RELIANT Domain Engineering Method for Exception Management

Susan Entwisle, Dr Sita Ramakrishnan, Dr Ian Peake, Dr Elizabeth Kendall
Faculty of Information Technology, Monash University, Melbourne, Australia
scent1@student.monash.edu.au, sita.ramakrishnan@infotech.monash.edu.au,
ian.peake@rmit.edu.au, ekendall@mit.edu.au

Abstract

Programming languages provide exception handling mechanisms to structure fault tolerant activities within software systems. However, the use of exceptions at this low level of abstraction can be error-prone and complex leading to new programming errors. Our research investigates the use of model driven approaches to exception management. In this paper, we present the RELIANT domain engineering methodology. This method provides a tailorable framework that provides guidance for a domain engineer from initial conception to implementation and deployment of reusable modules for a wide range of exception domains.

1. Challenges in the Development of Reliable Software Systems

Reliable software systems must be able to respond to exceptional situations. This is required in order to support continuity of correct services. To achieve this, fault tolerance mechanisms, such as language based exception mechanism, and hardware and software redundancy [1], must provide support for fault prevention, fault tolerance, fault removal and fault prediction.

The design and implementation of language based exception handling is complex. Software developers must anticipate exceptions during design and implement responses using language constructs. In these systems, the exception handling related code is often numerous and complex. This is demonstrated by an analysis of software systems that found up to two-thirds of a software systems' source code is related to error handling [2]. Furthermore, the implementation of the exception handling features is scattered across multiple source files, in different modules, and tangled with the description of numerous other concerns [3].

Exception handling is typically not addressed at the appropriate phase of system development [4]. Traditionally, it has been addressed late in the design and implementation phase of the software development lifecycle. The focus of this effort has been to manage faults that may have been introduced during the requirements analysis, design, and implementation phases. This has resulted from a lack of methodology to support the proper use of exceptions. This approach leads to a situation where the appropriate context in which errors should be detected and recovered is lost. To avoid these problems it is necessary for each phase of the software development lifecycle to incorporate the definition of exception classes within the context of the software system.

The integration of exception handling mechanisms with object oriented languages raises unique design issues. The effective management of exceptional conditions requires detailed context information and the explicit declaration of external exceptions in an interfaces specification. [1] and [5] have found that these exception handling requirements often conflict with the goals of object-oriented design. This can potentially result in decreased application reliability, maintainability, and robustness. Furthermore, object oriented languages traditionally do not provide appropriate constructs to represent exceptions as separate concerns. As a result, exceptions are difficult to modularise, compose and change without modifying other concerns because of high coupling and a lack of cohesion [6].

Our research proposes model driven development as a means for developing reliable software systems. In this paper, we outline a domain engineering method for exception management, called RELIANT. This method supports our extensible model driven exception management framework, referred to as the DOVE framework [7]. The DOVE framework provides an

architecture-based approach to the development of reliable software systems. The goal of the RELIANT domain engineering methodology is to provide a tailorable framework that guides the domain engineer from initial conception to implementation and deployment of reusable modules for a wide range of exception domains.

The remainder of this paper is structured as follows: Section 2 provides a high level overview of the RELIANT domain engineering method. Section 3 describes the domain analysis phase and outlines the activities as they relate to the development of the DOVE framework and the development of domain specific exception modules. Section 4 describes the domain modeling phase, while Section 5 outlines the domain design phase. Section 6 outlines the domain implementation phase. Section 7 discusses related work in domain engineering methods. Finally, Section 8 provides our conclusion and discusses future work.

2. Outline of the RELIANT Domain Engineering Method

RELIANT is an iterative and incremental domain engineering methodology designed to support the systematic analysis, design and implementation of domain specific exception management modules (e.g. network exceptions, database exceptions, etc). RELIANT has been derived from the ODM [8]. The special features of RELIANT include:

- Focus on three concept categories: exceptions, exception policies and services that manage the software failure;
- Feature starter sets for exception management;
- Use of feature diagrams for feature modeling (ODM does not prescribe any feature modeling notation);
- Focus on the development of domain specific languages;

This methodology was also used in the design of the DOVE framework. The RELIANT domain engineering methodology and the DOVE framework can be integrated with other software engineering methods, for example the Rational Unified Process [9], and also with broader frameworks, as required. The characteristics of domain specific exception management modules include the following:

- Main concepts are adequately captured as exceptions, exception policies and common services that manage the software failures;
- Exceptions are adequately represented as abstract data types (ADT) and algorithms that act on the ADTs to manage the software failure;
- Exceptions and the algorithms used to manage them come in a large variety. For example, there are many different kinds of database exceptions, run time exceptions, etc.

Figure 1 illustrates the major activities of the RELIANT methodology. This methodology is configurable and extensible. During development, the various activities in the RELIANT methodology may be scheduled and rescheduled dependent on the specific domain requirements. For example, the key concepts may need to be refined and revised based on insight gained during domain design and implementation. Thus, the process outlined in Figure 1 should be viewed as a default starting point template that can be customised to meet the specific domain engineering requirements.

1. Domain Analysis
  1.1 Domain Definition
    1.1.1 Goal and Stakeholder Analysis
    1.1.2 Domain Scoping and Context Analysis
    1.1.2.1 Analysis of exception domain categories and existing systems
    1.1.2.2 Identification of domain features
    1.1.2.3 Identification of relationships to other domains
1.2 Domain Modeling

1.2.1 Identification of Key Concepts
1.2.2 Feature Modeling of Key Concepts (commonality,variability, dependencies)
2. Domain Design
2.1 Identification of implementation architecture.
2.2 Identification and specification of domain-specific languages.
3. Domain Implementation
3.1 Implementation of domain-specific languages, model transformation, and services.

Figure 1: Outline of RELIANT Domain Engineering Method

3. Domain Analysis

The purpose of the domain analysis phase in RELIANT is to select and scope the domain to collect relevant domain features (requirements), definitions, concepts, and lexicons which are represented in a domain model. Domain analysis consists of two major activities: Domain Definition and Domain Modeling. The primary outputs of this phase are: 1) defined project objectives and a scope of a domain that is in alignment with the project requirements and 2) a domain model that describes the common and variant features of the system within the domain, with rationale for the variations.

3.1 Domain Definition

The first activity in domain definition is to set objectives and identify stakeholders. The complexity of this activity is dependent on the characteristics of the domain and the context of the project.

The next activity is to determine the scope and characterise the contents of the domain by defining its common and variable domain features. The primary objectives of scoping the domain are to ensure that the domain selection is in alignment with the project's objectives, that risks are identified, and that the domain of focus for the project is selected and bound. The domain of focus is the domain that will be modelled and used as a basis for engineering assets. To provide input into scoping the domain and the features to implement, it is useful to assess the domain using selection criteria. The selection criteria provide an informal and subjective framework to assess the domain in the context of the project's objectives and general selection criteria. For each criterion it is necessary to determine the acceptable range of values for successful reuse. The following outlines generic selection criteria that can be used in the analysis of domain-specific exception management modules:

- Is the domain of focus mature?
- Is there an appropriate collection of existing exemplar systems in the domain?
- Are the performance requirements attainable?
- Are domain experts available in the area of the domain of focus?
- Is the domain of strategic interest to the stakeholders?
- Is the domain of focus appropriately sized?
- Does the domain of focus contain sufficient commonality to achieve worthwhile reuse?

3.2 Domain Analysis for Exception Management

The domains of focus in exception management are typically horizontal domains that are distributed across the system. That is, the domain includes only a portion of the functionality implemented in a system, and it is distributed through separate subsystems. In our analysis of the

exception management domain, we have considered two primary sources of domain features: analysis of the application area, and analysis of existing exemplar systems.

We have analysed the application area of exception management. We have also analysed the features of existing language based exception handling mechanisms and exception management libraries. The results of these analyses are shown in Figure 2, Figure 3 and Table 1. Figure 2 shows the exception handling process for responding to an exception. Figure 3 shows the process for managing an unhandled exception in an application.

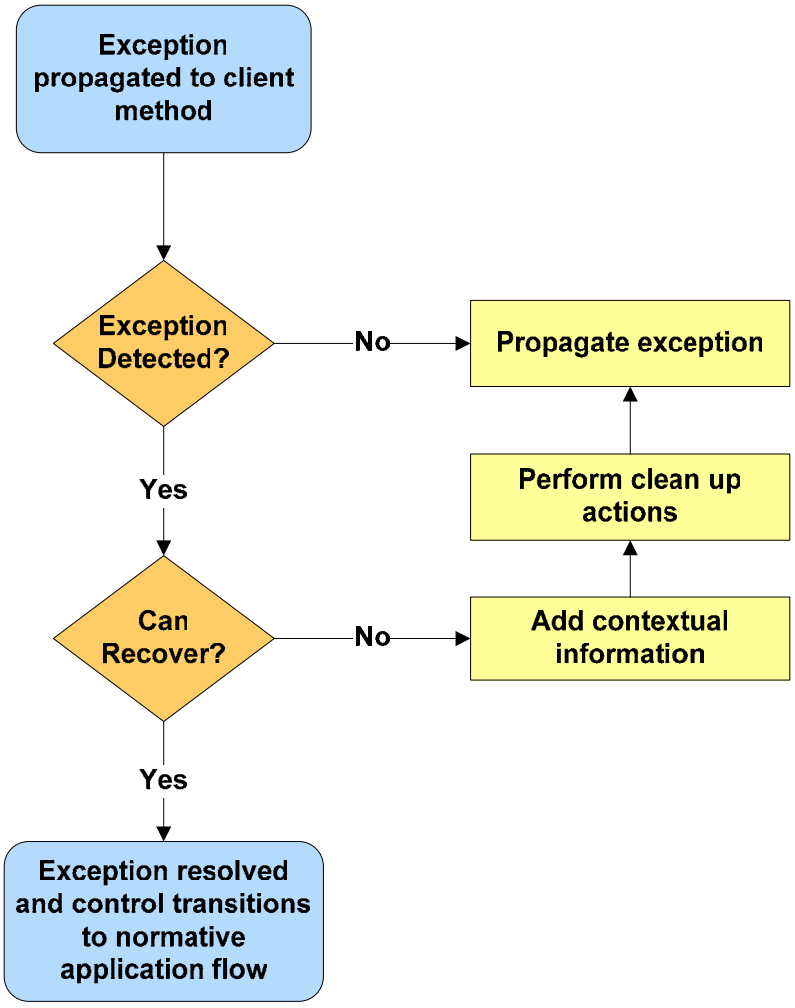

Figure 2: Exception Handling Process (based on [10])

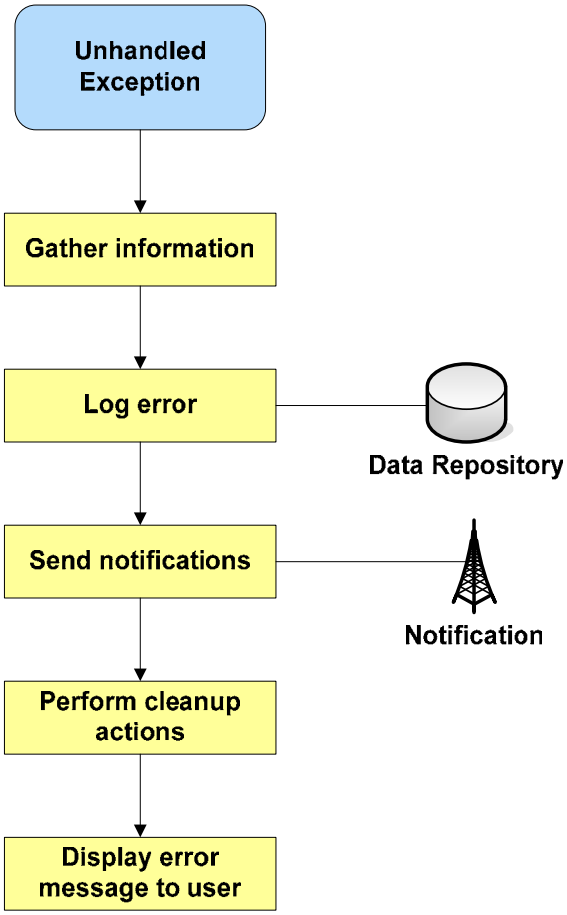

Figure 3: Unhandled Exception Handling Process (based on [10])

Table 1 outlines statistics collected on ten J2EE and .NET open source application patterns of exception handling. Applications A1-A5 are J2EE applications, while A6-A10 are .NET applications. A1-A4 and A6-A9 have been randomly selected based on size and represent a range of application areas (finance, application frameworks, etc). The PetStore reference application was selected because it has been implemented in Java (A5) and .NET (A10).

The statistical data was collected using a custom utility that parsed source code files recursively within a specified directory, to extract data on the number of methods, number of handlers and their associated exception handling response within a class. The responses were categorised into: swallowed exception, logged exception and custom exception. A swallowed exception is a handler that contains no source code statements to respond to the exception. A logged exception is a handler that logs source code statements in response to the exception. A custom exception is a handler with source code that can not be classified as logging source code. The statistical analysis highlights that a common pattern for exception handling is logging and notification. We also found that a large number of exceptions are ignored through empty catch blocks. This pattern causes defects because the propagation of an exception is terminated without returning the system to an appropriate state or taking remedial action to correct the failure. We analysed other common anti-patterns for exception management, including the following:

- **Invalid Termination of Exception (Swallowed Exception)**: terminating the propagation of the exception without returning the system to a valid state.
- **Unanticipated Exception**: propagation of the exception to client that do not have any source code statements to respond to the exception.

- **Exception Propagation from within the Handler**: no exception handling response to manage exceptions that are raised within the handler.
- **Single Catch Block for Multiple Exceptions**: exception handling block for a generic exception that responds to multiple exception types.
- **Mapping Exceptions**: mapping the exception based on type into a common error code.

Table 1: Exception Handling Application Statistics

APPLICATION	# CLASSES	# METHODS	# HANDLERS	# SWALLOWED	# LOGGED	# CUSTOM
A1	1840	31264	2243	119	1381	743
A2	999	12776	4250	92	2434	1724
A3	1247	8567	551	15	8	528
A4	3030	27992	2332	23	262	2047
A5	32	379	48	8	15	25
A6	1887	14933	457	42	67	348
A7	3072	25492	1071	111	72	888
A8	757	13438	940	2	1	937
A9	497	3236	269	4	4	261
A10	43	362	23	0	0	23

The overall conclusion of our domain analysis is that the main features of exception management are common exception types and strategies to manage failures. Each strategy specifies the exception handling responses that need to be invoked to respond to an exception. A strategy can be represented as a reusable exception policy that contains the exception handling actions and algorithms to respond to the failure. The results of the analysis of the application areas and exemplar systems has led to the identification of a number of major types of exceptions and responses which are common in object oriented exception handling mechanisms and libraries. These are summarised in Table 2 for the DOVE framework. The reason for including each of these features in the DOVE framework is also outlined in Table 2.

Table 2: Exception Types and Handling Responses in Feature Model

FEATURE – TYPE/METHOD	REASON
Pre defined (runtime) exception	Pre defined exceptions that represent runtime errors are in object oriented language core libraries. This is mandatory for any exception management mechanism.
User defined (application) exception	User defined exceptions that represent domain specific application exceptions are common in object oriented languages core libraries. This provides the type hierarchy to support extensibility modularity, and polymorphic behaviour. This is mandatory for any exception management mechanisms.
Concurrent exception	Concurrent exceptions have not been addressed as a core design feature in object oriented languages. Rather concurrent exceptions have been supported through locking mechanism and transaction management. To provide a consistent approach the management of concurrent exceptions should be integrated into the exception management mechanism.

FEATURE – TYPE/METHOD	REASON
Exception Policy	The exception handling response frequently consists of a number of actions to identify and respond to the failure. These actions can be aggregated together to form a strategy to manage the exception.
Cleanup actions	Appropriate actions must be undertaken to cleanup resources in the event of an exception.
Log exception action	The logging of contextual information and other related data is a common exception handling response in applications. This information can be used to report and assist in the resolution of faults in applications.
Custom handler action	The execution of custom actions is a common exception handling response in applications.
Send notification action	Notification of a fault is a common exception handling response in applications. This supports notifying key actors of faults in an application.
Propagate exception	Services and methods must be able to propagate unhandled exceptions to the caller for resolution.

To provide input into managing the development of the domain assets, it is useful to indicate the importance of the domain feature. The factors that influence this decision are informal and subjective. However, they are still useful in indicating the importance of various parts of the domain and will help in selecting what parts of the domain should be implemented first. The three following factors influence the priority of the domain feature:

1. Average rate of the domain feature in the analysed application
2. Average rate of the domain feature in the analysed exemplar system
3. Importance of the domain feature to the stakeholder

At the start of this phase, it is important to establish a domain dictionary and a register of the sources of domain knowledge. As the analysis progresses, both the domain dictionary and register need to be updated.

4. Domain Modeling

Domain Modeling involves two main activities: identification of key concepts and feature modeling of the key concepts. We describe these two activities in the following sections.

4.1 Identification of Key Concepts

By definition, RELIANT focuses on domains whose main characteristics are exceptions, exception policies and the common services needed to manage software failures. The identification of the key concepts is usually quite simple based on the analysis of the existing application and exemplar systems e.g. the key exception types in the DOVE framework are pre-defined (runtime) exceptions, user-defined (application) exceptions, and concurrent exceptions.

All faults that can impact a system during its life can be classified into eight viewpoints, as shown in Figure 4. If all combinations of the eight viewpoints were possible, there would be 256 different combined fault classifications. However, not all criteria are applicable to all fault classes. For example, natural faults cannot be classified by objective, intent and capability. Thus, [11] have identified 31 likely combinations, which are shown in Figure 5. The fault classes in Figure 5 belong to three groups. They are:

- **Development Faults**: all fault classes that occur during development.
- **Physical Faults**: all fault classes that affect hardware.

- **Interaction Faults**: all fault classes that include external interaction.

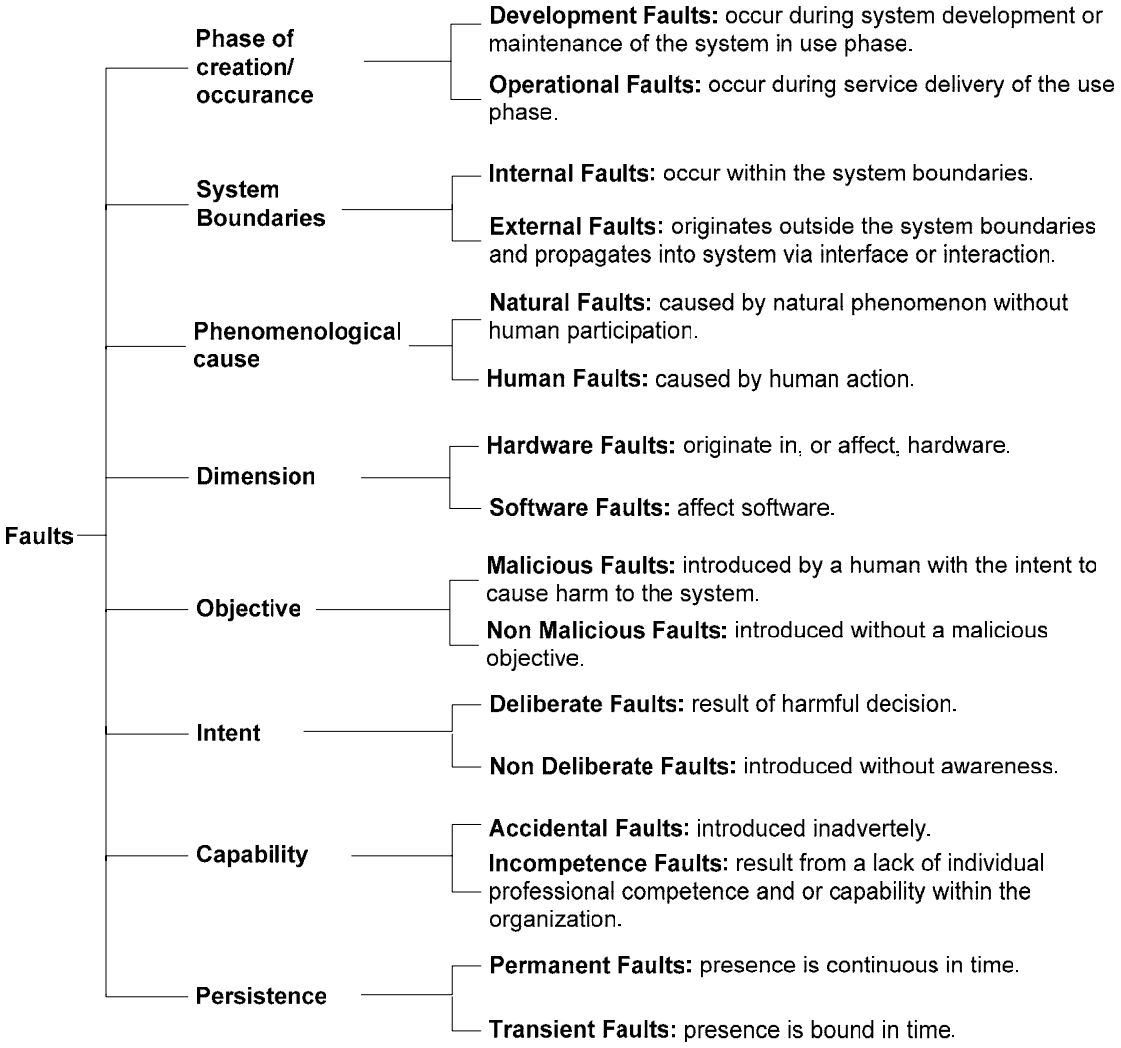

Figure 4: Fault Classification (based on [11])

Knowledge of all the possible fault classes supports the domain engineer in the systematic identification of key concepts for domain specific exception management modules. The fault classification and the map to the groupings provide a structured set of viewpoints for the domain engineer to evaluate when analysing existing exemplar systems. The identification of the key concepts enables us to formalise the specification of the domain definition. The domain dictionary must be updated with the key concepts and their attributes.

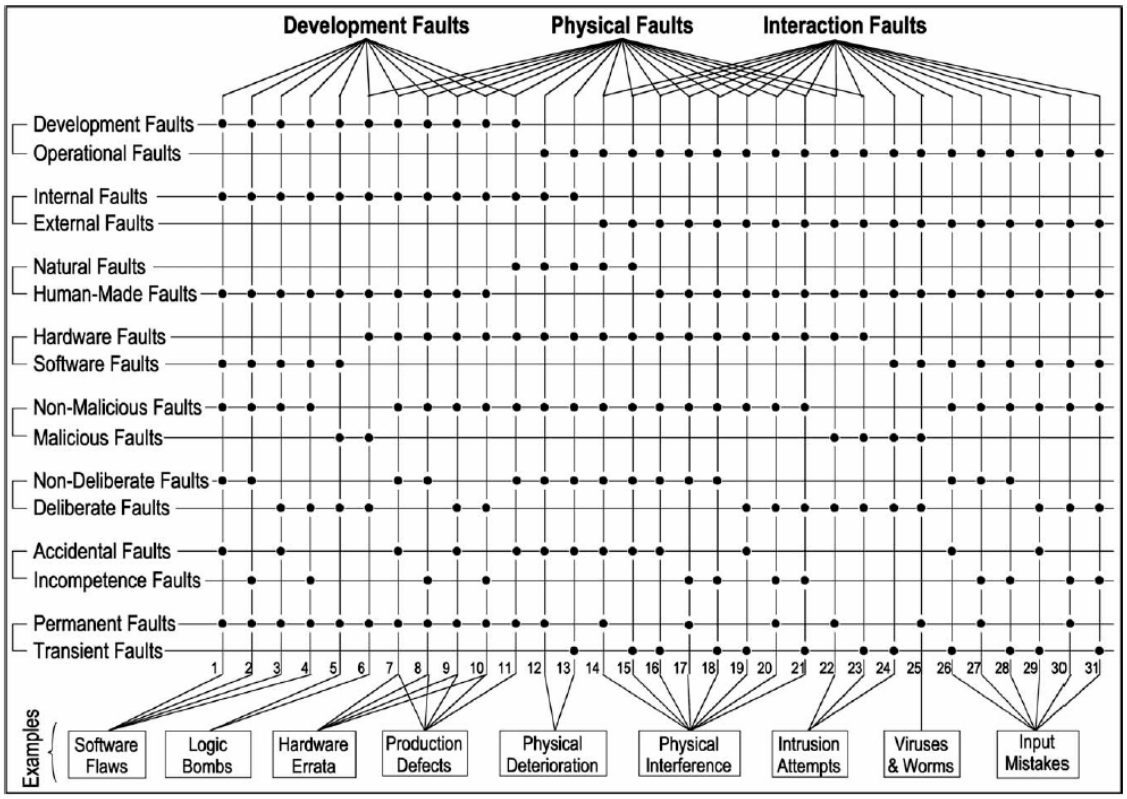

Figure 5: Fault Classification Mapped to Groups (based on [11])

4.2 Feature Modeling

The purpose of feature modeling is to develop feature models of the concepts in the domain. Feature models define the common and variable features of the concepts instances, the constraints and the dependencies between the features. The only addition to RELIANT is the feature starter sets for exceptions and exception policies outlined below.

Feature Starter Set for Exception

The following is the feature starter set for exception:

- **Attributes**: are the named properties of the exception, such as machine name, time of exception, and other contextual data related to the exception.
- **Operations**: are actions that need to be performed to manage the exception including providing contextual information on the exception state to the exception handlers responding to the software failure.
- **Synchronisation**: if the exception is used in a multithreaded environment, we have to synchronise access to the shared data.
- **Persistency**: some applications may require the ability to store the exception on disk (e.g. in a file or database). In such cases, we need to provide mechanisms to both persist and retrieve appropriate parts of the state of the exception type.
- **Perspectives and Security**: different clients may have different perspectives and security requirements on the exception type. Thus, we may consider organising the exception types into subjects based on the different perspectives and security requirements of the client.

Feature Starter Set for Exception Policies

The following is the feature starter set for exception policy:

- **Exception Type**: defines the exceptional event that invokes the execution of the exception policy.
- **Exception Signaller**: in the event of an exception the exception type is evaluated in the context of the exception signaller location, including both type and method, to determine what policy to invoke. This approach provides the flexibility to associate different policies for the same exception type based on the exception signaller's location.
- **Exception Handler Chain**: each exception policy has one or more corresponding exception handlers that perform actions to manage the exception.
- **Post Handling Action**: The post handling action occurs after the exception policy execution has completed. The supported post handling actions are: none, rethrow and notify user. The rethrow is used to indicate that the exception could not be resolved and needs to be rethrown. The notify user post handling action is used to communicate across tiers that an exception occurred and has been resolved.

The feature sets should be extended as soon as new relevant features are identified.

5. Domain Design

The purpose of the domain design phase in RELIANT is to develop a domain specific package consisting of exception policies and libraries to support different domain specific exception classes. Domain design builds on the results of domain modeling i.e. the feature models of different aspects of the exception and exception policies. The primary outputs of this phase are: 1) an architectural model that specifies the configuration, constraints and range of variability for the features in the domain of focus and 2) the specification of the required features, interfaces and behaviour of each asset for implementation purposes.

5.1 Domain Design Activities

Domain design in RELIANT involves the six activities described in this section.

i. Identify Packages

The exception management library to be developed is divided into a number of packages. A package consists of the exception policies and libraries required to implement the features specified in the feature model. Packages are specified using UML package diagram notation in order to represent the package under development.

ii. Identify Exception Policies

Exception policies are provided in the domain specific exception management module to their users as a graphical domain specific language. This specifies both the chain of exception handlers that will be invoked in order to respond to the exception and the post handling action. The configuration of an application to invoke the exception policy when certain exception types occur is modelled in the application's UML class diagram.

iii. Identify Interactions between Exception Policies

The data exchange requirements and dependencies between exception policies need to be identified. For example, the outcome of one exception policy may be to rethrow a modified exception type for resolution. This new exception may invoke another exception policy, which has been configured for the application, for resolution. The required contextual data may need to be added, modified or deleted from the exception instance by the handlers executed by the first exception policy. This may be required so that the second policy can resolve the software failure.

iv. Scope Exception Policies

The features to be covered by the implementation of the exception policy have to be selected based on the priorities recorded in the feature models, the current project goals, resources and constraints.

v. Specify Exception Policies

The exception policies are specified using the exception policy domain specific language.

vi. Specify Exception Library Components

The exception policy specifies the chain of exception handlers to resolve the software failure. The exception handler type interface and operations need to be specified as a component model using UML class diagrams.

6. Domain Implementation

The purpose of the domain implementation phase in RELIANT is to build the reusable assets to support application engineering for exception management. Domain implementation builds on the results of domain design i.e. the architectural models, component models and policies. The primary outputs of this phase are the reusable assets to support application engineering for exception management (for example, concrete domain-specific languages, components, code generators, etc). The implementation of the domain specific languages, libraries and common services of the DOVE framework is presented in [7].

The different parts of the exception management module require different implementation techniques:

- Exception types and operations can be implemented using classes, inheritance, and templates. All of these abstraction mechanisms are available in C#.
- Exception policies are specified using the exception policy domain specific language in a domain specific modeling environment.

The exception type must inherit from the base domain exception class in the DOVE framework. The application configuration that links the application's classes and services to the exception policy is specified in the application model. The metadata in this model is used by the model transformation engine to automate the generation of the application's configuration file. The application model and the model transformation engine is presented in [7].

7. Related Work

A large number of domain engineering analysis and design methods are being used in the development of software-product lines. To date, three surveys on domain engineering analysis and design have been published. [12] and [13] focus on domain analysis methods while [14] builds on these surveys by providing more recent developments in domain engineering methods. The major domain engineering methods are Feature-Oriented Domain Analysis (FODA) [15] and Organizational Domain Modeling (ODM) [8]. FODA is a domain analysis method that is based on identifying the features of a family of systems. The FODA process consists of three phases: context analysis, domain modeling and architecture modeling. ODM is a domain engineering method that leverages ideas from many different domain engineering methods and other non-software disciplines, such as organizational management, re-engineering, etc. [14]. ODM integrates both the organizational and strategic aspects of domain planning, domain modeling, architecture engineering and asset base engineering. FODA and ODM are insufficient alone to capture domain knowledge for non-functional areas, such as fault tolerance, real-time support, etc, as they require specialised modeling techniques to be integrated into a generic domain engineering method [14].

Although domain engineering focuses on providing the processes and modeling techniques to support identifying and representing commonality and variation within a multi-system scope, OOA&D methods provide effective and widely used system modeling techniques for single systems. [14] subsequently argue that domain engineering and OOA&D are candidates for

integration. Furthermore, it is highlighted that this has been occurring within the community with the development of methods, such as OORam [16], Reuse Driven Software Engineering Business [17], and FeatureRSEB [18]. To successfully integrate these methods requires consideration of a number of areas including the methods, goals, principles, process, models and notations. [14] provide a survey of OOA&D methods that have incorporated some aspects of domain engineering.

8. Conclusion and Future Directions

We have developed the RELIANT domain engineering method that provides a tailorable framework to support domain engineers in the design of reusable domain specific assets for different exception domains. The method focuses on three concept categories: exceptions, exception policies and services that manage software failures. The RELIANT domain engineering method provides guidance, common patterns for exceptions, a fault analysis framework to aid in identifying exceptions, feature diagrams and feature starter sets for exception management, and guidance on the design of domain specific modules for different classes of exceptions.

A further area of work is the extension, refinement and further case studies for the RELIANT domain engineering method. This includes the development of templates to support the use of the method, tool support to enable the integration of people, processes and tools during a domain engineering project, tool support to enable the customisation of the method, and further case studies to assess and refine the RELIANT domain engineering method.

References

[1] A. F. Garcia, C. M. F.Rubira, A. Romanovsky, and J. Xu, "A Comparative Study of Exception Handling Mechanisms for Building Dependable Object-Oriented Software," *The Journal of Systems and Software*, vol. 59, pp. 197--222, 2001.

[2] F. Cristian, "Exception Handling," *Dependability of Resilient Computers*, pp. 68-97, 1989.

[3] M. Lippert and C. V. Lopes, "A Study on Exception Detection and Handling using Aspect-Oriented Programming," in *Proceedings of the 22nd international conference on Software engineering*: ACM Press, 2000, pp. 418--427.

[4] C. Rubira, R. de Lemos, G. R. M. Ferreira, and F. Castor Filho, "Exception handling in the development of dependable component-based systems," in *Software: Practice and Experience*, vol. 35. Kent, U.K.: John Wiley & Sons, Ltd., 2005, pp. 195-236.

[5] R. Miller and A. Tripathi, "Issues with Exception Handling in Object-Oriented Systems," *Lecture Notes in Computer Science*, vol. 1241, 1997.

[6] J. G. Gray, "Aspect-Oriented Domain-Specific Modeling: A Generative Approach using a Metaweaver Framework," in *Computer Science*. Nashville, Tennessee: Vanderbilt University, 2002.

[7] S. Entwisle, H. Schmidt, I. Peake, and E. Kendall, "A Model Driven Exception Management Framework for Developing Reliable Software Systems," in *Proceedings of 10th International IEEE Enterprise Computing Conference*, IEEE, Ed. Hong Kong: IEEE, 2006.

[8] M. Simos, D. Creps, C. Klinger, L. Levine, and D. Allenmang, "Organizational Domain Modeling (ODM) Guidebook, Verison 2.0," 1996.

[9] IBM, "Rational Unified Process", http://www-306.ibm.com/software/awdtools/rup/, accessed January 2007.

[10] Microsoft, "Microsoft Patterns and Practices: Exception Handling Application Block", http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/ehab.asp, accessed 20th March 2005.

[11] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *Dependable and Secure Computing, IEEE Transactions on*, vol. 1, pp. 11-33, 2004.

[12] G. Arango, "Domain analysis: from art form to engineering discipline," in *IWSSD '89: Proceedings of the 5th international workshop on Software specification and design*: ACM Press, 1989, pp. 152--159.

[13] S. P. Wartik and R. Prieto-Díaz, "Criteria for Comparing Reuse-Oriented Domain Analysis Approaches," *International Journal of Software Engineering and Knowledge Engineering*, vol. 2, pp. 403-431, 1992.

[14] U. Eisenecker and K. Czarnecki, *Generative Programming: Methods, Tools, and Applications*, 1st ed: Addison-Wesley Professional, 2000.

[15] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Software Engineering Institute, Technical Report CMU/SEI-90-TR-021, November 1990.

[16] T. Reenskaug, *Working with Objects: The OOram Software Engineering Method*, 1st ed: Manning Publications Co., 1995.

[17] I. Jacobson, M. Griss, and P. Jonsson, *Software Reuse : Architecture, Process and Organization for Business Success*, 1st ed: Addison-Wesley Professional, 1997.

[18] M. L. Griss, J. Favaro, and M. Alessandro, "Integrating Feature Modeling with the RSEB," in *ICSR '98: Proceedings of the 5th International Conference on Software Reuse*: IEEE Computer Society, 1998.