DARE: Domain analysis and reuse environment *

William Frakes a, Ruben Prieto-Diaz b and Christopher Fox c

a Software Engineering Guild, 400 Drew Ct., Sterling, VA 20165, USA
b Reuse Inc., 12365 Washington Brice Road, Fairfax, VA 22033, USA
c Software Engineering Guild, 221 Lanier, Rte 2, P.O. Box 1105, McGaheyville, VA 22840, USA

DARE (Domain Analysis and Reuse Environment) is a CASE tool that supports domain analysis – the activity of identifying and documenting the commonalities and variabilities in related software systems. DARE supports the capture of domain information from experts, documents, and code in a domain. Captured domain information is stored in a domain book that will typically contain a generic architecture for the domain and domain-specific reusable components.

1. Introduction

Software engineering has traditionally been concerned with the development of custom-built single systems, and software engineering tools and methods have been developed to support single system engineering. Modern software engineering practice, however, requires a shift from crafting systems one at a time to the use of formal engineering principles for creating system models and architectures. Systematic software reuse – the practice of pre-planned reuse [Frakes and Isoda 1994] – offers a means of achieving such a transition. This paper presents a method and CASE tool for helping achieve systematic reuse through domain analysis.

Systematic software reuse is based on the observation that quality and productivity can be significantly increased by shifting the focus of software engineering to a domain-centered view. The domain-centered view of software engineering recognizes that most organizations do not build completely new software applications. Rather they build variants of systems within at most a few problem domains, or business areas. The key to quality and productivity improvements lies in adopting a process for building multiple related systems in a problem domain through systematic reuse. This activity requires infrastructure support.

The process of creating an infrastructure to support systematic reuse is called domain engineering. Domain engineering is analogous to creating the assembly line required in a manufacturing plant; running the assembly line is analogous to producing software applications. Different application domains (product lines) require different tools (machines), models (molds), processes, and so forth, that make the assembly line

* This work was partly sponsored by ARPA/US Army Missile Command under SBIR contract number DAAH01-93-C-R302.

© J.C. Baltzer AG, Science Publishers
unique. Components of the final product have to be designed and made to fit ahead of time. In this new paradigm the software life cycle becomes a process of customizing the assembly line for a given set of requirements, and then running it to produce a software product.

Domain engineering has two phases: domain analysis and domain implementation. Domain analysis is the activity of identifying and documenting the commonalities and variabilities in related software systems in a domain, that is, systems in a domain that share common design decisions [Prieto-Diaz and Arango 1991]. Domain implementation is the use of information acquired in domain analysis to develop reusable assets, such as reusable components or application generators, for the domain, and the creation of a production process for systematically reusing assets to generate new products.

Domain analysis is a complex activity requiring the intelligence of a human analyst. Many methods for domain analysis have been proposed. These methods provide useful high level guidance to a domain analyst. The methods so far, however, have lacked a comprehensive prescriptive strategy for domain analysis, and have lacked tool support, though we have found that many aspects of the domain analysis process can be automated.

DARE (Domain Analysis and Reuse Environment) is a CASE tool that supports the domain analyst in carrying out a well-defined domain analysis method. DARE supports an analyst in extracting and recording domain information from documents and code, acquiring and recording domain knowledge from experts, analyzing domain knowledge, producing various domain models (representations of important aspects of a domain), and producing a repository of reusable assets for the domain. While the DARE method is partially based on the STARS Reuse Library Process Model [Prieto-Diaz 1991], we found that the STARS method, along with other methods such as the Software Engineering Institute's Feature Oriented Domain Analysis (FODA) [Kang et al. 1990], and Organon Motives' Organization Domain Modeling (ODM) [Simo et al. 1995], are insufficiently prescriptive for automation, and often incorrectly focus on the analysis of low-level domain objects. FODA, for example, provides very little guidance to the central activity of documenting generic architectures, and the STARS method places too much emphasis on the bottom-up analysis of domain text.

DARE provides a more highly prescriptive method focusing on the extraction of high level domain information from domain experts. Though DARE is based on its own domain analysis method, it can be used to support FODA and ODM as well.

2. The DARE approach

The primary goal when using DARE is the creation of a generic architecture that describes architectural elements, and their relationships, for a family of systems. Some elements and relationships will be found in most systems in the domain – these are called commonalities. Other elements and relationships, the variabilities, are found in
only some of the systems in the family. The commonalities and variabilities must be recognized, analyzed, and used to create a generic architecture with a structure based on commonalities, but able to accommodate variabilities.

2.1. The domain book

To support this complex activity, DARE uses a book metaphor [Arango et al. 1993]. The domain analyst views the analysis task as one of creating a domain book that captures domain information from domain experts, domain documents, and code from systems in the domain. DARE also supports recording analyses and design decisions, and ultimately a generic architecture for the domain. It is the source for production of a repository of reusable components.

The DARE domain book both organizes the large quantity of domain information typically acquired and produced during a domain analysis, and helps guide the analyst through the DARE domain analysis method. As shown in table 1, a DARE domain book includes:

- all domain sources;
- the results of vocabulary analysis;
- the results of architecture analysis, including code analysis;
- summary information, such as a glossary, a bibliography, a user index, and appendices.

Once completed, the domain book provides a detailed specification of the domain.

2.2. Using DARE

Suppose an organization decides to analyze one of its domains. It will select at least one domain analyst for this task. In addition, the organization will need to identify and make available:

- Domain experts who are able to provide descriptions of systems in the domain and architectures for those systems.
- Source code from systems in the domain that will be analyzed to discover architectural information about systems in the domain and provide reusable code components.
- Documentation for systems in the domain, such as user manuals, requirements documents, design documents, and so on.

Using DARE, the analyst extracts domain information from these code and textual sources and records it in a DARE domain book. The analyst also elicits information from experts about systems in the domain, along with any insights they might have about what is common and variable among these systems, guided by DARE input forms. The domain experts also record system architectures in the domain book. Once all this data is collected and recorded, the analyst runs DARE text analysis tools to
<table>
<thead>
<tr>
<th>Sections</th>
<th>Chapters</th>
<th>Entries</th>
<th>How created</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table of Contents</td>
<td>-</td>
<td>Headings</td>
<td>Automatic</td>
</tr>
<tr>
<td>Part 1:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Domain Sources</td>
<td>Source Documents</td>
<td>Document files</td>
<td>User input &amp; ordering</td>
</tr>
<tr>
<td></td>
<td>Source Code</td>
<td>Code files</td>
<td>User input &amp; ordering</td>
</tr>
<tr>
<td></td>
<td>System Descriptions</td>
<td>Structured text</td>
<td>Directed user input</td>
</tr>
<tr>
<td></td>
<td>System Architectures</td>
<td>Architecture specs</td>
<td>Graphical user input</td>
</tr>
<tr>
<td></td>
<td>System Feature Table</td>
<td>Feature table</td>
<td>Assisted user input</td>
</tr>
<tr>
<td></td>
<td>Source Notes</td>
<td>Notes</td>
<td>User input &amp; ordering</td>
</tr>
<tr>
<td>Part 2:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vocabulary Analysis</td>
<td>Basic Vocabulary</td>
<td>Keywords &amp; phrases</td>
<td>Assisted text analysis</td>
</tr>
<tr>
<td></td>
<td>Facets</td>
<td>Facet table &amp; editor</td>
<td>Assisted text analysis</td>
</tr>
<tr>
<td></td>
<td>Synonyms</td>
<td>Synonym table</td>
<td>Assisted user input</td>
</tr>
<tr>
<td></td>
<td>Templates</td>
<td>Template table</td>
<td>Assisted user input</td>
</tr>
<tr>
<td></td>
<td>Thesaurus</td>
<td>Terms &amp; definitions</td>
<td>Assisted user input</td>
</tr>
<tr>
<td></td>
<td>Vocabulary Notes</td>
<td>Notes</td>
<td>User input &amp; ordering</td>
</tr>
<tr>
<td>Part 3:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architecture Analysis</td>
<td>Generic Architecture</td>
<td>Architecture spec</td>
<td>Graphical user input</td>
</tr>
<tr>
<td></td>
<td>Generic Features</td>
<td>Feature table</td>
<td>Assisted user input</td>
</tr>
<tr>
<td></td>
<td>Code Structure</td>
<td>Hierarchies</td>
<td>Automatic code analysis</td>
</tr>
<tr>
<td></td>
<td>Architecture Notes</td>
<td>Notes</td>
<td>User input &amp; ordering</td>
</tr>
<tr>
<td>Glossary</td>
<td>-</td>
<td>Words &amp; definitions</td>
<td>User input, auto ordering</td>
</tr>
<tr>
<td>Bibliography</td>
<td>-</td>
<td>Citations</td>
<td>User input, auto ordering</td>
</tr>
<tr>
<td>User Index</td>
<td>-</td>
<td>Words &amp; locators</td>
<td>Automatic</td>
</tr>
<tr>
<td>Appendix</td>
<td>Analysis Parameters</td>
<td>Parameters &amp; values</td>
<td>Automatic</td>
</tr>
<tr>
<td></td>
<td>Activities Log</td>
<td>Event descriptions</td>
<td>Automatic</td>
</tr>
</tbody>
</table>

extract terms and phrases from domain documents, and then uses the DARE cluster editor and facet table tool to create facets (discussed further below). These facets provide low-level organizational information the analyst can use to check and refine the system architectures provided by the domain experts. Other low-level structural information is provided by automated analyses of code, such as the call tree structure of a system. Facets are also used to help generate a system feature table that captures summary information about commonalities and variabilities of systems across the domain.

The facet table, system feature table, system architectures, and code structures are used by the analyst creating a generic architecture for the domain, and a generic feature table characterizing the stable common aspects and the allowed variabilities of systems built using the generic architecture. Source documents characterizing aspects of systems, or software components implementing aspects of systems conforming to the generic architecture, are selected and placed in a repository of reusable assets, with indexing mechanisms based on the earlier textual and faceted analyses. At project completion then, the domain book represents a thorough study and analysis of the
domain, and the generic architecture, feature table, vocabulary, and the repository of reusable assets, provide the basis for a domain implementation.

2.3. The DARE user population

DARE will be useful to many kinds of software engineers and managers, as shown in figure 1. A domain book created using DARE becomes a central organizational resource because it provides information about an organization’s major lines of business (domains).

Requirements writers, for example, can use DARE to support writing requirements for new systems in the domain. Software architects can use the generic architecture developed with DARE to specify architectures for new systems in the domain. Programmers can use DARE to identify and retrieve reusable assets; new reusable assets can also be added to the repository as they are developed. Maintenance engineers can use DARE to assess the impact of proposed requirements changes. Domain engineers can use DARE domain models and information to guide them in developing reusable assets such as reusable components and application generators. Finally, domain analysts and domain experts can use DARE to retrieve already-developed domain models as a starting point for new domain analyses.
3. The DARE prototypes

DARE has been developed in a series of research prototypes. In this section we review these prototypes and use them to illustrate the major features and functions of DARE.

3.1. The first prototype

The first DARE prototype was developed in C on a UNIX workstation running X-Windows and Motif during 1994. This version of DARE was used to develop and elaborate the domain book metaphor, and to investigate machine-assisted graphical word and phrase extraction and clustering.

The concept of a domain book as the means to structure and represent the outputs of the domain analysis process is important because it solves two vexing problems shared by current domain analysis methods, namely:

- What should the output of domain analysis be?
- What are the criteria for analysis completion, that is, how does the domain analyst know when a domain analysis is complete?

Both issues are current research questions in the reuse community. The book metaphor helps answer both of these questions: the output of domain analysis is a domain book, and the process is complete when all sections of the domain book are written.

An added benefit of the book metaphor is the structure it brings to the whole domain analysis process. Users naturally write a domain book from front to back and the sections of the book are ordered according to the DARE domain analysis method. Part 1 of a domain book, Domain Sources, is the input part of DARE where the material used in domain analysis is entered. Parts 2 and 3, Vocabulary Analysis and Architecture Analysis, structure the outputs of the domain analysis process. These parts also correspond to the two main steps of the DARE domain analysis method:

- bottom-up analysis – validate the generic architecture and generic features through analysis of domain documents (text) and source code;
- top-down analysis – postulate a generic architecture and generic features based on expert knowledge and experience.

The domain book structure neatly organizes the outputs of these two activities. The remainder of the domain book, the Glossary, Bibliography, Index, and Appendix, is end-matter that provides reference material and a search mechanism for locating items in the domain book.

Figure 2 shows the DARE process model. It illustrates how DARE tools support book sections.

There are three domain sources: text, code, and domain experts. Input from these three sources is processed and placed in the Domain Sources section of the book. Section 2, Vocabulary Analysis, contains the results of extensive bottom-up analysis of the
textual domain sources. This processing includes lexical analysis to generate words and their occurrence frequencies, machine-assisted selection of important words and phrases in the domain, machine-assisted determination of conceptual clusters, creation of facets, and definition of term synonyms. The Architecture Analysis section contains the results of code analysis, and the products documenting the domain analyst's decisions about the commonalities and variabilities of the domain. In particular, the generic architecture provides a reference architecture for new systems in the domain, and the generic feature table documents the major commonalities and variabilities of new systems in the domain. The Glossary and the Bibliography are created and updated.
continuously during domain analysis. The Table of Contents, Index and Appendix are generated and maintained automatically by DARE.

Figure 3 shows the screens for the DARE Table of Contents from the first prototype. This example shows the Table of Contents for a domain book for the information retrieval system domain [Frakes and Baeza-Yates 1992]. Sections of the domain book are expanded or collapsed by clicking on the triangle buttons to the left of the section headings. The Domain Sources section, and its subsection System Descriptions, are both shown expanded. Five information retrieval systems have been described in this domain: Catalog, Pirex, Grep, SMART and New System 5. The ”Go” buttons on the right open the portion of DARE used to display or manipulate the corresponding entry. For example, pressing the Go button beside Pirex opens a form displaying the Pirex system description and allowing it to be changed.
The DARE cluster editor supports domain analysts in creating conceptual clusters of related words and phrases. Clusters of words and phrases help identify common domain entities such as functions and objects. Clustering is an excellent bottom-up technique for discovering abstract concepts not obvious from a top down decomposition. This tool is also helpful for deriving faceted classification schemes. Faceted classification schemes in library science are derived by clustering related concepts. The cluster editor gives the user a graphical way of analyzing and manipulating the similarities between different words and phrases within the domain to arrive at a set of candidate facets.

A cluster is a group of words and phrases sharing some degree of conceptual similarity. Clusters may be represented by a single central word or phrase, with radial words and phrases extending outward from it. The length of each radial line can indicate how near or how far the radial word or phrase is from the cluster's center. Figure 4 shows a simple example of a cluster.

The words and phrases in this cluster all relate to one category, that of user input methods. In this example, the term Keystroke was calculated as the closest term to the cluster's center, while the phrases Mouse Click and File Read were calculated to have similarities fairly close to the center. The more exotic input methods Voice and SCSI Input are shown to be less similar to the central cluster concept by their longer radial lines.

The DARE cluster editor provides automatic clustering mechanisms for creating initial clusters, along with manual mechanisms for adjusting them. Figure 5 shows the initial cluster editor screen. The cluster editor shows the active vocabulary list (words and phrases from textual sources selected by the analyst with assistance from DARE). These words may be clustered automatically using either of two agglomerative clustering algorithms (average-link clustering or seed-based clustering) [Rasmussen 1992], or the analyst may select words and phrases to move onto the editing surface. If automatic clustering is used, various clustering parameters can be adjusted and the words reclustered until a set of desirable initial clusters is achieved.
Figure 6 shows how the clustering surface looks after an automatic seed-based clustering operation on a set of active vocabulary terms.

Once a set of initial clusters has been formed manually or automatically, the cluster editor supports manual cluster refinement. The analyst can use mouse operations to add or delete words in a cluster, combine or split clusters, or alter the distance between a word and its cluster's center. The ultimate goal in all of these manipulations, however, is to form facets, which constitute the critical categories of words and phrases that fall within the domain. Once the user has decided that a particular cluster should become a facet, he or she can promote it and name it. Facets are maintained if the active vocabulary is reclastered, so an iterative process of automatic clustering, manual refinement and promotion of a cluster to a facet, and then reclustering, is supported.

3.2. The second prototype

We found that the C, UNIX, and X-Windows environment did not allow rapid enough development of DARE, so we sought an environment that would. We chose Visual Basic, and the second DARE prototype was developed in Visual Basic 3 on a PC running Windows 3.1 during 1995. We found that this environment cut development
time about tenfold. Especially useful was a commercial graphical editor VBX that we were able to tailor to our needs.

The book mechanism and the cluster editor from the first prototype were recreated, and several major subsystems were implemented, including:

- the domain expert information entry forms and system feature table;
- the system architecture editor;
- the facet table;
- the generic architecture editor and the generic feature table.

Figure 7 shows a domain expert entry form used to record an information retrieval system description from a domain expert.

It may be important for the analyst to know about the expert giving the system information, so in addition to the system name, information fields include the expert's name and information about the expert such as his or her position, experience, and domain familiarity. System information falls into two categories: system characteristics that are properties of the system as a product, and process characteristics that are properties of the development activities used to produce and maintain
DARE System Description Form

Domain: Information Retrieval (IR)

System Name: Catalog

EXPERT INFORMATION

Expert Name: Bill Frakes

Expert Position:  
- X Engineer  
- Manager  
- Other  
- Systems Analyst

Years of Experience: 15

Domain Familiarity:  
- 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  
- X Minicomputer  
- Workstation  
- PC  
- Mac  
- Other

Operating system used to build this system:

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

System Overview:

Catalog is a boolean retrieval system that allows a user to create, modify, and search text databases using boolean queries.

Architectural Style:

Dataflow Systems

- Batch sequential
- Pipes and Filters

Call and Return Systems

- X Main program and subroutine
- OO Systems
- Hierarchical layers

Independent Components

- Communicating processes
- Event systems

Figure 7. DARE system description form.
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:
Port to Windows and Macintosh.
<table>
<thead>
<tr>
<th>Feature</th>
<th>Catalog</th>
<th>PIREX</th>
<th>NewIR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stemmer</td>
<td>Porter</td>
<td>N</td>
<td>Paice</td>
</tr>
<tr>
<td>Wordcount</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Ranking</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Full Text</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Boolean Searching</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Figure 8. DARE system feature table screen.

the system. System characteristics include a system overview, architectural style, type of users, tasking, programming language, hardware, and operating systems. Process characteristics include the analysis/design method, the quality assurance method, the change strategy, and expected changes to the system. Change strategy information will be helpful in designing the generic architecture to minimize the rework that will be required to accommodate new systems in the domain.

The DARE system feature table, shown in figure 8, is used to record key commonalities and variabilities across domain systems as perceived by domain experts. Rows in the table correspond to important domain features, with values given for different systems in the domain. In the example, possible values for the feature stemmer are defined as either Porter, Paice or N (none). Porter and Paice are alternative algorithms for implementing a stemmer. The feature table shows the analyst, at a glance, how systems are common, and vary, in the domain.

The final key piece of domain information acquired from domain experts is the architecture of each system in the domain. System architectures are recorded using a general purpose architecture editor. An example screen for this editor can be seen in figure 9.

The movable palette in the upper left of the figure contains graphical symbols representing architectural elements such as bubbles, boxes, diamonds, pillboxes and so on. The drawings window in the lower left lists all of the architectural drawings for a system; a system may have several architectural representations, for example data flow diagrams, entity-relationship diagrams, structure charts, etc. The title and description windows are editable displays of the current drawing's name and description. The diagram on the drawing surface shows a dataflow diagram that describes the architecture of an information retrieval system.

This architecture editor allows architecture specification using different notations because experts will typically use many different specification notations. The range of diagrams supported by the architecture editor palette includes object diagrams,
dataflow diagrams, structure charts, state transition diagrams, flowcharts, layer or onion diagrams, and mixtures of these.

The facet table displays the clusters promoted and named in the cluster editor. One window of the facet table shows facet names. Selection of a facet name displays all the terms (words and phrases) in the facet, initially in order as determined by the distance of the word or phrase from the facet center. The analyst may adjust the ordering of facet terms in the table. Facets may also be returned to the cluster editor for further refinement.

The generic architecture editor and facet table are similar to the system architecture editor and facet table but contain generic domain information rather than system specific information.

3.3. The third prototype

Even Visual Basic proved too slow a development environment as time and project resources became short. We therefore created another prototype using commercial-off-the-shelf tools (COTS) and freeware. We found that this prototype allowed us to create a version of DARE with much of the planned functionality in a fraction of the time required for either C or Visual Basic. We are currently using
DARE-COTS [Frakes et al. 1997] to refine the DARE domain analysis method, and to investigate how the method can be supported by various commercial tools.

4. Summary and conclusions

We have described DARE, a tool that provides automated support for domain analysis. DARE has been implemented in a series of prototypes used to refine both the tool and the method. DARE structures the domain analysis process by equating it with writing a domain book. The domain book organizes all domain information and becomes the product of domain analysis. DARE provides navigation support for traversing the domain book and for tracing outputs to inputs. It also provides the necessary tools for producing the different parts of the book. The main DARE tools include:

- a form-based interface for acquiring general domain information from experts,
- a graphical architecture editor for recording system architectures,
- a feature table for summarizing system commonalities and variabilities,
- a suite of text processing tools for extracting a domain vocabulary from text sources,
- a clustering tool for deriving faceted classification schemes from the domain vocabulary and for identifying commonalities and variabilities,
- a graphical architecture editor for creating and recording generic architectures,
- a system feature table for recording decisions about commonalities and variabilities in systems based on the generic architecture, and
- a glossary, bibliography, index, and appendices for collecting and organizing reference information.

The information in the domain book representing outputs of domain analysis include a generic architecture, a domain vocabulary, a faceted classification scheme, and a set of descriptors for common functions in the domain. The tool set has been designed to support a systematic and repeatable domain analysis method that employs a simultaneous top-down and bottom-up strategy.

The main conclusion of this work is that domain analysis, a human intensive process, can be partially automated. Other important findings of this project include the following:

- Domain experts are the most important source of information for domain analysis. We have found that there is generally no substitute for domain expert knowledge in deriving generic architectures. Bottom-up information from code and text can serve as valuable clues and checks on expert information, but they are usually at too low a level of abstraction to provide direct information about domain architectures. This observation has resulted in a new and improved methodology for domain analysis – the DARE method, which has a different focus and perspective than other methods.
• The domain book metaphor is crucial for structuring the domain analysis process, and storing and organizing domain information.

• Several text analysis algorithms are useful for domain analysis, including a new algorithm for automatic clustering developed for DARE.

While significant progress has been made in understanding, codifying, and automating the domain analysis process, much work remains to be done. We are currently evaluating DARE to identify potential problem areas and to further refine the DARE method and tool set.

References


