as a software metric application, relatively recent age (none are over five years old)."

- Experience of Developer

Each of the System Descriptions of the SourceForge exemplars listed 5 years as the years of experience of the authors. One exemplar stated, "It was developed "… to scratch his own itch and learn a bit of Java at the same time."

6.3.2.2. Finding Analysis Tools

Some analysts expressed difficulty finding analysis tools to use in the project. This problem falls in the same category as finding exemplars. It is, however, a particular problem to the analysis stages of the DARE process – vocabulary analysis, architectural analysis, and software analysis.

One project in particular reported spending 50 hours looking for "good tools". Considering the total project time was 193 hours, 26% of time was spent searching for tools. Another 65.25 hour project spent 16.5 hours looking for tools, or 25% of the project time.

6.4. Source Analysis

Of the 13 projects, 6 or 46% had code analysis sections in their project.
Recorded hourly times of software analysis were 1, 2,2,3,3,4,30, and "5 days."

6.4.1. No Analysis

One project did not do analysis because of the poor exemplar quality. "Due to the differing intents of the authors, their differing Java skill levels, and their differing use of program and GUI generators, we find it difficult to perform a direct code analysis of the three programs and wouldn't use any of them in a real-world application".

Another stated he/she did not code analysis because they could not find the source code for two of their exemplars.

6.4.2. Analysis Problems

Of those who performed some degree of software analysis, problems came from several sources.

Lack of good tools for software analysis caused some individuals problems. One project indicated that they did not analyze the all the sources, because they used an application trial version that limited their use.

Yet another did not find satisfactory analysis tools, and concluded the exercise was fruitless. As recorded in the DARE book:

" Both the Vanilla and Align CMU tools were written in C/C++. The Moore tool was written in Perl. For Vanilla and Align CMU, I attempted to use the suggested cia and cflow tools. Cia seemed to be decommissioned (I could not find a functioning copy). There was a subsequent release called Acacia, but I also couldn't find a copy of this. Therefore, the analysis used only the cflow tool. After many fruitless searches, gave up on analyzing Perl. Apparently, there are no static code analysis tools for Perl. The main Windows implementation of Perl, ActiveState Perl, did not have a static analysis tool, though it had such tools for other languages. In any case, Moore's implementation does not uses subroutines or classes, so the static analysis would've been useless anyways."
"I do not believe this phase of the Domain Analysis gave me a deeper understanding of the domain systems commonalities and variabilities. This may be largely due to the fact that the systems are simple from a structural point of view (call and return) and they are small size (~ 1 KLOC)."

One subject did not use any tools, but manually inspected the source code, looking at the architecture similarity, code style and quality, location of modules, reusability potential.

Of the five individuals using software analysis tools,

- Manually inspected producing a description in pseudo code, used a tool to create a class diagram, used an editor to create a flow activity diagram, collection of static metrics
- Used a tool to list the call tree, as well as static metrics.
- Used a tool to generate call graphs for selected modules.
- Listed all the system functions

Source code analysis was, in summary, one of the weaker stages of the DARE process. Possible reasons for this and solutions are left to section 7.

6.5. Vocabulary Analysis

Vocabulary analysis encompasses the entire process of analyzing domain source vocabularies to select a key word set. Disciplines in this stage include various lexical analysis techniques such as conflation, stemming, and frequency analysis as well as creating word clusters to produce a facet table and template.

6.5.1. Deriving the key word set

Vocabulary analysis strategies varied across the projects.

- Five analysts selected their key word set manually, based upon their domain knowledge. (Projects 1,3,7,9,13)
- Six analysts used frequency analysis and stop lists to derive large sets, and then used their domain knowledge to select key words. These individuals derived sets of 1000, 2045, 2268, 4905,and 4816 words respectively, before choosing to use domain knowledge to select the key words. (projects 2, 4, 5, 8, 11) One analysis generated three system vocabulary frequency analysis tables of 398, 920, and 1314 before using domain knowledge. (project 12)
- One subject used frequency analysis, and then decided the automated process was inconclusive, opted to study the graphic interfaces for each application to derive important function names. (project 10)
- One subject incorporated frequency analysis and clusters with stemming, but felt that the automated tools were "overkill". *"This domain analysis phase yielded lots of fine-grained information. This is useful if an individual is unfamiliar with terminology from their domain, but this wasn't the case for me and my domain. I don't think it was helpful in finding commonalities/variabilities, though this step may have been more helpful if the process was more tightly bound with the rest of the domain analysis process (e.g. with the DARE tool). Overall, I felt that there was too much manual work for little payoff with the vocabulary analysis*." (project 6)

In summary in the sampled projects, there is much room for improvement towards teaching analysts to properly use lexical techniques.

6.5.2. Cluster Tables

There were three projects that incorporated cluster tables in their analysis. Of these, the exercise appears to be relevant, and contribute to the creation of a facet table.

- Generated sets of 5, 10, 15 and 20 with and without stemming. Then generated an “Agglomerative Cluster”. All this was done using a tool called CLUTO.
- Eight clusters ranging from 2 to 6 word
- Six cluster ranging from 6 to 12 words.

These three successes are in contrast to the remaining 77% of the projects that did not include cluster tables in their book.

6.5.3. Facet Tables

Regardless of methodology, all 13 projects derived facet tables. Within this set, there are variations of validity and quality.

One project produced one facet table for each of its three exemplars, failing to combine the systems into a single facet table. This represents a misunderstanding of the purpose of a facet table.

Another project whose domain was one single large program did not derive a facet table, but rather reproduced the generic feature table. In this case a facet table may have not been possible due to the constricting domain.

Facet Count	7	8	3	4	3	7	10	6	6	6	4	NA	7	8

6.5.4. Templates

Eleven of the thirteen projects, or 85%, had templates. The table below illustrates the number of words for each template and the number of variables.

Word Count	52	NA	66	95	29	52	104	32	55	88	15	NA	83	52
Variables	7	NA	12	12	3	8	10	6	6	20	3	NA	16	7

Two templates that have three variables might be considered too small. Two templates with 16 and 20 points of variability might be too complex.

What is valuable in terms of observing points of failure is to compare the number of facets with the number of variables in the templates. Comparing the two tables one can see that four of the projects more than doubled their number of variables between the facet table and template. One increased the variable count by one, and another decreased the variable count by one.

_	7	8	3	4	3	7	10	6	6	6	4	NA	7
_	7	NA	12	12	3	8	10	6	6	20	3	NA	16

-

-

This reveals that in the minds of the domain analysts, there is not necessarily a one to one mapping of the facet table to the template, though there should be.

6.6. System Analysis

6.6.1. System Feature Tables

There was little consensus as to how to represent system and generic feature tables of a domain. As seen in the tables below, although all participants had three or greater exemplars, 7 of the 13 had one system feature table and four of the 13 did not create a generic feature table.

Number of exemplars	3	7	3	3	3	4	3	8	4	3	3	3	3
Number of system tables	1	1	3	3	3	0	3	6	1	1	1	3	1
Generic Feature Table	1	0	1	0	0	1	1	1	1	1	1	1	0

The descriptive table below captures the different groups as they created system and generic feature tables.

Analyst	**System Feature Tables**	**Generic Feature Tables**
3,8,12	Three tables, each of a different system	One table with systems as columns.
5,7	Three tables, each of a different system	One table, features and variability. (no system names)
4	Three tables, each of a different system	None. A descriptive sentence.
10	One table with systems as columns.	One table, features and variability (no system names)
2,9,13	One table with systems as columns.	None
11	One table with systems as columns.	One table with systems as columns.
1	One table, features and specification. (no system names)	One table, features and specification. (no system names.
6	None. Referenced the generic feature table	One table with systems as columns.

With the exception of group (3,8,12), there was in general a misunderstanding that 1) system tables are distinct tables, each representing a system of the domain, and 2) generic feature tables combine these systems in to one table, but contain column headers of each system. To this end, group (5, 7) were correct on system feature tables, but reduced the generic feature table to a single table without system names. Group 10 and 2,9,13 produced a correct system table, but labeled them as system feature tables, leaving them with nothing for the generic feature table.

6.7. Architectural Analysis

System Architecture Count	9	14	3	26	4	3	3	6	0	3	5	6	3	9
Generic Architecture count	1	4	1	2	1	1	1	1	2	1	2	2	1	1

Architectural analysis, both at the system architectural level, and the generic level had representation in all projects. Subjectively this seems to be one of the more successful aspect of the DARE book. There may be some points of failure in terms of architectural validity or failures to map architectures to implementation, but this will be left to a late study.

6.8. Supplementary Material

Although the majority of the projects contained a glossary, synonym table, thesaurus, and index, these elements were most likely an afterthought, rather than an important part of the domain analysis. Reviewing the statistical summaries of these elements it is evident that as a group their priority was low. Only project 1 and project 6 included all the tables in their project.

Glossary – Number of Elements												
20	117	10	30	0	9	0	0	0	4	0	5	0

Percent Contained a Glossary: 54%
Average number of elements of those who had glossaries: 27.8
Minimum: 4
Maximum: 117
Median: 10
Range 113

Synonym Table – Number of Elements											
11	0	3	0	3	13	0	4	0	4	0	0	8

Percent Contained a Synonym Table: 54%
Of those who had synonym tables:
Average: 6.14
Median: 4
Minimum: 3
Maximum: 11
Range 10

Thesaurus – Number of Elements												
12	0	12	0	0	14	0	22	0	0	2	0	8

Percent contained a thesaurus: 46%
Average: 10.14
Median: 12
Minimum: 2
Maximum: 22
Range 20

Index – Number of Elements												
11	0	37	0	0	55	35	6	0	0	0	0	0

Percent contained an index: 30%
Average: 10
Median: 11
Minimum: 2
Maximum: 22

Bibliography – Number of Elements												
11	0	7	7	0	6	6	20		3	9	4	0

Percent Contained a Bibliography: 69%
Average: 28.8
Median: 35
Minimum: 6
Maximum: 55
Range: 49

6.9. Domain Implementation

Ten of the thirteen projects implemented some sort of reusable asset.

Of these ten, all created a reusable component. Four of these ten implemented a code generator and three created some aspect of a little language.

Why specific projects failed to create reusable assets is uncertain and requires further study..

6.9.1 Those with no programming experience, but otherwise excellent analyses, failed to produce working code.

Conflation (1)

- Had well established exemplars and documentation, Sought expert advice - contacted the author of one project for clarification, used paragraph explanations for system architectures, domain scope consisted of verbal descriptions and set notation,
- Code analysis consisted of verbally describing each step of the algorithm, class diagrams, and activity diagrams, and static metrics,
- Architectural analysis consisted of pseudo code for each algorithm, top-down functional diagrams, and a use case diagram
- Implementation consisted of three parts based functions designed to performed simple string manipulations. This was not working code, but represented a best effort from a non-programmer with a good grasp of the domain.
- The little language as well was not operational, but rather was a BNF representation of the syntactical elements of a little language.

Conflation (11**)**

- Well established exemplars and documentation.
- Complete system architectures

- Combined system feature tables into one table, listing systems in columns.
- Domain scope consisted of word descriptions and set notations
- Implemented word clusters
- Correct Facet table
- Correct template
- No code analysis
- Two good generic architectures - a top down functional architectural, and decision flow.
- Correct generic feature table.
- Implementation consisted of eight conflation functions described in pseudo code.

6.9.2 Those who had great difficulty in analysis either had no code, or created relatively simple functions

AHLTA Longitudinal Domain (2)

- Chose a domain that was actually a large program,
- Systems were program modules,
- Had no domain scope,
- Confused the facet table with a system table,
- Had no template …
- No implementation.

Simple Metrics (5)

- Fifteen years experience,
- Chose exemplars of poor quality produced by authors with less than 5 years experience,
- Failed to do any code analysis,
- Had generic feature table with only 3 features,
- Architectural analysis was lacking as the exemplars had no architectural diagrams.
- Generated a reusable asset that counted lines of code and comments.

Blog Domain (9)

- Picked a new domain with little standardization
- Immature exemplars,
- Chose vocabulary manually,
- No code analysis (had five different languages in the domain),
- No system architecture,
- Minimal generic architecture,
- Implementation consisted of an abstract data type high level description.

Static Code Metrics (12)

- One year of programming experience,
- Spent 38 hours searching for exemplar systems Turned to system expert for help.
- Had three exemplars and three sets of documents.
- Exemplars did not have architectures, so studied algorithms and created architectures manually. Produced a top-down functional architecture and data flow diagrams
- Misunderstood facet tables and created three facet tables.
- Misunderstood templates and created three templates
- Created a reusable component

7. Failure Points Summary

Looking at the previous collection of specific failure points, it is possible to make few general statements about the categories of failure.

7.1. A new process leading to misunderstood concepts

For all the participants, this was their first exercise in both domain engineering and creating a DARE book. As such, subjects were both learning the domain of domain engineering and their chosen domain. Many of the concepts, key to domain analysis, were new and were prone to misunderstanding.

7.2. Shortcomings of skill set

Irrespective of an analyst's experience with the DARE methodology, domain engineering is a multi-disciplinary process, and any one individual may not have all the requisite skills necessary to complete all its tasks. Some individuals with MIS backgrounds had a very thorough analysis phase and then came up short on implementation. Likewise, some CS individuals who might rather be coding shortchanged many of the analysis steps, and produced relatively well developed reusable assets.

7.3. Interdependent Stages

The following failure mode diagram captures many of the failure points discussed. This fish-bone diagram illustrates how failings upstream of the process affected subsequent stages.

8. Implications for Success

Having taken a failure modes approach to the DARE exercises, highlighting principles of success observed in the data will provide a more complete picture. Suggestions for improvement in section 9 are based on both failures and successes.

8.1. Success is a function of Time

We observed that those who spent less than 70 hours on their project had weaker projects than those who spent greater than 120 hours all found success.

Total time spent on DARE project	
<=75	13,49, 53, 65, 65.25, 75
> 120	122, 126, 176, 188

Domain Preparation

Considering the set of operations from selecting domain sources and scoping the domain, we find a similar pattern. Those who spent more time found success, both in this first stage of the project as well as the overall project.

Preparing the domain (choosing, finding exemplars, scoping)	
< 20	7, 9, 14, 15, 18.25
> 30	33, 50, 52, 74

Vocabulary Analysis

Vocabulary analysis is a large portion of a project. The journey from documentation and software vocabulary to a template involves many skills and as such was the section with the most errors. This being said, those with stronger overall projects, also invested more time in vocabulary analysis.

Vocabulary Analysis	
< 10	3,3,4,5
> 20	20, 21, 29, 36

Reusable Asset

It has already been observed that there were some individuals without programming experience. Excluding these from consideration, and comparing the reusable assets of those who reported programming skills, more substantial reusable assets were developed by those who spent more than 20 hours.

Time spent implementing Reusable Assets		
<= 5	3, 4, 5,	
>=24	24, 55, 67, 70,80	

8.3 Success is a function of persistence

As said before there was a great variety in the time recording process. It is assumed that if someone recorded a time that it represents a significant investment of time on the project.
Viewing the outliers, therefore, adds more weight to their importance. Outliers may indicate areas where individuals struggled through a new area, or in the case of programmers, it may indicate predisposition to programming. What is significant is that the projects that contain time outliers were among the set of projects that were more successful. The clear message is that success comes to those who persist through difficult areas.

Time outliers					
Finding good tools		50			
Gathering Source Information	38				
Study of domain			46		
Architectural Analysis		60			
Vocabulary Analysis				30	
Architectural Analysis					
Development of reusable asset		55	80	67	
Total time preparing domain					74
Total DARE Time	126	188	176	122	126

8.2. Success is a function of knowledge

Domain expertise, coupled with an understanding of the various components of analysis, was a clear indicator of project success.

One of the strongest projects of the group came from an individual who did not score highly on any of the outlier tests, and was only within the 2nd quartile of the total project time. This 75 hour project spent 14 hours preparing the domain, 29 hours of vocabulary analysis, and 24 hours developing a reusable asset. This project's success factor was going into the project with knowledge of the domain, strong analysis tools, and a good grasp of architectural analysis.

8.2.1. Those with a great deal of programming experience created reusable asset, regardless of the quality of their analysis.

Sentence Alignment Systems (6)

- Combined system feature tables into one table
- Found code analysis fruitless and did not help to better understand and domain,
- manually picked out domain vocabulary,
- Created functioning reusable component program

Personal Information Managers (10)

- Chose small manageable domain,
- Chose toy project exemplars,
- Manually selected vocabulary based upon the graphical interface.
- Code analysis consisted of listing all functions of the exemplars.
- Created a project generator and little language.

Open Source Java Metrics (3)

- domain scope contained no sets,

- manual code analysis,
- simplistic architecture,
- Produced Master's Thesis level code reuse architecture.

8.2.2. Those with strong domain knowledge and programming experience excelled.

Conflation (7)

- Described the domain in pseudo code,
- Template contained five templates capturing the five different facet tables,
- Extracted vocabulary manually,
- Very strong architectural analysis …
- Produced functioning reusable component program and a little language.

Conflation (8)

- Thirteen years experience in C and C++,
- Strong analysis,
- Implementation: eight reusable parts based functions, application generator and little language

Symmetric Encryption (4)

- Three years experience in C++
- Manually picked out vocabulary,
- No generic system table,
- No thesaurus, synonym table, no cluster tables,
- Template had twice as many variables as the facet table …
- Created reusable component program and code generator.

8.2.3. Success is an iterative process

One project took 46 hours of the 176 project studying the domain. Considering that it invested 80 hours developing the reusable asset, this means they spend 48% of there analysis studying the domain. This project, first based upon symmetric encryption, revised its scope and subsequent facet tables, templates, and generic architecture three times. As the domain and software exemplars were explored, the scope was narrowed to a generic encryption algorithm using the "Fiestal Network".

9. Recommendations

Running throughout all the examples of this report is the central message that Domain Engineering and the DARE methodology is must be understood by the analyst before proceeding. Regardless of the analyst's chosen domain, the DARE book will only be as good as the knowledge of the DARE method that the analyst has. Future DARE efforts, therefore, will be more successful if they begin with a preliminary skill assessment, so that the analyst can address those areas where they are weak.

9.1. Essential Vocabulary

Domain engineering is multi-disciplinary and is based on a set of skills and concepts. It requires an understanding of such concepts as "system", "component", "induction" and "empirical". These are not always taught today's computer science and information science curricula. Appendix A contains necessary vocabulary for an understanding of domain engineering.

9.2. Essential Skills

The abilities to find source material and understand software architecture are examples of skills that are fundamental to successful domain engineering. The following table contains an outline of these skills. Future domain analysis exercises might begin by evaluating their skill set.

- Research techniques
    - Discovering appropriate domains
    - Finding source code
    - Finding exemplar documents and architectures
- Proficiency programming in one language
    - Compiling source material
    - Software development
- Mathematical
    - First order predicate calculus
    - Set notation
    - Discrete Mathematics
- Software Architecture Styles
        - Main Program/Subroutine
        - Batch-Sequential
        - Pipe and Filter
        - Layered
        - Client/Server
        - Object Oriented
        - UML
            - Class diagrams
            - Use Case
            - Sequence
            - Flow
- Software Analysis
    - Lines of Code
    - Systematic Complexity
    - Essential Complexity
    - Memory Use
    - Algorithmic analysis

- § Speed/ Efficiency
- Vocabulary Analysis
  - Stop List
  - Stemming
  - Conflation
  - Frequency Analysis

9.3. Domain Engineering Skills

The various tables and diagrams that are produced during Domain analysis must be clearly understood by the analysts.

- Set notation to describe the domain
- System feature tables
- System architectures
- Cluster table
- Facet Table
- Template
- Generic feature table
- Generic architecture

9.4. DARE Book Creation

Variations of the time recordings of the DARE book suggests that it might be beneficial to standardize future time logging practices. This may include more explicit direction.

9.4.1. Formalize the time recording requirements

Future DARE research will be improved if the time recordings are standardized. This will improve the correctness of the recordings and reduce the likelihood of individuals omitting to record some parts as well as standardize the vocabulary. This will serve several purposes. It will improve future research efforts such as this one. It will remind analysts of the mandatory steps of the process. Time recordings such as “learning the domain”, “searching for exemplar sources”, “reviewing vocabulary, essential skills, and domain engineering skills” will paint a more complete picture of the process.

9.4.2. Provide research information regarding time expectations.

It would be beneficial to an analyst, especially on their first project, to be provided some perspective as to the time and effort required for success. Data from this report may play an important part in that. The following table provides time investment summaries of successful projects.

Time of most successful projects						**Average**
Total DARE Time	75	122	126	176	188	137.4
Preparing the Domain	14	33	74	52	50	44.6
Vocabulary Analysis	29	3	20	12	20	16.8
Domain Implementation	24	67		80	55	45.2

- Successful projects of this group invested more than 70 hours on the project, with the majority investing over 120 hours.
- Successful projects spent on average 44.6 hours “preparing their domain”. These steps include the process from

domain selection to domain scope. Those projects who spent over 120 hours invested on average 52.25 hours preparing their domain.

Given this small sample, these times are by no means conclusive, but they do give a general sense of how much time someone should anticipate investing if the wish to be successful.

9.5. **DARE book project check list**

The fish-bone chart of section 7.3 can serve as the starting place for a DARE book check list. At each step of the process, selecting exemplars, scoping the domain, etc. the analyst can evaluate their book against the failure points provided.

10. Further Research

- More analysis

These projects contained many data points, and not all were explored. Areas of analysis not addressed in this report include 1) the quality of the exemplar sources, 2) quality of architectural diagrams and the subsequent architectural analysis, 3) evaluating the reusable assets both in terms of their quality as well as exploring if the reusable assets satisfy the models produced from domain analysis.

- More data

Although the data used in this study has provided a unique opportunity to study the strategies of individuals implementing the DARE methodology, the sample size of this report suggests that more examples are needed. This report's suggestions must be evaluated against repeated examples.

- Ways to improve

Survey data taken during and after the projects provides a wealth of information which can be further explored. (Appendix C) During this report's research a new survey was conducted, opening responses to the current set of project authors as well as a new class of first time domain engineers (Appendix B). This survey, focusing on vocabulary analysis, suggests these respondents had more success than observed in the sample set of this project. Gathering data from these individuals, and exploring reasons for this delta of improvement would provide the basis for another report.

11. References

[1] Frakes, W., R. Prieto-Diaz and C. Fox (1998) DARE: Domain Analysis and Reuse Environment. Annals of Software Engineering, v. 5, pp. 125 -141.

[2] Frakes, W.B., and Kang, K. "Software Reuse: Status and Future", IEEE TSE, 2005.

[3] Frakes, W. (2000, Nov. 6-9). "A Method for Bounding Domains". In IASTED International Conference Software Engineering and Applications 2000 (SEA 2000), Las Vegas, NV:

[4] Frakes, W.B., (Ed.): (2000) Software Reuse: Advances in Software Reusability, 6th International Conference, ICSR-6 Vienna, Austria, June 27-29, 2000 Proceedings ISBN: 3-540-67696-1

[5] Frakes, William and Sadahiro Isoda. "Success Factors of Systematic Reuse." Introduction to special issue on reuse IEEE Software (September 1994): V11, n5, pp. 14-19.

[6] Frakes, W. "Software Reuse Introduction and Basic Concepts", http://frakes.cs.vt.edu/SEportalReuse.htm

[7] D. Weiss and R. Lai, Software Product Line Engineering, Addison-Wesley, 1999.

[8] K. Kang, et al., Feature-Oriented Domain Analysis(FODA) Feasibility Study, tech. report CMU/SEI-90-TR-21, Software Eng. Inst., 1990.

[9] P. Naur and B. Randell, (Eds.). Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968, Brussels, Scientific Affairs Division, NATO (1969) 231pp. http://homepages.cs.ncl.ac.uk/brian.randell/NATO/

[10] D Garlan and M Shaw, An Introduction to Software Architecture.", Advances in Software Engineering and Knowledge Engineering, Volume 1 World Scientific Publishing Company, 1993.

[11] Biggerstaff T, "A Perspective of Generative Reuse," in Annals of Software Engineering, William Frakes (ed.), Vol. 5, 1998

[12] K Czarnecki ,U.W. Eisenecker ,"Generative Programming – Methods,Tools,and Applications, 11-12-2001, http://www.generative-programming.org

[13] Biggerstaff T. J.,Software Generators, 2006, http://www.softwaregenerators.com

[14] Generative Programming and Component Engineering: ACM SIGPLAN/SIGSOFT Conference, GPCE 2002,

Pittsburgh, PA, USA, October 6-8, 2002. Proceedings. Volume 2487/2002,
[15] J Clements, P Graunke, S Krishnamurthi, M Felleisen, “Little Languages and their Programming Environments”, Rice University, Brown University, May 2001
[16] Bentley, "Little languages", Communications of the ACM, 29(8):711-21, August 1986. Stroustrup, B. (1996)
[17] A. Deursen,P. Klint J. Visser, Domain-Specific Languages:An Annotated Bibliography CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
http://www.cwi.nl/ arie,paulk,jvisser/
[18] W Frakes, T Pole, An Empirical Study of Representational Methods for Reusable Software Components, IEEE Transactions on Software Engineering, Vol. 20, No 8, Aug. 1994
[19] Csound, Dr. R Boulanger, 11-12-2006,http://www.cSounds.com
[20] Make, Gnu Project and Free Software Foundation (FSF), 12-2006 http://www.gnu.org/software/make
[21] D. Clarke, T. Wrigstad, “A Fistful of Languages”, 12-2006,http://www.littlelanguages.net
[22] H Mili, F Mili, A Mili, Research Software: Issues and Research Directions, IEEE Transactions on Software Engineering, Vol. 21, No. 6, June 1995.
[23] C. McClure, “The three R s of software automation: re-engineering, re-pository, Reusability, ” Prentice-Hall, 1992.
[24] S.C. Cheung, and J. Kramer, “Enhancing compositional reachability G. Gruman, “Early reuse practice lives up to its promise,” IEEE Sop- ware, pp. 87-91, Nov. 1988.
[25] Frakes, W. (2000, Nov. 6-9). "A Method for Bounding Domains". In IASTED International Conference Software Engineering and Applications 2000 (SEA 2000), Las Vegas, NV:
[26] Basili, V., McGarry, F., Page, G., Pajerski, R., Waligora, S., Zelkowitz, M., "Software Process Improvement in the NASA Software Engineering Laboratory," Technical Report CMU/SEI-94-TR-22, Pittsburgh, Pennsylvania: Software Engineering Institute, Carnegie Mellon University, December 1994.
[27] Joos, R., "Software Reuse at Motorola," *IEEE Software*, September, 1994, pp. 42-47
[28] Lanergan, R.G., Grasso, C.A.; ”Software Engineering with Reusable Design and Code‘; IEEE Transactions on Software Engineering; Sept 1984; Vol.10 No.5 P498-501
[29] W. Frakes, C Fox, “Sixteen Questions about Software Reuse”, Communications of the ACM, June 1995, Vol. 38, No. 6.

12. **Appendix A Essential Vocabulary (Domain Engineering Key Word Set)**

Domain Engineering: The entire process of reusing domain knowledge in the production of new systems. Comprised of Domain Analysis and Domain Implementation, information of a class of similar systems are identified, captured, and organized with the purpose of systematic reuse. Product line engineering [1]

Domain Analysis: Process of identifying domains, bounding them, and discovering and documenting the commonalities and variabilities within related software systems. Leads to Domain Implementation [1]

Architecture: Set of components and the interactions between those components, and patterns that guide their composition and constraints on these patterns. [1]

Architecture Components: are such things as Client, Server, Databases, Filters, Pipes, and Layers in a hierarchy

Architecture Interactions: can be complex protocols like Client Server or just shared data.
Examples: Data flow or Pipe and Filter, Use case, Layered, Blackboard, Interface, Translation, Abstraction / Object Oriented /, Top Down, Data abstraction, object oriented, Client Server

Data: a collection of facts or data from which conclusions may be drawn; “statistical data” [syn: information, fact] Knowledge differs from data or information in that new knowledge may be created from existing knowledge using logical inference. If information is data plus meaning then knowledge is information plus processing.

Inference: Deriving new knowledge from knowledge or facts. For example: Tom is David’s father and David is John’s father. With these two points of information, then one can infer that Tom is John’s grandfather.

Inductive: arising from inductance; “inductive reactance” 3: of reasoning; proceeding from particular facts to a general conclusion; “inductive reasoning” Not scientifically provable, although statistically demonstrable. It is the basis of the scientific model. For example: 100 clowns sampled have red noses. Bob is a clown. You then can’t prove that Bob has a red nose, but it is statistically probable.

Deduction involves inferences from general principles. Can be used in logic proofs
For example: All Apples are Fruit
All fruits grow on trees.
Therefore all apples grow on trees.
Examples: Predicate Calculus, Propositional Logic
This is the same as knowledge

Index: Controlled, Classification (like this outline)
Entity: Something that exists as a particular and discrete unit.
Object: An entity that has attributes and may exhibit behaviors.
Element: A fundamental, irreducible constituent. A member of a set.
Constituent: One of a set of elements into which a construction or component may be divided by analysis. Part of a whole
Component: A constituent element, as of a system.
Module: A uniform structural component used repeatedly in building.
System: A group of interacting, interrelated, or interdependent elements forming a complex whole.
Stemming: attempt to reduce a word to its stem or root form
Conflation: To relate word forms. (engine, engineer, engineering)

Facet Table: Table listing the commonalities and variabilities of a set of systems. Column labels represent the commonalities and table rows the variabilities.
System Feature Table: Table listing the categories and features of a single system in a domain.
Generic Feature Table – Table highlighting the features of a set of system feature tables. Columns represent individual

systems, rows contain features of the systems.

Commonalities: Features in common among a set of systems in a given domain.
Variability – features differing in a set of systems in a domain.
Generic – representing a combination of systems.
Domain – an application area or a set of systems that share design decisions.[2]

Software Reuse: the reusable asset created as a product of domain implementation.
There are two types of reusable assets: 1) parts based – (from scratch, reengineered, purchased, free ware) 2) formal language – (little language or application generator)

13. **Appendix B : Recent Survey**

Which automated vocabulary analysis tools did you use?	
Frequency Analysis	(64%)
Stemming	(73%)
Conflation	(9%)
Cluster Tools	(27%)
Lex	(18%)
None	(0%)

Using domain knowledge to manually pick out words is more useful than using automated vocabulary tools	
Strongly agree	(9%)
Agree	(45%)
Disagree	(27%)
Strongly disagree	(9%)
No opinion	(0%)
They are equally useful	(9%)

Vocabulary analysis was helpful in developing the reusable software asset.	
Strongly agree	(9%)
Agree	(55%)
Disagree	(27%)
Strongly disagree	(9%)
No opinion	(0%)

Which of the following best describes your experience finding automated tools for vocabulary analysis?	
Successful	(36%)
Somewhat successful	(45%)
Minimally successful	(9%)
Not successful	(9%)
Did not look	(0%)

In my vocabulary analysis I differentiated between nouns and verbs.	
True	(45%)
False	(55%)

At the time of the project the following terms were unclear.	
Facet Table	(55%)
Template	(9%)
Feature Table	(27%)
Cluster Table	(27%)
None	(27%)
Other	(0%)

In my project, my reusable asset was based on the template.	
True	(18%)
False	(18%)
Not Sure	(55%)
no answer	(9%)

Which of the DARE phases did you had trouble with?	
Finding Domain Source	18%
Selecting exemplars	9%
Scoping the domain	18%
Describing the domain using set notation	36%
Developing system feature tables	9%
Performing Architectural Analysis	18%
Vocabulary Analysis	27%
Developing a reusable asset	45%
Other	27%

What suggestions do you have to make improvements in future DARE efforts?
Emphasize the automated lexical analysis as an aid rather than an end in itself. It is far too facile to think that automated tools that simply look for words are sophisticated enough to obviate human thought and analysis of the material at the semantic level. Yet this was the emphasis in class. This method offers less than Abbott's original idea of identifying nouns and verbs in a prose description of system requirements (although he was not doing domain analysis for reuse, the idea is the same).
From this project experience, it's clear why subject matter experts are important. If I did this for a work project, I would make sure I had a couple SME sources.
A clear example would be very helpful to understand the concepts in the beginning of the process.
More concrete examples. Instructor review of each component in a section before proceeding to next section and well before project is due.
The vocab. analysis was extremely useful for developing the glossary. Glossary was an extremely valuable asset by itself. Vocab analysis did not yield any re-usuable components. That was derived from the arch. analysis. That was a very manual process.
I thought it was a good project and can not think of any improvements at this time.
Provide a unified DARE framework to aid in the process. Many of the early lectures/papers hinted at such a framework, which in the end, did not exist. Too much time was spent finding disparate and only marginally useful tools (just to fulfill the requirements of the DARE process ... most of which did not aid in the process of developing a generic architecture).

I found the DARE process to be effective in helping us to define our specification. I see it as one of those things that gets easier and more effective with practice. The versatility (and value) of the results became clearer to me at the end of the process, with respect to the breadth of development efforts it enabled for programmers, sw architects, engineers, managers, requirements analysis,..ect.

What general comments do you have about the DARE process?
This response does not reflect a full-blown DARE project, but a toy example for a class. So my answers may not fit exactly with what you are looking for. Under "Which of the DARE phases did you had [sic] trouble with?" we did not find domain sources, select exemplars, use set notation, nor develop a reusable asset as part of our project. Please feel free to contact me furhter at jseigle@vt.edu
It was a learning experience and it's something you need to do multiple times before you get accurate and consistent results. It's not something you do one time and have it mastered. As far as an automated means for performing domain analysis the process worked well, but it would be nice to have an integrated tool to assemble the book. Using COTS is doable, but you can't count on everyone having all the software packages on their system. For instance, when I tried to open the ins files in teh DARE example, my system thought they were some type on configuration file and not Inspiration files.
I think result was very useful.
Process was informative and I learned something.
Very interesting. I would use it again.
The most useful part of the process for me was having to document system architectures. This helped me to discover commonalities between the systems which were not apparent before.
I believe there should be a stronger emphasis on inclusion of domain modeling techniques for the generic architecture and subsystems. I see these as invaluable keys to communicating a detailed specification of the domain. Incorporating the DARE book itself in a defined, consistent and flexible format (like OneNote) might help promote understanding, would encourage commonality in communication, and ultimately reduce analysis time.

14. **Appendix C: Survey of present sample.**

1.

Using the following scale, for each question below, please circle the number that best reflects your opinion about reuse activities in your organization.		
0 Not Applicable 1 Never 2 3 Sometimes 4 5 Always	Responses	Average
We design our software for reuse	1,3,3,3,3,3,4	2.86
We get parts from a reuse library	1,1,3,3,3,4,4	2.71
We have a major subsystem supplied to by other organizations	1,2,3,4,4,5,5	3.43
We informally reuse pieces of previous systems in new systems.	3,4,4,4,4,5,5	4.14
We practice reuse by reusing or adapting parts we've previously created	2,3,3,4,4,4,4	3.43
Reuse is practiced by individuals who adapt their own parts	3,3,3,4,4,4,5	3.71
We practice software reuse.	2,3,3,3,3,4,4	3.14

2.

What percent of the lifecycle objects your organization creates are typically composed of reused parts.	**Responses**	**Average**
Requirements	0,15,35,45,45	24.17
Designs	15,15,15,25,25,55	25.00
Code	5,15,25,45,45,75	35.00
Test Plans	15,65,0,0,75,75	39.29
Test Cases	0,0,15,45,75,75	36.43
User Documentation	5,15,25,45,65	24.29
Other Please Specify		
	Average All	**30.69**

3.

What percent of lifecycle objects you personally create are composed of reused parts?	**Responses**	**Average**
Requirements	35,45,0,5,45,15	18.33
Designs	25,55,25,15,15,15	25.83
Code	25,75,45,5,45,15	39.29
Test Plans	15,65,0,0,75,75	38.57
Test Cases	15,55,5,85,85,25	35.00
User Documentation	25,45,0,0,15,15	14.29
Other Please Specify		
	Average All	**28.55**

4.

What percentage of the parts your reuse are from external sources?									Average
Responses	0,	25	20-30	25	50	50	60	80	38.75% - 40%
Descriptive Statistics	Mean			38.75					
	Median			37.5					
	Midrange			40					
	Variance,			662.5					
	St Dev			25.74					
	Range			80					
	Minimum			0					
	1st Quartile			22.5					
	2nd Quartile			37.5					
	3rd Quartile			55					
	Maximum			80					

5.

Using the following scale, for each question below circle the number that best reflects your opinion.	
0 Not Applicable 1 Disagree 2 3 Agree somewhat 4 5 Always	Average
I feel I know how to reuse software	2.866
Reuse is economically feasible in my organization	3.14
I believe reuse works	3.71
Software developed elsewhere meets our standards	3.00
Project time constraints allow time for reuse.	2.57
It's more fun to write my own software than to try to reuse	3.43
I'm inhibited by the possible legal problems.	1.71
I've had good experiences with the quality of reusable software	3.00
CASE tools have promoted reuse across projects in our organization	1.00
A common software development process has promoted reuse across projects in our organization.	1.57
Reuse of parts not designed for reuse will never be cost effective.	2.43
Reuse is a process of adapting or modifying existing software to meet new requirements.	3.00

6.

Using the scale below, please indicate the frequency of the following situations.		
0 Not Applicable 1 Never 2 3 Sometimes 4 5 Always		Average
The part I needed existed and was available		3.00
The parts were probably around somewhere, but I couldn't find them		2.71
There was a library to look in for the part.		2.29
I found the part, and understood it sufficiently to reuse it		3.14
I found and understood the part, and it was good enough to reuse		3.14
I couldn't integrate the part into my system without extensive modification		2.71
If you couldn't integrate it, the reason was	Language Incompatibilities	3.17
	Improper Form	3.67
	It was too slow	1.50
	It took too much memory	1.50
	Other (please specify)	

7.

Using the following scale, for type of reuse below, please write the number that best reflects your opinion.		
0 Never Used 1 Not Valuable 2 3 Somewhat Valuable 4 5 Very Valuable		Average
	Responses	
Booch	0,0,3,0,0,0,0,0	0.38%
4 GL	0,3,0,0,0,0,0,0	0.38%
Unix	4,3,2,4,3,4,0,3	2.88%
Grace	0,0,0,0,0,0,0,0	0%
X-widgets	4,0,0,0,4,0,0,0	1%
Document Templates	5,3,3,3,0,4,3,5	3.25%
FORTRAN Libraries	0,0,0,4,0,0,0,0	0.5%
Program templates	4,4,0,3,2,4,0,3	2.5%
Ada Math Libraries	0,0,0,0,4,0,0,0	0.5%
Cosmic	0,0,0,0,0,0,0,0	0%
Other (please specify)		

8.

What is your primary job?	
Programmer	3
Software Engineer	3
Systems Engineer	0
Manager	1

9.

How many years of software engineering experience have you had?	Average
0,1,3,5,7,9,13,15	5.63

10.

On how many different projects have you worked?	Average
0,2,5,8,8,10,10,11	6.00

11.

For how many different organizations have you worked?	Average
1,2,3,3,4,5,6	3.43

12.

What is your highest degree?	Average
High School	
BS	6
MS	1
Ph.D	
BA	1

In what area is your highest degree?	
CS/IS	5
EE	1
Math	
Other (please specify)	Education - Psychology

13.

I was educated about software reuse in school	Responses
Yes	3
No	4
Don’t Know	1

14.

My organization maintains one or more	Responses

reusable repositories	
Yes	2
No	4
Don't Know	1

15.

My organization rewards reuse as follows	Responses
Recognition	
Cash bonuses	
No rewards	

16.

My company has a reuse organization	Responses
Yes	1
No	5
Planning One	
Don't Know	1

17.

My organization has an education program about software reuse	Average
Yes	
No	6
Planning One	
Don't Know	1

18.

My organization has a program in place to measure level of reuse	Responses
Yes	
No	6
Planning One	
Don't Know	1

19.

My organization has a program in place to measure software quality.	Responses
Yes	3
No	2
Planning One	1
Don't Know	

20.

My organization has a program in place	Responses

to measure software productivity	
Yes	2
No	4
Planning One	
Don't Know	

21.

How many employees are in …	Responses					
Your company	400	3,000	3,500	125	50	10,000
Your division	80	40	200	10	5	200
Your project	21	3	1	3	1	5

22.

What is the primary business of your company?	Responses
Aerospace	
Telecommunications	1
Manufacturing	
Software	2
Other (please specify)	biotech, military, power

23.

Please rank order the languages that are used in your company (1 = most common).	Responses
Ada	
Pascal	1,2
C	2,3,3,3
Lisp	
C++	1,1,1,1,2,2
COBOL	
FORTRAN	3
Smalltalk	
PL-1	
Assembler	4
Jovial	
Other: Java	1,1
Other: PL/SQL	2
Other: Perl	3
Other: C	4

Other: Visual Basic	1

24.

What problems have you had when trying to reuse software?
Since we do not have a systematic reuse program, we sometimes have compatibility problems.
Difficult to understand the module. Performance issue
Haven't established a well organized reuse library. Hard to find codes to reuse, need experienced programmer to point out the existence of similar code.
Code bloat, "almost but not quite" syndrome, corner cases, not pre-existing components, written/prototyped in matlab
How to modify similar software
Finding it
I have not reused software before
Not knowing where to find reusable components

25.

What other comments about reuse do you have?
Reuse is good but also important as integration of reused modules.
Works in some situations, but not all.
Savings on testing and verification

26.

How would you increase reuse in your environment?
By having a systematic reuse program
Make reuse repository and use external reusable components.
Engage management in importance of reuse and benefit conversations and need to establish a team to coord. on research.
Easier search/discovery
Create a reuse library

DARE System Description

Domain:

System Name:

EXPERT INFORMATION

Expert Name:

Expert Position: (choose one)
_____ Engineer _____ Tester
_____ Manager _____ Other
_____ Systems Analyst

Years of Experience:

Domain Familiarity: (choose one)
_____ Novice _____ Expert
_____ Specialist _____ Other

SYSTEM CHARACTERISTICS

Implementation languages used to build this system:

_____ Ada _____ X _____ C _____ C++
_____ Cobol _____ Fortran _____ PL/I _____ Other

Hardware used to build this system:

_____ Mainframe _____ Minicomputer _____ Workstation
_____ PC _____ Mac _____ Other

Operating system used to build this system:

_____ DOS _____ Windows _____ Windows-95 _____ MacOS
_____ UNIX _____ VM _____ VMS _____ Other

System Overview:

Architectural Style:

Dataflow Systems

_____ Batch sequential
_____ Pipes and Filters

Call and Return Systems

_____ Main program and subroutine
_____ OO Systems
_____ Hierarchical layers

Independent Components

_____ Communicating processes
_____ Event systems

Virtual Machines

_____ Interpreters
_____ Rule-based systems

Repository Centered Systems

_____ Database
_____ Hypertext
_____ Blackboard

Other: (specify)

Users:

_____ Programmer _____ Manager _____ Engineer _____ Other
_____ Technical Support _____ Staff _____ End User

Tasking:

_____ Single Thread _____ Multi-Thread _____ Other

Analysis and Design

_____ SA/SD _____ SADT _____ JSP _____ Other
_____ JSD _____ OOA/OOD _____ Func. Decomp. _____ None

QA Methods:

_____ Requirements Reviews _____ Design Reviews
_____ Formal Inspections _____ Walkthroughs
_____ Unit Test _____ Integration Test
_____ System Test _____ Regression Test
_____ Coverage Analysis _____ Program Proofs
_____ Cleanroom _____ Statistical Process Control
_____ Process Audits _____ Other
_____ None

Change Strategy:

_____ Evolve Single System _____ Evolve Multiple Systems
_____ Develop New Systems _____ Port Uniform System
_____ Port Customized Systems _____ Other
_____ None

Expected Changes:
Support new languages (presently, only English-German and English-French have been tested)
