DRACA: Decision-support for Root Cause Analysis and Change Impact Analysis

by

Sarah Nadi

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2009

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Most companies relying on an Information Technology (IT) system for their daily operations heavily invest in its maintenance. Tools that monitor network traffic, record anomalies and keep track of the changes that occur in the system are usually used. Root cause analysis and change impact analysis are two main activities involved in the management of IT systems. Currently, there exists no universal model to guide analysts while performing these activities. Although the Information Technology Infrastructure Library (ITIL) provides a guide to the organization and structure of the tools and processes used to manage IT systems, it does not provide any models that can be used to implement the required features.

This thesis focuses on providing simple and effective models and processes for root cause analysis and change impact analysis through mining useful artifacts stored in a Configuration Management Database (CMDB). The CMDB contains information about the different components in a system, called Configuration Items (CIs), as well as the relationships between them. Change reports and incident reports are also stored in a CMDB. The result of our work is the Decision support for Root cause Analysis and Change impact Analysis (DRACA) framework which suggests possible root cause(s) of a problem, as well as possible CIs involved in a change set based on different proposed models. The contributions of this thesis are as follows:

- An exploration of data repositories (CMDBs) that have not been previously attempted in the mining software repositories research community.
- A causality model providing decision support for root cause analysis based on this mined data.
- A process for mining historical change information to suggest CIs for future change sets based on a ranking model. *Support* and *confidence* measures are used to make the suggestions.
- Empirical results from applying the proposed change impact analysis process to industrial data. Our results show that the change sets in the CMDB were highly predictive, and that with a confidence threshold of 80% and a half life of 12 months, an overall recall of 69.8% and a precision of 88.5% were achieved.
- An overview of lessons learned from using a CMDB, and the observations we made while working with the CMDB.

Acknowledgments

I would like to thank my supervisor, Prof. Ric Holt, for his continuous support. He patiently guided me through the dark pathway, commonly referred to as research, until I found a way through. I would also like to thank him for taking the time to improve my writing and presentation skills. His suggestions and advise always made a lot of difference. Thanks a lot Ric!

Many thanks to Prof. Mike Godfrey and Prof. Ladan Tahvildari for taking the time to read my thesis, and providing me with valuable feedback. I would also like to thank Dr. Ian Davis for his continuous professional and personal support. Thank you Ian for always taking the time to brainstorm with me for ideas, and above all, thank you for being a great friend.

I would also like to thank Serge Mankovskii along with many of CA's research recipients for their valuable input, guidance and suggestions. I am also grateful to CA's GIS team for providing me with data, and for taking the time to guide me through it. I would also like to thank our participating CA customers for their contributions.

To Olga Baysal, thanks for constantly giving me the tips and tricks to graduate school survival. The SWAG experience really wouldn't have been the same without you! To Ghada, Amr, Heba, Amira, and May, thanks for putting up with my absence and for still being there despite the distance. To all the new friends I made here, thanks for making my stay at Waterloo fun and memorable.

To Karim, I simply wouldn't have done it without you! Thanks for patiently listening to my endless complaints of how I'll never finish. You always gave me a confidence boost when I most needed it, and always provided me an enjoyable escape from the endless work.

Last, but not least, my dear parents. I really owe everything to you. I would like to thank you for giving me this opportunity to broaden my horizon and explore a different experience. No words can ever repay you for all the endless love, trust and support you always provided me.

Dedication

This thesis is dedicated to my parents who always believed in me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Information Technology (IT) systems are the basis of most business services today. Let us take a simple example of a photo printing service which allows customers to buy its services online. Assume something goes wrong with their website which as a consequence prevents customers from logging into their accounts. This in turn prevents customers from uploading their photos for printing which might cause them to divert to another company offering the same service. This situation could result in a financial loss to the company. It is, therefore, essential to properly manage such an IT system, and to quickly recover from any problem to minimize financial losses as much as possible.

Two of the important activities of Enterprise IT Management (EITM) are *root cause analysis* and *change impact analysis*. Root cause analysis is the process of identifying the primary cause of a failure in a system. Quick and accurate root cause analysis means spending less time in identifying the cause of a problem which results in a shorter time to recovery. In practice, it is common to find that changes to an IT system may cause unforeseen problems. It is, therefore, important to carefully plan any change to the system, and to proactively identify all impacted items so that they can be systematically changed according to the plan. Accordingly, change impact analysis is the process of identifying all entities that will be impacted by a change to avoid any unexpected consequences.

For example, if a security patch is to be installed on a server, analysts should ensure that none of the applications hosted by this server will be negatively impacted. This may occur if the security path prevents an application from accessing

certain remote resources, for example. In this case, the set of entities to change, i.e. the change set, will include not only the server, but all of the affected applications as well. In order to successfully implement a change without causing costly disruptions to the system, the correct change set should be identified before the change is implemented. Correctly planning changes in an IT system results in a more maintainable system with fewer undetected errors. As the examples show, change impact analysis is a proactive measure while root cause analysis is a reactive measure. However, both processes seek to minimize undesirable IT disruption.

As IT companies become more aware of these needs, different tools and processes are being developed to manage and monitor IT systems. For example, the Information Technology Infrastructure Library (ITIL) [25] has developed standards that lay the foundation for service management, and outline the tools needed to achieve it. Such tools include a Service Desk to manage incoming requests and complaints, a Configuration Management Database (CMDB) to keep track of the different components in the system and the relationships between them, network monitoring tools etc.

A CMDB is used to store information about the various critical components in a system including hardware, software, and services provided by the company. It records information about these items, their change history, their incident history, as well as the relationships between them. Each item stored in the CMDB is referred to as a Configuration Item (CI). Thus, a CMDB serves as the repository for important information about a system which can be used in decision making processes such as root cause analysis and change impact analysis [25].

However, despite ITIL providing the standards of what (in precise) needs to be tracked in an IT system, and the tools needed to track them, there are no formalized models that explain how this tracked information can be used to provide accurate root cause analysis and change impact analysis. Standardized models that make use of the different gathered information to provide decision support for the management of IT systems are therefore needed.

The work described in this dissertation was carried out in collaboration with CA Labs Canada. CA is one of the providers of IT management tools such as Service Desk and CMDB. Our collaboration with CA allowed us to contact its employees and customers. We have been investigating CMDBs from the perspective of CA CMDB experts, as well as from the perspective of three of CA customers who have been using CMDBs. From our discussions, we have identified key usages of the CMDB as well as missing features from the customers' perspective. Our interviews

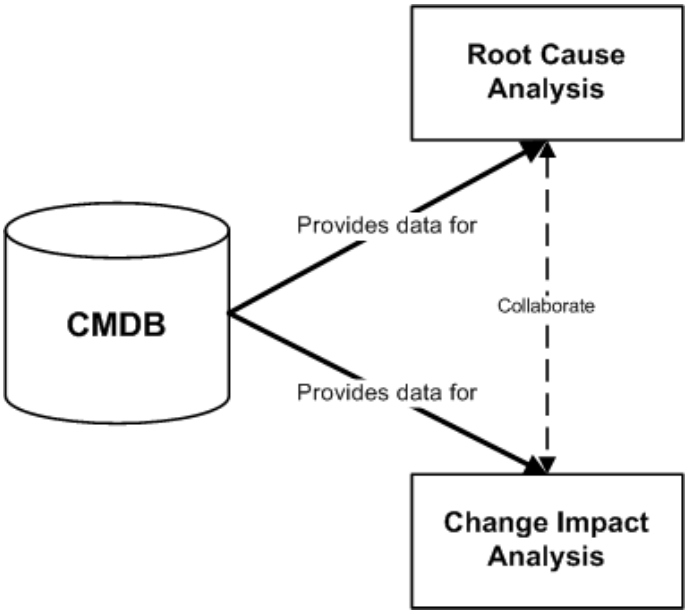

Figure 1.1: High Level Overview of the DRACA Framework

have confirmed that root cause analysis and change impact analysis are critical processes in any large organization, and that more intelligent decision support for these processes is needed.

1.2 Overview of Thesis

The work described in this dissertation seeks to provide semi-automated decision support for root cause analysis and change impact analysis. To achieve these goals, we must identify what information is needed to support these analyses, and how this information can be extracted and modeled. Much like model-driven engineering and model-driven development aim to provide platform-independent and technology-independent architectural models [17], we aim to present generalized models that can be used for root cause analysis and change impact analysis. As the processes and tools that help manage IT systems mature, more research will be geared towards developing such models for the usage of this information. These models should identify the exact data needed to accomplish the specified goals, and how this data will be used. The models should not be specific to a single type of repository, and should be independent of the structure of the data. This thesis is a first attempt at providing these models.

In this work, we present DRACA, a Decision support framework for Root cause

Analysis and Change impact Analysis along with its underlying models. In this work, we rely on the data available in a CMDB. However, the design of our presented models and processes are independent of the structure of the data, and could easily leverage other data sources. In this sense, our models provide a basis for what information needs to be recorded in order to perform root cause analysis and change impact analysis, but does not specify a specific format or process for recording this data.

Figure 1.1 shows a high level overview of the DRACA framework. DRACA has two underlying models: a causality model for root cause analysis, and a matrix-based model with *support* and *confidence* measures for change impact analysis. In our work, both the models are populated with data from the CMDB. However, our models are not geared towards the structure of a particular CMDB. They are applicable to other repositories as long as the information needed to populate our models is stored there. Within each type of analysis, the data from the CMDB is then processed and represented in the respective model to provide a quantitative correlation between CIs. In root cause analysis, the model shows the probability that CI x could be the root cause for CI y while in change impact analysis, the model shows the probability that if CI x is changed, CI y might change as well.

When a CI encounters a problem, the root cause analysis process provides a ranked list of possible root causes. When a change is proposed to a CI, the change impact analysis process provides a ranked list of CIs that may be impacted by this change and so might need to be changed as well. Although each process uses its own models for analysis, both processes may collaborate together to confirm their suggestions. For example, if a CI x is likely to be a root cause for a problem in CI y then it may also likely that a change in CI x would induce a change in CI y or vice versa.

1.3 Thesis Contributions

This work combines ideas from several research fields. To populate our models, we mine the needed data from the CMDB. This is a largely unexplored topic within the mining software repositories (MSR) field. Most of the work done in the MSR community has been performed on repositories that store artifacts related directly to software development, such as source code revisions and defect reports. Providing results from system level repositories such as CMDBs is therefore a new contribution to the research community.

The models for root cause analysis and change impact analysis provided in this thesis will lay the foundation for further models to be created, and for additional sources of information to be added to provide better decision support for IT management. The field work performed in this thesis provides guidelines to the practical needs of customers providing a link between research and the industry. Additionally, DRACA is unique as it combines two important processes, root cause analysis and change impact analysis, under one framework. The main contributions of this work are:

- An exploration of data repositories (CMDBs) that have not been previously attempted within the mining software repositories research community.
- A causality model providing decision-support for root cause analysis based on this mined data.
- A process for mining historical change information to recommend CIs for future change sets based on a ranking model. *Support* and *confidence* measures are used to make the suggestions.
- Empirical results from applying the proposed change impact analysis process to industrial data. Our results show that the change sets in the CMDB were highly predictive, and that with a confidence threshold of 80% and a half life of 12 months, an overall recall of 69.8% and a precision of 88.5% were achieved.
- An overview of lessons learned from using a CMDB, and the observations we made while working with the CMDB.

1.4 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 provides background concepts and definitions. This includes introducing ITIL, and explaining several of its processes such as configuration management, change management and their accompanying tools such as the Service Desk and the CMDB. This chapter also a detailed overview of root cause analysis and change impact analysis. Chapter 3 then presents related research in the various fields our work combines. Chapter 4 presents the overall design of the DRACA framework. This chapter also describes the industry interactions we had and the requirements we gathered from them.

Chapter 5 and Chapter 6 present the core work of this thesis where Chapter 5 discusses the model used for root cause analysis, and Chapter 6 discusses the model used for change impact analysis. Finally, Chapter 7 summarizes this thesis while presenting possible future directions for this work.

Chapter 2

Background

This chapter presents background information about the concepts and tools used in this work. Section 2.1 introduces the Information Technology Infrastructure Library (ITIL) along with some of its processes and tools. Section 2.2 then explains terminology used in our models. Section 2.3 and Section 2.4 define root cause analysis and change impact analysis respectively. Section 2.5 defines causality graphs as used in the context of this work.

2.1 Information Technology Infrastructure Library (ITIL)

In our work, we concentrate on systems that generally follow ITIL [25], which is a set of standards concerned with providing best practices for business effectiveness and efficiency in the use of information systems. Since the mid 1990s, ITIL has been recognized as the de facto standard for service management [25]. ITIL provides general guidelines for a service management framework. It outlines the processes needed such as Configuration Management and Change Management as well as the ideal tools needed to accomplish these processes such as CMDBs, Service Desk, security management tools etc.

2.1.1 Configuration Management

ITIL defines configuration management as "the process of identifying and defining configuration items in a system, recording and reporting the status of configuration

items and requests for change, and verifying the completeness and correctness of configuration items" [25]. A configuration item (CI) is a component of an IT infrastructure, such as a server or a software application, which is put under the control of the configuration management process. CIs and their interrelationships are modeled in the Configuration Management Database (CMDB). The level of detail stored in a CMDB varies from one organization to another in scope and granularity. For example, a computer can be represented as one CI, or it can be represented as various CIs including the operating system used, the keyboard used etc.

The goal of configuration management, apart from accounting for all IT assets and their configuration, is to provide decision support for change management, incident management, problem management, and release management [25]. These processes depend on having the correct information stored in the CMDB. Accordingly, the information in the CMDB should be accurate and the stored CIs and their relationships should reflect the actual state of the system.

2.1.2 Change Management

Change management in ITIL is defined as "the process of controlling changes to the infrastructure or any aspect of services, in a controlled manner, enabling approved changes with minimal disruption" [25]. ITIL emphasizes the need to identify other CIs that will be impacted whenever a change to a specific CI is proposed. Change management is responsible for managing changes to both hardware and software components. These changes may be a solution to a previous incident or problem, or may be a proactive measure to reduce cost or improve performance.

2.1.3 Service Desk

The Service Desk is the interface employees and customers use to report any problems they face or to request changes [25]. The Service Desk is similar to the commonly known concept of a help desk. The main idea behind having a Service Desk application is to have a single point of contact which receives all incoming requests. Requests (incidents, problems, changes etc.) are then routed to the appropriate teams.

Ideally, the Service Desk would have an underlying or integrated CMDB such that when a request for a particular CI is received, analysts can easily view the

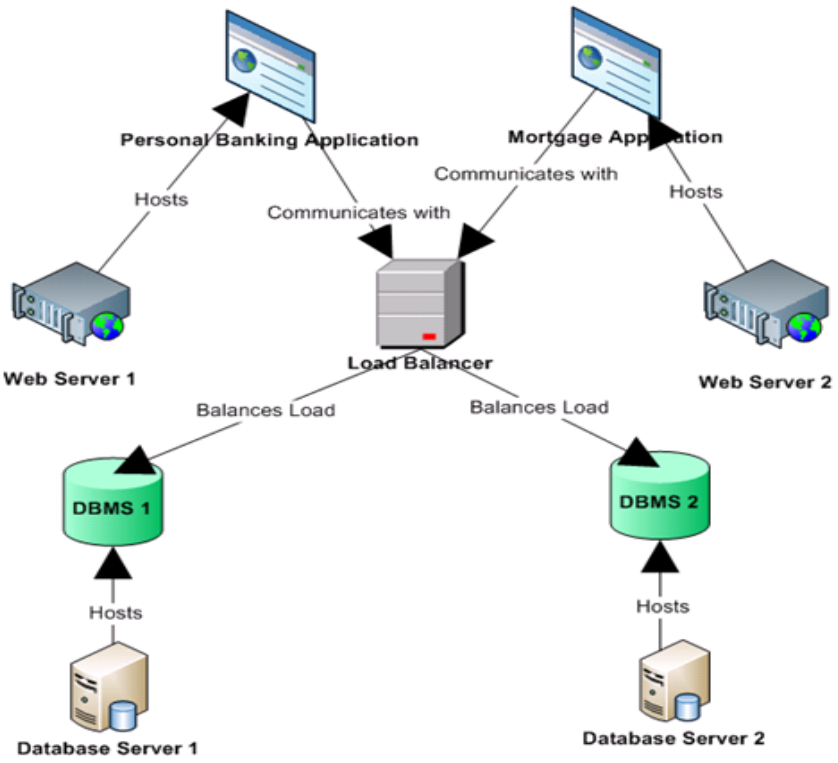

Figure 2.1: An IT System Stored in a CMDB

relationships this CI has to others. Additionally, it would allow viewing a CI's incident, problem and change history.

2.1.4 Configuration Management Databases (CMDB)

A Configuration Management Database (CMDB) is useful in Enterprise IT Management (EITM) since it provides information about the various critical components in a system including hardware, software, and services provided by the company. It records the configuration of these items, their change history, their incident history, as well as the relationships between them. Each item stored in the CMDB is referred to as a Configuration Item (CI). A CMDB usually provides visualization capabilities to view the different CIs in the system and their inter-relations. Figure 2.1 shows an example CMDB to illustrate the concepts of CIs and relationships. In this example, there are two software applications being hosted by two different web servers. Both applications communicate with the same databases through a load balancer. Such a visualization allows an IT analyst to better understand the system at hand.

A CMDB provides a basis for decision making processes such as Incident Man-

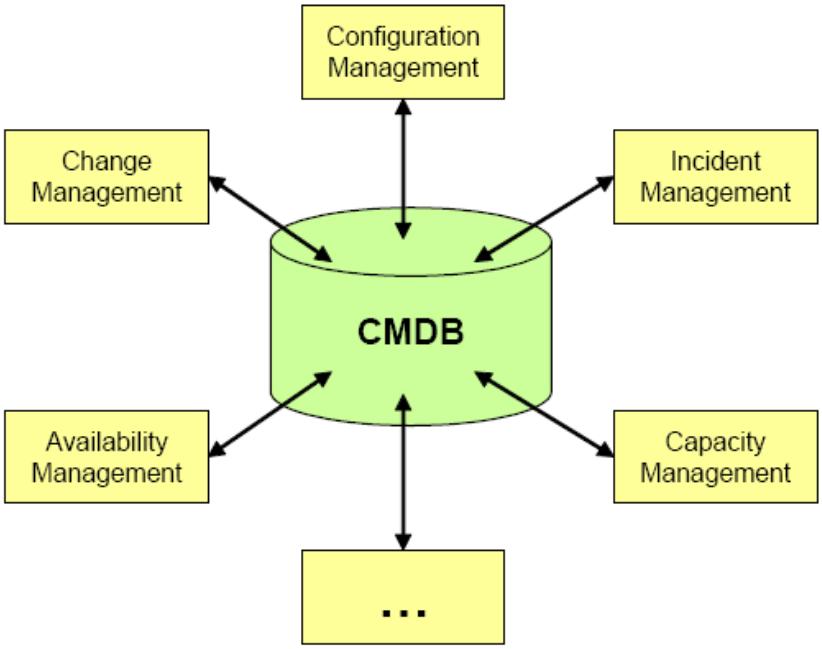

Figure 2.2: Role of a CMDB

agement, Change Management, etc. Figure 2.2 shows how a CMDB is central to these processes [9].s More formally, Messineo and Ryder define seven fundamental use cases of a CMDB [20]. These include impact analysis, root cause determination, change governance, auditing and compliance, resource optimization, services mapping, and service performance planning.

2.2 Terminology: Faults, Failures, and Incidents

The terminology used in this work follows the ITIL standards [24]. A *fault* is a design flaw or malfunction that causes a *failure* of one or more CIs. A *failure*, which generally caused by a fault, is the loss of ability to operate to specification, or to deliver the required output. A failure may cascade to cause more failures in other CIs. A failure, or a failure cascade, may eventually cause an *incident*. An *incident* is an event that is not part of the standard operation of a service and that may cause an interruption to, or a reduction in, the quality of that service. An incident is externally observable and is usually recorded in an incident report. Figure 2.3 illustrates these definitions.

It is worth noting that in the software engineering literature, the terms *error*,

Figure 2.3: Faults, Failures and Incidents in ITIL

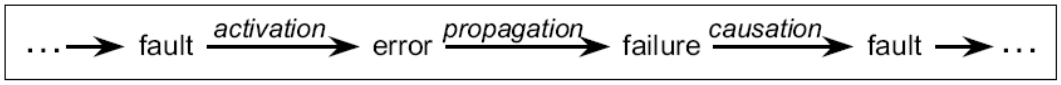

Figure 2.4: Traditional Error Propagation

error latency, and *fault latency* have been commonly used in this context [3]. When compared to the ITIL definitions described in the previous paragraph and in Figure 2.3, the difference between the traditional software engineering definitions and ITIL definitions is the introduction of the term *incident*. We differentiate between a failure and an incident in that a failure is not necessarily noticeable at a service level to the user. In other words, an incident can be thought of as a service failure. Figure 2.4 show the traditional cause effect relationships, especially in the fault tolerance area of research, as presented by Avizienis et al.[3]. The error latency is the time it takes for a fault to be activated and manifested as an error while the error latency is the time it takes an error to propagate and become a failure.

2.3 Root Cause Analysis

A *root cause* is the underlying original fault leading to a particular incident. *Root cause analysis* tries to map an incident to its underlying fault. For example, the inability to view a certain web page is an incident. A failure associated with this incident may be the inability to access the web server hosting the web page. The root cause of the problem may be that someone changed the settings in the Domain Name Server (DNS), and thus the web server is no longer reachable by that name. In the literature, root cause analysis has also been commonly referred to as fault localization, fault diagnosis, and fault identification [5, 16, 32].

2.4 Change Impact Analysis

A *change* is the addition, modification, or removal of anything that could affect on IT services. A poorly planned change may lead to a fault in the system. *Change impact analysis* tries to predict if a change will cause a fault which may lead to

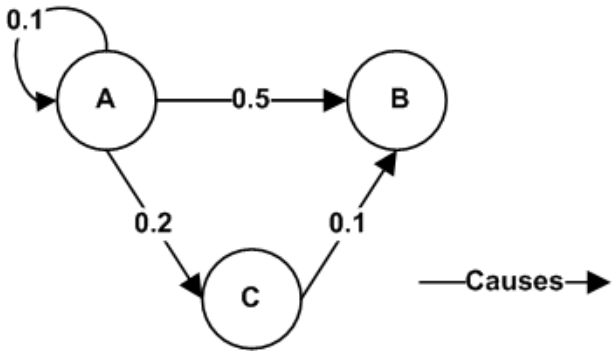

Figure 2.5: Example Causality Graph

an incident. The concept of change impact analysis exists in different domains. For example, in medicine, if the dosage of a certain medication is changed for a patient, the doctor must determine if this will affect the required dosages of other medications the patient is taking. For software systems, Arnold and Bohner [2] define software impact analysis as "the determination of potential effects to a subject system resulting from a proposed software change". This involves identifying which other components will need to be changed.

2.5 Causality Graphs

A *causality graph*, records the cause and effect relations among different components of a system. The nodes are the components, and edges are the relations. A causality edge from x to y indicates that a problem in x could cause a problem in y. Thus, a path on this graph shows a cause-effect chain. Causality graphs often model the cause/effect strength of the relationships through probabilities. Figure 2.5 shows a simple example of a causality graph. Here, an event in A can cause an effect in B as well as an effect in C. The propagation of effect from A to B is independent of that from A to C which means that the probability of the outgoing edges from A do not have to add up to 1.0. As the graph in Figure 2.5 indicates, there is a 50% probability that a change or event in A would cause an effect on B, but only a 20% chance that it will have an effect on C. As shown in Figure 2.5, in our definition, the causality graph does not necessarily have to be a Directed Acyclic Graph (DAG) [8]. The edges must be directed, but the graph does not necessarily have to be acyclic. As will be discussed in Section 5.5, our causality model does account for cycles in the graph.

2.6 Summary

In this chapter, we presented some of the key terminologies and tools that are used in the remainder of this thesis. We defined Configuration Management since accounting for all CIs and their relationships provides a basis for root cause analysis. On the other hand, Change Management involves the process of change impact analysis. We elaborated on the concept of CMDBs as they are used as a main source of information in this work. The terminologies of faults, failures, incidents, and change provide a basis for understanding root cause analysis and change impact analysis as used in our framework. In this chapter, we have also defined causality graphs as used in the context of the root cause analysis model presented in this thesis. In the next chapter, we will present related research in the areas of root cause analysis and change impact analysis. Some of the related research also uses some of the concepts and terminologies defined in this chapter.

Chapter 3

Related Work

Root cause analysis and change impact analysis are problems that have existed for some time. However, there are different levels at which these problems can be dealt with, and these levels have drawn different amount of attention in the literature. Root cause analysis and change impact analysis can be done at the source code level, at the network level or at the service level. Since the different levels have several commonalities, we present related work from all three types although our work concentrates on the service level. Section 3.1 presents related root cause analysis and change impact analysis work on the source code level. Section 3.2 then presents related work on the network level. Finally, Section 3.3 presents root cause analysis and change impact analysis work done on a system or service level. We concentrate more on presenting related work at this level as this is the focus of our work.

3.1 Source Code Level

3.1.1 Root Cause Analysis

On the source code level, root cause analysis has been commonly known as debugging or fault localization. When a piece of software fails, the developer usually uses some form of debugging to identify the part of the code causing the problem. Different debugging techniques have been proposed to provide faster methods of isolating the fault. For example, Zeller [38] uses delta debugging to isolate the variables and values relevant to a failure. This process helps provide a cause-and-effect chain leading to the observed erroneous behavior of the program. To be able to

isolate the possible causes, an automated test, two comparable program runs and access to the state of an executable program are required. Similarly, Renieris and Reiss [30] provide fault localization by comparing faulty runs to a number of correct runs to identify suspicious parts of the program that might be causing the failure. However, their presented technique does not indicate which of the identified parts might be more faulty than others.

From the perspective of mining software repositories, Ostrand and Weyuker [26] mine information from defect tracking databases to predict fault-prone files. Nagappan et al. [22] predict defects by constructing dependency graphs from the software artifacts and using network analysis on these graphs. Zimmermann et al. [41] use data from repositories such as CVS and Bugzilla to map bugs in the Eclipse project to their responsible source code locations. This mapping is similar to mapping the failure of an entity to its root cause (the incorrect source code in this case).

3.1.2 Change Impact Analysis

During the development and testing phases, many changes are done to the source code. Since not every developer in the team has to be aware of the details of other classes or components in the system, it is essential to identify the effects of a change to avoid affecting other parts of the system that might depend on it. Work done towards identifying other parts of the code that may potentially be affected by a change includes that done by Badri et al. [4]. This work uses a control call graph based technique to identify which methods might be affected by a change. A control call graph shows the sequence in which calls get executed. It clarifies which calls are conditional and which are not. Depending on where a change is going to be made, the control call graphs can be used to trace other places in the code that may be affected by this change.

Ying et al. [37] mine the change history from source code versioning systems such as CVS repositories to predict related files such that when a developer changes one file, a set of related files to consider can be recommended. This is done through identifying changed files that were checked in during the same time frame and with the same check in comment. The algorithm then extracts change patterns formed from sets of changes that occur together frequently enough to form a pattern.

Similarly, Hassan et al. [15] also use data from CVS along with four main heuristics to predict change sets, historical co-changes being one of them. They use pruning techniques to improve the recall and precision values while combining

their proposed heuristics in different ways. Without pruning, the heuristics produced high recall values, but very low precision values. With a hybrid technique that combines historical co-changes with file structure information and pruning, an average recall of 0.51 and an average precision of 0.49 was achieved.

On the same note, Zimmermann et al. [42] present their tool, ROSE, which guides programmers during the change process of updating source code by suggesting other program entities that previous programmers have changed in the same situation. This is done by mining the CVS repository to discover which program entities were changed together. Rules are then produced based on the entities that change together frequently, and are assigned a *support* and *confidence* level described in [40]. The authors looked at eight different systems including Eclipse, GCC, JEdit, and KOffice. Their results show that for stable systems like GCC, ROSE obtained a precision of 0.44 and a recall of 0.28. Other probabilistic techniques to model historical co-changes include using Bayesian Belief Networks [27] such as the work done by Mirarab et al. [21]. However, in our work, we chose to use the support and confidence measures defined by Zimmermann et al. [40] in our change impact analysis process. Section 6.4 explains these measures in detail.

In related work, Canfora et al. [7] predict change sets by mining textual descriptions of change requests rather than considering entities that changed at the same time only. That is, when a change request is received, related change requests are retrieved based on the similarity of the textual descriptions of both reports, and accordingly, the changed files in the retrieved reports are recommended according to the rankings.

This type of work is relevant because the same ideas are applicable on a service or system level. Instead of recording changes that have been done to a source code file, a CMDB records changes that are done to the configuration of CIs. By looking at change reports, and seeing which CIs changed together often, we can recommend a set of CIs to check when changing a certain CI just as a set of files is recommended when one file is changed.

3.2 Network Level

3.2.1 Root Cause Analysis

Decision trees have often been used in failure diagnosis to locate the faults in a network. Chen et al. [39] first train a decision tree to identify failed and successful

requests in a network through request logs, and then post-process the tree to remove irrelevant paths according to a set of heuristics. They apply their technique to the eBay network for several months, and obtain promising results. This work, however, concentrates on Internet sites, and would be hard to apply to large systems of different nature. In their work, the authors assume the presence of the initial tree, and do not explore the problem of acquiring this data. However, obtaining different possible paths for the different events in a system is not an easy task.

Lewis presents a case-based approach to managing faults in communication networks [19]. His work assumes that all problems in a network have filed trouble tickets associated with them. Trouble tickets contain fields such as the alarm that went off, the date and time of the problem, the problem device, the kind of trouble, history of the problem, possible cause, and finally, the resolution to the problem. When a new trouble ticket comes in, related trouble tickets will be retrieved from the trouble ticket repository, and the closest trouble ticket would be used. The closest trouble ticket is based on the number of matching fields between the two trouble tickets. The strategy or solution of the chosen trouble ticket is applied. If it does not work, then the strategy of the next matching ticket will be applied and so on.

3.2.2 Change Impact Analysis

Hariri et al. [14] developed a framework for identifying the impacts of faults and attacks in large scale networks. Although the studied feature here is not a planned change, it still follows the same idea of proactively determining the impact of some event. In this case, the event is an anticipated attack on the network such as a denial of service attack, for example. The developed framework identifies critical parts of the system whose failure would severely impact all the system's behavior due to this attack.

3.3 Service/System Level

Service Management is gaining the interest of many researchers as it is a current problem faced by most companies. IT managers need tools to help them manage their services and to ensure they meet their specified Service Level Agreement (SLA). Researchers have been trying to devise with practical solutions that help IT

personnel quickly diagnose problems (root cause analysis) and proactively discover the impact of a change before applying it (change impact analysis).

3.3.1 Root Cause Analysis

When incidents or failures are reported, they are usually reported in terms of a service not working properly such as the email service being too slow, for example. To resolve the incident, a mapping of this high level observation to the resource causing the problem should be made. Hanemann [12] developed a framework for fault diagnosis based on service related events. The main idea behind this work is that concepts from the fault diagnosis literature should be adapted to work with service oriented fault localization. In most IT systems, incidents are received at the level of the services in the form of performance degradations, SLA breaches, or total failures. Hanemann proposes an event correlation framework to correlate between events received at the service level and those received at the resource level in order to identify the root cause of any problem with a service.

Hanemann [11] proposes a hybrid approach for resolving service failures which combines rule-based and case-based reasoning. Rules which map service events to their possible root causes on the resource level will be stored in a rule database. When an incident is received, the appropriate rule is chosen from the database. If no such rule exists, then control is turned to the case-based reasoner which tries to match this event to previous cases. If no match is found (i.e this is the first time such an incident is received), the root cause of the problem will be manually identified and then stored as a case in the case database. Eventually, the rule-based reasoner is updated to deal with such cases after they have been resolved by the case-based reasoner. The downside to this approach is that it is difficult to identify all the needed rules, and manually resolving each case might be very costly. Additionally, having static rules that do not examine the actual events that took place in the system can often lead to identifying the wrong root cause.

Wang et al. [35] propose a knowledge engineering framework for service management based on a Bayesian Network. Their proposed framework is based on data from operation management tools, event log files, CMDB, and other resource information. They have based their work on IBM Change and Configuration Management Database (CCMDB) which had more than 1000 kinds of CIs and 86 kinds of relationships. They used expert knowledge to build the required Bayesian Network based on the type of nodes and the relationships between them.

To use the model, the technician would enter the observations that can be made about the system, and the network would produce a list of ranked CIs according to their probability of being true. This is done by calculating the probability of each node in the graph being true considering the observations made. Accordingly, the technician could know which CIs to check to solve the current problem. Their work shows that the top two ranked CIs are usually enough to use to solve the problem. Although the authors clarify how they would identify the causality edges in the Bayesian Network (through templates produced by experts), they do not clarify how they extract the prior and conditional probabilities for the network. They mention that they will use history records and expert knowledge, but they do not explain how exactly would that be done. In our work, we also use more information, such as temporal constraints to produce more accurate root cause analysis.

Log files have also been an important source of information for discovering errors or violation of system requirements. Wang et al. [36] use logs to monitor the behavior of a system. They present a solution based on propositional satisfiability (SAT) to diagnose software requirements. The first step is to construct a goal model to represent the correct behavior of the system. This correct behavior is defined by the requirements the system should fulfill, and each main goal would be divided into subgoals. The system is then instrumented at the parts that show the different monitored goals in the constructed model. Therefore, log files would show whether these goals were accomplished successfully or if there were any problems. The goal model is then turned into a SAT formula. The values for the variables in the SAT formula are obtained from examining the log files which show if a goal has been achieved (i.e., has a true value) or not (i.e., has a false value). The different combinations of diagnoses for the wrong behavior of a system are predefined, and the SAT solver is repeatedly invoked to find an assignment that corresponds to all possible diagnoses.

Wang et al. [36] are the first to propose a SAT-based solution. However, there are two main drawbacks to its practicality. For example, the source code has to be instrumented in order to get the needed information into the log files. This is a big overhead especially if there is no access to the source code, and service level management should not have to go down to the source code level. Additionally, identifying all the combinations of events that violate a requirement is very challenging. In practice, many problems come from unforeseen circumstances. This solution would not identify these scenarios. Similarly, Razavi and Kontogiannis [28] use log analysis as well to determine if a specific undesired scenario has occurred. They use threat tree models to represent an undesired or threatening pattern, and

then use pattern matching to identify whether the events in the system map to this pattern or not. Again, it is very challenging to identify all variations of possible threats in order to model them in a threat tree. Other work analyzing logs includes that by Reidemeister et al. [29] who use machine learning techniques to recognize symptoms of recurrent faults from log data.

Since static information is not always enough, considering temporal issues is one of the challenges we address in our work. In [33], Tawfik and Neufeld discuss various techniques for considering temporal information while modeling causality. They augment regular Bayesian Networks with temporal information by adding time to the Conditional Probability Table (CPT). Instead of having a CPT for each random variable (RV) in the graph, they represent the conditional probabilities as continuous functions with respect to time. This allows the model to answer the question of what is happening at time t. More precisely, it allows them to find the probability of X happening at time t given event Y. This method, however, requires the network to keep track of each node's value over time and the time at which any particular observation is made which may not be very practical [6].

Temporal issues provide many challenges especially in dynamically changing networks that do not have a static infrastructure. Natu and Sethi [23] provide a way to deal with this dynamic aspect of the networks in order to determine the root cause of problems. Their model stores relationships between faults and their possible symptoms. This is done through a matrix where rows represent observable symptoms and columns represent the possible network faults. If a symptom is correlated with a fault, an entry in the matrix mapping this fault to this symptom would be present. This entry would be a probability value indicating the strength of the relationship.

The algorithm used to identify root causes analyzes symptoms one at a time. A hypothesis is drawn with the first symptom and then updated with each upcoming symptom until a final hypothesis is reached. To incorporate temporal information, the model is updated at regular increments of time, and is time stamped with the time it was formed. Additionally, the symptoms are also time stamped. During analysis the time of the symptom is compared with that at which the model was constructed. The difference between the symptom time and the time at which the model was constructed affects the confidence of the derived hypothesis. The closer the two times, the more confidence there is in the hypothesis. Therefore, each time stamp will have a dependency matrix associated with it. Hypothesis are also stored and time-stamped to be used in later calculations. Old hypothesis or matrices are removed once the information becomes irrelevant because it is very outdated.

Our root cause analysis model also uses a matrix to store the probabilities tied to edges representing causal implications between problems and their possible reasons. Natu and Sethi refer to them as faults and symptoms. The first difference is that in our matrix, the rows and columns are the same set of CIs as any CI may have an incident and any CI could be a possible root cause. We do not divide them into separate sets of faults and symptoms.

Additionally, the main issues their work addresses differs from ours. They address wireless ad-hoc systems where nodes are continually changing. In our work, the nodes themselves are fairly stable and so this is not a significant concern for us.However,the change in the edges between the nodes is a concern for us i.e., if they exist at certain times or not. A common concern both techniques have, however, is the time of analysis, and thus the best model (stored matrix) is used according to the time of the incident.

3.3.2 Change Impact Analysis

At the system level, where services are involved, change impact analysis must identify the effect of a change of any component or service in the system on other components as well as other services in the system. Kumar et al. [18] present an enterprise ontology that overlays the data in a CMDB which helps in identifying and quantifying the impact of changes. This work attempts to account for the dynamic aspect of IT systems as well as quantifying the change impact. Additionally, they also incorporate impact of changes on service provision; this has become an important feature of modern Enterprise IT Management (EITM). The model assumes four types of entities that are mapped on top of CMDB CIs. These are: *infrastructure components*, which are physical resources; *service components*, which run on the infrastructure components; *process transactions*, which are abstractions that represent end-to-end business processes executed by the service components; and *business goals*, which are delivered through the process transactions. The authors define a set of attributes for each entity. These attributes and the relationship between the different components allows the quantification of the requirements of a component to run correctly in terms of the other components. Four use cases of a medical example are presented to show how the impact of a change can be determined based on their defined attributes.

Similarly, Boer et al. [10] propose an ontology of enterprise architectures in order to analyze the impact of changes in a system. By defining relationships in an ontology, and using their semantic meanings to determine the effect of a change,

impact analysis can be performed on enterprise systems. They use ArchiMate [1], an enterprise architecture modeling language, as the basis to define heuristic rules for determining the effects of different types of changes based on the semantic meanings of the relationships. ArchiMate consists of structural and behavioral concepts that are used to model a system. Boer et al. use some of these concepts such as role, component, data model, trigger, process and service. The main relationships used to relate concepts are use, assign, realize, access, and trigger.

Heuristic rules are then defined for each of these relationships. For example, the *access* relationship that models a behavioral concept A accessing a data object B. The heuristics defined include that if A is deleted, B is not affected, while a modification of A might involve a new way of accessing B. Another heuristic based on the *use* relationship is that if B uses A, then a modification in A might involve a need to modify B so that it can still use the same functionalities it depends on in A. Based on such heuristics, the ripple effect of a change is calculated by calculating the immediate impact of a change from one concept to other directly related concepts, and then repeatedly finding the effect on the newly discovered impact to directly related concepts and so on until there are no more new impacts discovered.

It is important to distinguish between failure impact analysis and change impact analysis. For example, [13] presents a framework for failure impact analysis to determine the impact of a failure in order to determine the best course of action to be taken. This is dependent on the degree of violation to the service level agreements in place. Change impact analysis, on the other hand, concentrates on predicting the impact of a change before implementing it.

The above presented related work shows different approaches for change impact analysis on the system or service level. However, there is no work, to the best of our knowledge, that applies data mining techniques on a CMDB and uses historical data to predict change propagation on the system level.

3.4 Summary

This chapter presented related research in the fields of root cause analysis and change impact analysis. Work in these fields is done at three different levels: source code level, network level, system and service level. On the source code level, version control systems such as CVS and bug tracking systems such as Bugzilla have been mined to extract fault prone files and co-changes of source code files. On the

network level, fault trees have been used to identify the causes of problems in a network. More related research has been provided at the service level as this is the focus of this thesis. On the service level, root cause analysis has been done through log analysis, Bayesian networks as well as rule-based and case-based reasoning of previous faults. Change impact analysis has been done by defining ontologies for the relationships between system components. Our work combines different features of root cause analysis and change impact analysis into one framework, and augments the analysis with temporal information which allows for a more dynamic analysis.

Chapter 4

Overview of DRACA Framework

In order to define the requirements needed for the Decision Support for Root cause Analysis and Change impact Analysis (DRACA) framework, and to identify possible sources of information, we held interviews and discussions with CA analysts and CA customers. This chapter presents the important points that arose from our discussions which lay the foundations of the features presented by DRACA. Section 4.1 presents these discussions. Section 4.2 then gives a black-box overview of DRACA's two processes.

4.1 Field Work

We were able to contact three CA customers who use a CMDB. Two of these customers use CA's CMDB, while the third was using an in-house CMDB and considering buying CA's CMDB. Additionally, we also conducted several discussions with CA's Global Information Systems (GIS) team who manages CA's IT systems. To do so, they use CA's CMDB and other CA tools as well.

The goal of our discussions was to better understand the needs associated with managing a large IT system providing different services to its clients. We wanted to understand why a CMDB is useful to them, and how they use it. Additionally, we looked for deficiencies in the functionalities offered by the CMDB which we can ameliorate in our framework.

Most of our interviewees saw many benefits to the CMDB in terms of organizing the information about their system, and keeping track of their CIs. They all used the CMDB for root cause analysis and change impact analysis. From our discussions, we got real life examples of changes such as modifying the configuration of a

firewall which sometimes causes some applications to fail if any dependencies were missed. Another example given was that an application depended on two databases one running Oracle and the other running SQL Server. If the Oracle database was upgraded, for example, then they must make sure that the new version of Oracle is still compatible with the SQL Server.

Given an initial CI to change, analysts commonly follow the relationship edges stored in the CMDB to determine a list of other CIs that may be impacted. However, apart from getting a list of the CIs that might be impacted, no additional information — such as ranking or some quantification of this impact — is provided to them. Our interviewees indicated that they would like to get more details about what the impact of a change is, and which CIs are more likely to be impacted than others.

When asked how they perform root cause analysis, those using CA's CMDB indicated that they look at the dependency edges to find all related CIs, and then check the status of each. They indicated that it would be useful to have a more automated way of discovering the root cause of a problem. From the manual steps indicated by the other customer using its home-grown CMDB, we were able to identify some key points that need to be included in root cause analysis. When a problem occurs, IT analysts go back and check any change or event that occurred in the last 48 hours to narrow down their suspect CIs. We consider this point in our automated root cause analysis since events or changes that are closer in time to the time of a problem are more likely to be its cause.

Another significant point that arose in our discussions is that of temporal constraints on the relationships in an IT system. That is, some relationships only exist at certain times. These are usually scheduled events such as build processes or backup processes that only occur at specified times in the calendar of a system. The fact that these relationships do not exist at all times should be taken into consideration while performing both root cause analysis and change impact analysis. All of the points raised in our discussions were taken into consideration when designing DRACA.

4.2 DRACA: The Big Picture

The DRACA framework consists of two main processes: root cause analysis and change impact analysis. Figure 4.1 shows the overall structure of DRACA from a black-box perspective. The root cause analysis process constructs its model from

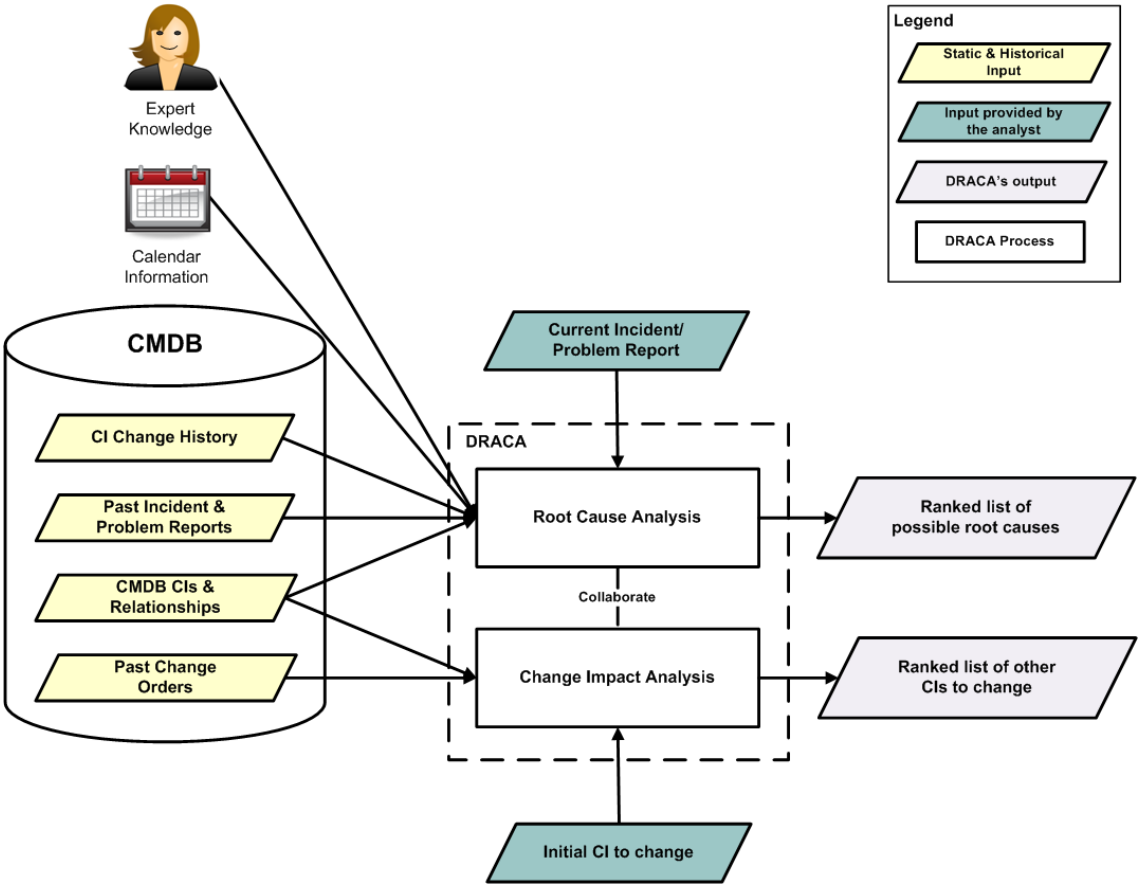

Figure 4.1: Overview of the DRACA Framework

the CIs and their relationships, incident and problem reports, CI change history, and calendar information. Chapter 5 gives the details of how this information is used. When an incident or problem is encountered, the incident or problem report is provided to DRACA such that DRACA knows the affected CI as well as the time of the incident. Using this information and its stored model, DRACA provides a ranked list of suspect root causes of this incident.

On the other hand, the change impact analysis process constructs its model from historical co-changes by analyzing previous change orders. It also needs to identify the CIs in the system, and thus uses the stored CI information. When an analyst decides she wants to change a specific CI, she provides this CI to DRACA. DRACA then produces a ranked list of other CIs that should be considered for inclusion in the analyst's current change set. Chapter 6 provides the details of this process.

Table 4.1 compares the root cause analysis process and the change impact analysis process. For root cause analysis, the input is the CI for which the incident is

	Root Cause Analysis	**Change Impact Analysis**
Input	Problematic CI/Observed Incident.	Proposed Change to a CI.
Output	Ranked list of suspect CIs (least number of possible CIs). Must be *accurate*.	Ranked list of possibly affected CIs to include in change set. Must be *thorough*.
Considered Dependencies	Look at what incident CI depends on.	Look at historical co-changes
Ranking	Rank results according to a suspect rate (probability), usually based on history.	Rank results according to *confidence* based on history
Learning	Learning from past incidents and previous root cause analysis results.	Learning from past change orders and previous change impact analysis results.
Time	Have to consider time of incident and time of previous events.	Have to consider time of analysis and the time of previous change sets.

Table 4.1: A comparison between Root Cause Analysis and Change Impact Analysis

observed; for change impact analysis, the input is the CI the analyst would like to change. The output of both processes is a ranked list of CIs. For root cause analysis, this is the list of possible suspect root causes while for change impact analysis, this list is the set of CIs that may also need to be included in the analyst's change set for the change to be complete. Root cause analysis considers the dependencies stored in the CMDB in order to determine the relation between CIs while change impact analysis looks at the historical co-changes of CIs in order to determine the relation between them. In both root cause analysis and change impact analysis, the ranking is done according to a probabilistic measure although how this measure is estimated differs between both processes. Chapters 5 and Chapter 6 discuss how these probabilities are mined in more details. Additionally, both processes have a learning aspect to them since new change orders or incident/problem reports get stored back into the CMDB. Accordingly, when the model is updated, this new knowledge will be incorporated in it. Finally, temporal information should be considered in both processes. In the root cause analysis process, the time of the incident as well as the calendar information of the CI relationships is considered. In change impact analysis, older co-changes have less potential relevance to future changes, which is reflected through exponential forgetting.

4.3 Summary

In this chapter, we have presented an overview of the DRACA framework in terms of the inputs and outputs of both the root cause analysis process and the change impact analysis process. The field work performed to extract the functionalities needed in each process was also presented in this chapter. Through the discussions conducted, we identified the key required functionalities which in turn specified the information needed to provide them. The next two chapters discuss each of the involved processes in more details to explain how this information is processed and modeled.

Chapter 5

DRACA's Root Cause Analysis Process

This chapter presents the root cause analysis process and the underlying root cause analysis model included in DRACA. Before going into the details of the process, and its underlying model, we first define the sources of data needed for this model. Although we collect this data from a CMDB, our model does not depend on any specific format, which makes it general enough to be used with any IT system. The needed information is presented in Section 5.1.

The root cause analysis part of DRACA consists of three steps that are shown in Figure 5.1. The remaining sections of this chapter explain each of these steps in details. Section 5.2 presents Step 1 which produces a root cause matrix based on the CIs and their relationships in the CMDB and a defined relationship mapping scheme. Section 5.3 then discusses Step 2 which examines the calendar information of the system and correlates it with the time of the current incident to produce a root cause matrix that matches this time. Section 5.4 presents the last step, Step 3, which considers the change history of CIs to weigh each one according to its last time of change. Changes closer to the time of the incident will get higher weights. This step produces the current weighted root cause matrix which shows the probability of each CI being a root cause of the current incident. Based on these probabilities, a ranked list of suspect CIs can be produced. Finally, Section 5.5 discusses some of the accuracy and effectiveness of DRACA's root cause analysis process.

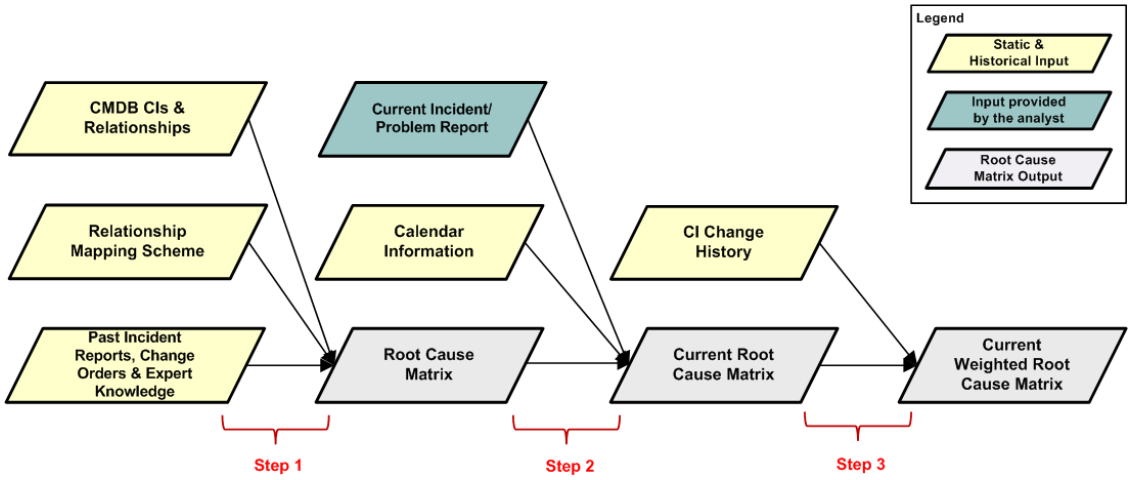

Figure 5.1: DRACA's Root Cause Analysis Process

5.1 Data Needed for Root Cause Analysis

In order to populate the causality model we use for root cause analysis, the following data should be available:

- **CIs and Relationships:** A record of which CIs are in the system, and the relationships between them is. A list of all possible types of relationships is also needed to be able to produce the relationship mapping scheme.
- **Incident and Problem Reports:** Reports must record the affected CI as well as the cause of the incident or problem. Additionally, these reports must be time-stamped.
- **Change history of CIs:** Any changes to a CI or events associated with it must be tracked.
- **Calendar Information:** All scheduled tasks must be recorded to be able to determine what CIs and relationships are active at any point in time.

5.2 Step 1: Producing the Root Cause Matrix

The root cause matrix is produced in three sub-steps. Step 1a produces the causality graph by mapping the CIs in the CMDB into nodes, and mapping the CMDB relationships into causality edges according to a defined mapping scheme. Step 1b attaches probabilities to the causality edges by mining incident reports, change

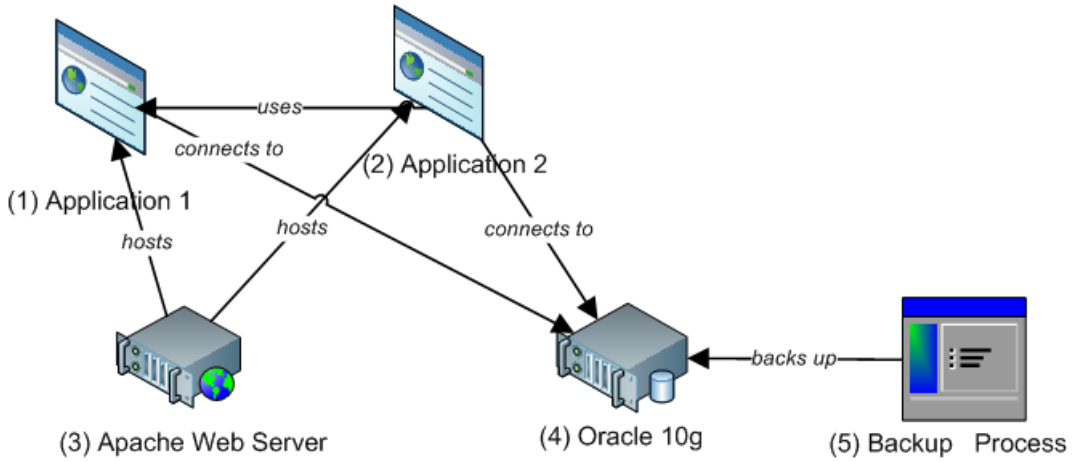

Figure 5.2: Example IT System in a CMDB

reports, and expert knowledge. Step 1c stores these probabilities in a causality matrix K and calculates the probability that each CI could be the root cause of another CI. This probability will be stored in the root cause matrix R.

Step 1a: Mapping CMDB Relationships into Causality Edges

The data available in a CMDB records how different CIs interact together. Figure 5.2 shows a small example system stored in a CMDB. The CIs in the graph have been numbered in order to simplify reference to them in the rest of this chapter. Based on these relationship meanings, we develop a mapping into causality edges. For example, in Figure 5.2, Application 2 connects to the Oracle database, so if the Oracle database has any problem, Application 2 will malfunction because it can no longer fetch data from the database correctly. In other words, a problem in Oracle can cause a problem in Application 2. In a causality graph, this would map to an edge going from Oracle (node 4) to Application 2 (node 2). To achieve this, we develop a mapping scheme to convert the different relationships in the CMDB into causality relationships. Table 5.1 shows the scheme we use to map the CMDB in Figure 5.2 to the causality graph shown in Figure 5.3.

In a real system, there would be more relationships defined in a CMDB. Additionally, as time passes, CMDB managers may add new types of relationships to meet their needs. In such a case, they should enhance the mapping scheme to include the new relationships.

CMDB Relationship	Causality Mapping
A hosts B	A causes B
A connects to B	B causes A
A backs up B	A causes B
A uses B	B causes A

Table 5.1: CMDB Relationship Mapping Scheme

Step 1b: Estimating the Probabilities

Given the causality edges calculated in step 1a, step 1b estimates the probabilities on these edges. Estimating the strength of the causality edges, which we represent as probabilities, is one of the most challenging parts of root cause analysis. This is done by mining the incident reports that were prepared for previous incidents. The reports will contain information about the root cause of the incident, and how it was solved. Additionally, some incident reports may contain information about other related failures or symptoms that occurred concurrently with this incident. This is useful in estimating the probability on the edges. Simple counting techniques such as how many times CI x had a reported incident (count(x)) and how many times CI y was the cause of this incident (count($y \rightarrow x$)) can give us an estimate for the strength of the relation between y and x in the form of count($y \rightarrow x$)/count(x).

Change reports are another source of relevant information. When a change is to be implemented, related CIs that need to be checked and other changes that need to be implemented as a result are usually documented. The same counting technique can be used to estimate the strength of the relationship between two CIs. If two CIs commonly change together, then there is a greater chance that these CIs are related and that a fault in one can cause a failure in the other. In DRACA, this information is stored in the change impact analysis model allowing the root cause analysis process to leverage it.

Finally, expert knowledge is an important source of information. If the above information is not well documented, experts who have been working with the system for years could provide an estimate for it as they have extensive knowledge about the things that go wrong, and their most likely root causes. We use these sources of information to estimate the probabilities shown in Figure 5.3. Ideally, incident reports and problem reports would be used to estimate the probabilities, and experts would verify this information or provide additional information when proper documentation is missing. However, the amount of reliance on expert knowledge

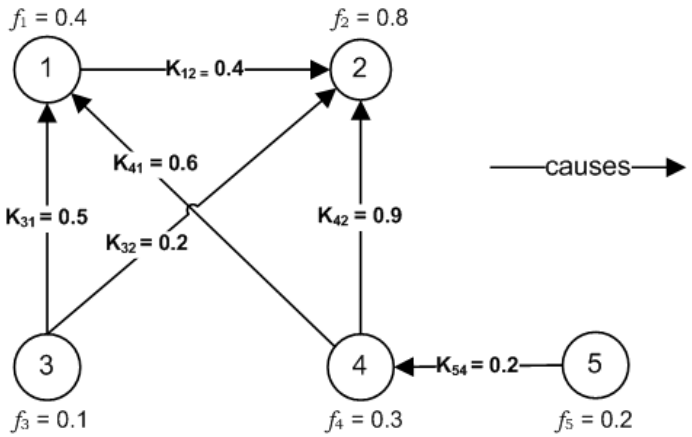

Figure 5.3: Example Causality Graph Corresponding to Figure 5.2

greatly depends on the quality of historical information present.

Apart from estimating the strength of causality edges, we must also estimate how prone a CI is to having a fault. For example, some servers are more prone to failure than others. This could be because the hardware used is more faulty for example. On the other hand, some applications may fail more often than others because they have not been vigorously tested. We, therefore, need a measure of how prone a CI is to having a fault of its own accord. That is, without being a consequence of a fault in another CI. We call this measure *fault proneness*. We define the fault proneness f_i of a node i as the probability that i has a fault of its own accord. For example, $f_1 = 0.4$ means that there is a 40% chance that node 1 might incur a fault based on its behavior in the past. f_i is also mined out of historical incident and problem reports by showing the percentage of time a CI had an incident out of the total number of incidents reported in the system.

Figure 5.3 shows the corresponding causality graph for the example in Figure 5.2. It shows the probability on the causality edges as well as the fault proneness of every CI. For example, in Figure 5.3 the probability on the edge from node 1 to node 2, $K_{12} = 0.4$ shows that if there is a fault or failure in node 1, there is a 40% chance that it will cause a failure in node 2. We assume that the propagation of errors along a node's outgoing edges are independent events, and are not mutually exclusive. This means that an error in one node can simultaneously propagate along more than one edge that leaves from that node. Accordingly, the probabilities on the edges coming out of a node do not have to add up to 1.0. For example, probabilities on the edges propagating out from node 4 (0.6 and 0.9) add up to 1.5, and not to 1.0.

Step 1c: Calculating Root Cause

Steps 1a and 1b produce the causality graph that we use to produce the root cause matrix. In order to do this, we first store this causality graph in a matrix K. Given a causality graph with n nodes, the edges in the graph which represent a direct causal relation between two nodes will be stored in the n by n matrix K. The probability that CI i might directly cause a problem in CI j is found in the entry K_{ij}. The following causality matrix K stores the probabilities shown in the graph in Figure 5.3.

$$K = \begin{pmatrix} 0.0 & \mathbf{0.4} & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ \mathbf{0.5} & \mathbf{0.2} & 0.0 & 0.0 & 0.0 \\ \mathbf{0.6} & \mathbf{0.9} & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & \mathbf{0.2} & 0.0 \end{pmatrix}$$

Representing this information in a matrix provides a convenient way for calculating indirect cause propagation. An entry K_{ij} is the probability of an error propagating from i to j through a direct edge (i.e a path of length one). Now, consider K^2, which is the square of K, i.e., K multiplied by K. Entry K^2_{ij} is the probability of propagating from i to j through any path of length two. In order to calculate this probability, we must consider all paths of length two from i to j and assume we take any, some or all of them. This probability is calculated by finding the probability of taking any of the possible two-step paths from i to j according to the following probability rule where A and B are independent events:

$$P(A \cup B) = P(A) + P(B) - P(A) * P(B) \tag{5.1}$$

We use this probability rule when multiplying the matrices and when adding them to compute the transitive closure matrix.

Let us consider the graph in Figure 5.3, and consider the ways a problem in node 5 can propagate to affect node 2. To start with, $K_{52} = 0$ since there is no path of length one from node 5 to 2. Now, to calculate K^2_{52}, we need to consider going from node 5 to node 2 in any path of length two. Only one such path exists in our graph: $5 \rightarrow 4 \rightarrow 2$. Therefore, K^2_{52} is calculated as follows:

$$\begin{aligned} K^2_{52} &= K_{54} * K_{42} \\ &= 0.2 * 0.9 \\ &= 0.18 \end{aligned}$$

Similarly, K^3_{52} is calculated through the path $5 \rightarrow 4 \rightarrow 1 \rightarrow 2$ as follows:

$$\begin{aligned} K^3_{52} &= K_{54} * K_{41} * K_{12} \\ &= 0.2 * 0.6 * 0.4 \\ &= 0.048 \end{aligned}$$

When we calculate the probability of taking any path leading from i to j in one step (K_{ij}), or two steps (K^2_{ij}), or three steps (K^3_{ij}), or ... , or ∞ steps (K^∞_{ij}), we will get the overall probability that an error or failure in i could propagate to j through a path of any length. In other words, we are computing the transitive closure of matrix K which we shall call T:

$$T_{ij} = \cup_{l=1}^{\infty} K^l_{ij} \tag{5.2}$$

For simplicity and illustration purposes, let us assume we will calculate the transitive closure matrix T from K, K^2, and K^3 only. Accordingly, to calculate entry T_{52} for example, we would do the following:

$$\begin{aligned} T_{52} &= K_{52} \cup K^2_{52} \cup K^3_{52} \\ &= 0 \cup 0.18 \cup 0.048 \\ &= (0 + 0.18 - 0 * 0.18) \cup 0.048 \\ &= 0.18 + 0.048 - 0.18 * 0.048 \\ &= 0.21936 \\ &\approx 0.22 \end{aligned}$$

T_{ij} gives the probability that an error or failure in i causes a failure in j through a path of any length. Following equation 5.2, the matrix T below, rounded to two decimal places for presentation purposes, is the transitive closure of the given matrix

K. For example, $T_{52} = 0.22$ means that if node 5 has a failure, there is a 22% chance that this failure will propagate to node 2.

$$T = \begin{pmatrix} 0.0 & \mathbf{0.4} & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ \mathbf{0.5} & \mathbf{0.36} & 0.0 & 0.0 & 0.0 \\ \mathbf{0.6} & \mathbf{0.92} & 0.0 & 0.0 & 0.0 \\ \mathbf{0.12} & \mathbf{0.22} & 0.0 & \mathbf{0.2} & 0.0 \end{pmatrix}$$

Using the fault proneness, f_i, and the transitive closure, T_{ij}, we can compute R_{ij}, the probability that node i could be the root cause of a failure in node j, as follows:

$$R_{ij} = f_i T_{ij} \tag{5.3}$$

This calculates the probability that node i had a fault that propagated, directly or indirectly, to cause a failure in j. Therefore, the term R_{ij} gives us the probability that a failure in j has i as its root cause. The fault proneness of each node in Figure 5.3 is shown beside it. These values can be represented in a vector, f as follows:

$$f = \begin{pmatrix} 0.4 \\ 0.8 \\ 0.1 \\ 0.3 \\ 0.2 \end{pmatrix}$$

According to Equation 5.3, R would have the following probabilities, rounded to two decimal places for presentation purposes:

$$R = \begin{pmatrix} 0.0 & \mathbf{0.16} & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ \mathbf{0.05} & \mathbf{0.04} & 0.0 & 0.0 & 0.0 \\ \mathbf{0.18} & \mathbf{0.28} & 0.0 & 0.0 & 0.0 \\ \mathbf{0.02} & \mathbf{0.04} & 0.0 & \mathbf{0.04} & 0.0 \end{pmatrix}$$

From the above matrix, $R_{42} = 0.28$ means there is a 28% chance that node 4 is the root cause of an incident in node 2.

5.3 Step 2: Producing the Current Root Cause Matrix

In step 1, we produced a root cause matrix based on the relations in the CMDB and other information such as incident reports. However, one such static view of the system may not always yield accurate results if certain aspects change from one point in time to another. Therefore, we take the calendar information of the system into consideration while performing root cause analysis. For example, if a backup process runs every night at 8:00 pm to backup a database server, then the relationship between the backup process and the database server only exists at that time. Such information should be considered to produce more accurate analysis.

Depending on the time of the incident being examined, we will adjust our root cause matrix according to the calendar information available. Using the example in Figure 5.3, if the backup process (node 5) only takes place on Wednesdays from 7:00 - 8:00 pm, then the edge from node 5 to node 4 will only exist at that time. Accordingly, if we receive an incident on Sunday at 2:00 pm, we look at our calendar information and check if there are any causality edges we can exclude to improve the analysis. In this case, we can exclude the edge from node 5 to node 4 since this relationship does not exist on Sunday at 2:00 pm.

After examining the calendar information, we exclude irrelevant edges according to the time of the incident which means that our causality matrix K will change. We then calculate a new current root cause matrix according to the new edges being considered. To calculate the current root cause matrix, we follow the calculations done in step 1c. For example, given the scenario mentioned in the previous paragraph, the current root cause matrix would be as follows:

$$R_{current} = \begin{pmatrix} 0.0 & \mathbf{0.16} & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ \mathbf{0.05} & \mathbf{0.04} & 0.0 & 0.0 & 0.0 \\ \mathbf{0.18} & \mathbf{0.28} & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

If no calendar information is available for the system, then we simply use the root cause matrix calculated in step 1 as our main matrix for all times. In practice, we can skip generating the root cause matrix in step 1c, and simply generate it as needed in step 2.

CI	Last Change	Weight
1	January 1, 2009 10:00 am	0.00001
3	May 4, 2009 10:00 am	0.94
4	April 30, 2009 2:00 pm	0.09

Table 5.2: Temporal Weights

5.4 Step 3: Producing the Current Weighted Root Cause Matrix

After calculating the current root cause matrix in step 2 by considering the time of the incident and the available calendar information, we still need to consider the events and changes that actually happened in the system before the incident. This follows from the idea that if X causes some incident in Y, X has to occur before Y. Additionally, the fault or failure in X should occur at a time that is reasonably close to the time of the incident in Y. For example, it is more likely that a change or event that happened 2 hours before the incident occurred could be its root cause versus some other event that happened two days ago.

To do this, we compare the time of the incident and the time of the last change that occurred to related CIs such that a higher weight is given to CIs that had events or changes that took place closer to the time of the incident. A weighting scheme is needed to accomplish this. The choice of the weighting scheme does not affect the rest of the calculations we do. The main idea is to give decreasing weights to changes or events as they occur further back from the time of the incident. We choose to use an exponentially decaying weighting scheme from 1.0 to 0.0. The rate of decay, however, depends on the nature of the system. The exponential weighting scheme we use in this example is shown in Figure 5.4.

According to when the last time a CI changed or an event associated with it occurred, its row in the root cause matrix, R, would be multiplied by its temporal weight. Accordingly, CIs which have not changed in a long time will have a very low weight factor (almost 0) which will greatly decrease their probability to be a suspect root cause of the current incident.

Assume we have received an incident for CI 2 on Monday May 4, 2009 at 2:00 pm. Table 5.2 shows the different change times of the CIs, and their temporal weight according to the weighting scale shown in Figure 5.4. For example, since the last change in CI 4 happened on April 30, 2009 at 2 pm (i.e., 96 hours from

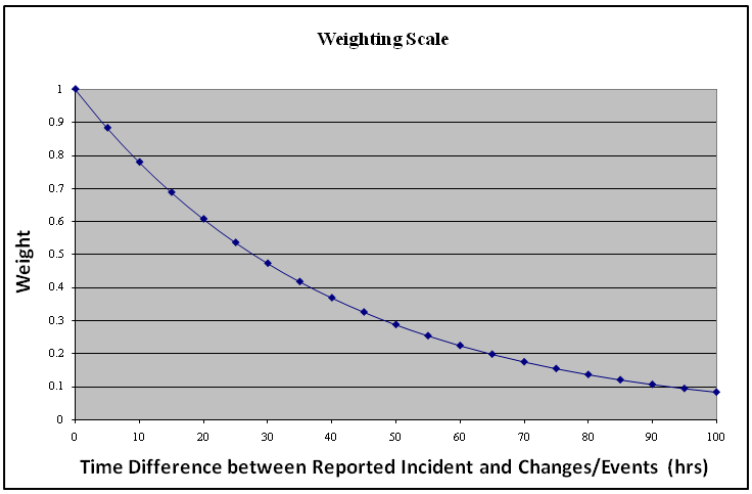

Figure 5.4: Weighting Scale

the time of the current incident), its weight will be 0.09 according to the graph in Figure 5.4.

We then multiply the temporal weight of each CI by its corresponding row in the root cause matrix R to get the new probabilities. Accordingly, the probabilities shown in $R_{current}$ in Step 2 (See page 39) will now change to those in $R_{weighted}$. These are shown below, rounded to three decimal places for presentation purposes. Note that since the incident occurred on a Monday, the edge from node 5 to node 4 is not considered as this edge only exists on Wednesdays from 7:00 - 8:00 pm as shown in Step 2. That is why all the entries in row 5 are zeros in $R_{weighted}$, and we do not need to worry about the changes done to CI 5.

$$R_{weighted} = \begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ \mathbf{0.047} & \mathbf{0.034} & 0.0 & 0.0 & 0.0 \\ \mathbf{0.016} & \mathbf{0.025} & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

Since the incident is in CI 2, we look at column 2 in the root cause matrix to determine the ranking of the suspect CIs. Without temporal information, our most likely suspect for the incident in CI 2 was CI 4 (with a 28% chance) followed by CI 1 (with a 16% chance) as shown in matrix $R_{current}$ from step 2 (See page 39). With temporal information, our primary suspect now becomes CI 3 (with a 3.4% chance) followed by CI 4 (with a 2.5% chance) as shown in matrix $R_{weighted}$ above. This makes sense since CI 3 had a change very close to the reported time of the

incident.

5.5 Discussion

As with any model or framework, the accuracy of the produced results is a function of the quality of the data available. The more detailed the information stored about incidents, their root causes, and side effects, the more accurate the estimated probabilities are likely to be, and thus the more accurate the root cause analysis will be. Since ITIL standards are becoming increasingly popular and are being adopted by organizations, we believe that in time more companies will adhere to these standards and have detailed documentation of their incidents. Additionally, as the need arises, the structure of the reports may eventually change to include more information that will be useful to future root cause and change impact analysis.

In order to have an effective prediction framework, learning from the results of previous analyses is crucial. When an incident is reported, and the framework reports a suspected root cause, this suspect CI should be fixed, and then the problem should be re-evaluated to determine if it has been resolved or not. Whether the problem was fixed or not, and accordingly, whether the identified root cause was accurate or not should be stored in the incident reports. The causality graph should be updated at regular intervals with the new information mined from these incident reports.

As incidents occur and are fixed, the totality of the picture may change, and the strength of the causality edges may change accordingly. The frequency with which the causality graph is updated depends on the nature of the system at hand. If the system being managed is a highly dynamic system with many incidents occurring, a higher frequency of updating is needed. On the other hand, if the rate of incidents is low, the initial causality graph will be more or less stable, and a lower frequency of updating is needed.

One of the challenges of using a causality graph is how to deal with cycles in the causality graph. Practically, a cycle implies that an error in a particular CI could cause a ripple of failures which loop around and cause another failure in the original CI, repeatedly. This implies that the loop can be traversed many times. Although we chose not to address loops yet, as they are not very common in real systems, and most work in the field has chosen to ignore them, our model does, however, allow for them. Using matrix multiplication to calculate the transitive closure can approximate the impact of loops.

5.6 Summary

In this chapter, we demonstrated how the data in the CMDB can be mined to produce a causality graph, and how this graph can be represented in a matrix. Through manipulations on this matrix, we produced a root cause matrix which records the probability that one CI is the root cause of an incident in another CI. This is done through three steps: forming the basic root cause matrix from the data in the CMDB, considering the time of the reported incident along with the system's calendar information to adjust the current root cause matrix accordingly, and finally using different weights for different CIs according to the last time they have changed to produce the final root cause matrix. The final root cause matrix effectively provides a ranked list of the suspect CIs for the current incident.

Chapter 6

DRACA's Change Impact Analysis Process

This chapter presents the second process supported by our framework: change impact analysis. It explains the collaborative process involved between the analyst and DRACA, and describes the model used in this process. We also present the empirical work done to evaluate our change impact analysis process.

Section 6.1 presents the sources of data needed to populate the presented model. Section 6.2 explains the change impact analysis process provided by DRACA. Section 6.3 explains how DRACA's performance will be evaluated. Section 6.4 explains the underlying model DRACA uses for suggesting CIs. Section 6.5 presents the empirical work performed to evaluate DRACA. It explains the structure of the data used, the experiment setup, as well as the results obtained from the base case of our experiment and those from the various filtering techniques we used.

6.1 Data Needed for Change Impact Analysis

If the analyst determines that CI x is in a new change set, our method essentially searches for previous change sets, stored in the CMDB, that contain x, and suggests that CIs in those sets (appropriately ranked) should be considered for inclusion in the new change set. Our model uses *support* and *confidence* measures to estimate how closely related nodes x and y are, based on how often they have appeared together in past change sets. To be able to calculate the support and confidence measures, we need to have the following information available:

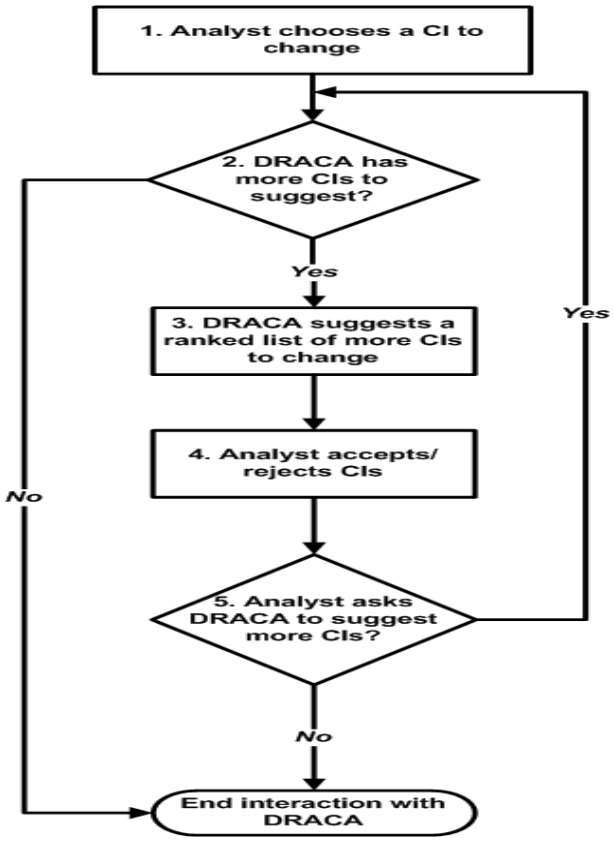

Figure 6.1: Flowchart of DRACA's Interactive Change Impact Analysis Process

- **Change Orders:** Change orders must record the change set involved (i.e., the set of CIs that were changed in this change order) as well as the date this change was implemented on.

Our current change impact analysis method relies only on historical co-change. Since historical co-change can be mined out of the change sets stored in change orders, we do not need additional sources of information for our current approach. In the future, as we use more information for our analysis, more data sources might need to be used.

6.2 DRACA's Change Impact Analysis Process

The change impact analysis process that we propose is a collaborative interaction between the analyst and the DRACA tool as illustrated in Figure 6.1. Figure 6.2 shows the Graphical User Interface (GUI) for our prototype supporting this process.

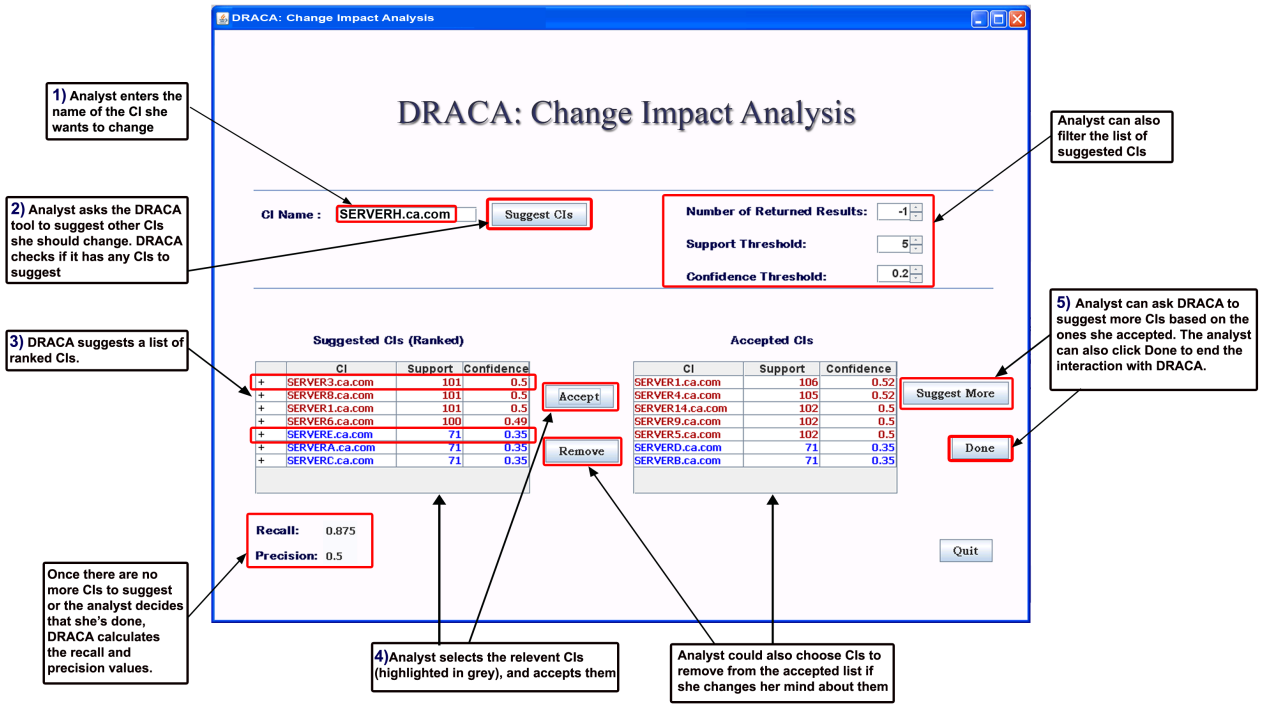

Figure 6.2: The DRACA Prototype Tool Suggests a List of Ranked CIs to the Analyst, and Allows her to Ask for More Suggestions Based on the CIs she Accepts

This process consists of five main steps as shown in both figures. In step 1, the analyst provides DRACA with the CI she wants to change (the initial CI). In our prototype tool, she enters the name into the 'CI Name' text box. In step 2, the analyst clicks the 'Suggest CIs' button which triggers DRACA to check its model to search for CIs that have previously changed with the initial CI.

If DRACA has no suggestions to make to the analyst (e.g., the indicated CI has not previously changed with any other CIs in DRACA's stored model), then this ends the process of interaction between DRACA and the analyst. However, if DRACA has suggestions to make, it moves to step 3 and provides the analyst with a list of ranked CIs. These CIs are displayed in the 'Suggested CIs (Ranked)' table, and are color coded according to their ranking (red, blue, and green).

The analyst, in step 4, then chooses the CIs to include in her change set. The information provided by DRACA in terms of support and confidence guide the analyst to choose the CIs to accept. The accepted CIs (along with the initial CI the analyst provided) form the change set that is being constructed. To accept any of the suggested CIs in Figure 6.2, the analyst highlights the CI(s) she wants to accept, and clicks on 'Accept'. This moves the selected CIs to the 'Accepted CIs'

table.

At this point, in step 5, the analyst may ask DRACA to make more suggestions based on the new CIs she added to the change set through the 'Suggest More' button, or she may choose to end the interaction with DRACA through the 'Done' button. If she asks DRACA for more suggestions, then for each newly accepted CI, DRACA searches for other CIs that have changed with it and suggests them to the analyst. DRACA will not, however, suggest any CIs that have been previously suggested. This process continues until either DRACA has no more suggestions to make or the analyst decides not to ask DRACA for more suggestions, and clicks 'Done'. Once this process is over, the analyst's interaction with DRACA ends.

If there are CIs that the analyst wants to add to the change set, and which DRACA has missed, the analyst is free to add these CIs to the change set. If this situation occurs, then DRACA was not able to suggest all CIs relevant to the analyst. However, if the analyst does not add any extra CIs after its interaction with DRACA is over, then DRACA found all the relevant CIs. After the analyst chooses the change set, she finalizes the change order which will be stored in the CMDB repository. This means that in future analyses, the model would be updated to include this newly created change set.

Additionally, the DRACA prototype tool gives the analyst the flexibility of choosing different filters for the list of returned CIs as shown in Figure 6.1. Sections 6.4 and 6.5 explain a few of the possible filters to use. These include a support threshold and a confidence threshold. We also allow the analyst to control the number of suggested CIs. Depending on the nature of the system, the DRACA tool will be set up with default thresholds that guarantee the best performance of the tool. This is convenient for novice analysts that are still not familiar with the system. On the other hand, more experienced analysts can change these settings to vary the list of returned CIs according to their need.

6.3 Measuring the Performance of DRACA

We need a way to evaluate DRACA's suggested CIs. Ideally, DRACA would suggest all the CIs in the change set without making any errors, but of course this does not often happen in practice. The recall and precision measures from the information retrieval field are appropriate for this type of evaluation. Recall measures the proportion of correct CIs retrieved by the system, while precision measures the proportion of suggested CIs that are correct [34].

The *Change Set* is the set of CIs that need to be considered for a change to be complete. We define the *Predicted Set* (P) as the set of all CIs DRACA suggests through the full iteration process (see Figure 6.1). We define the *Occurred Set* (O) as the CIs remaining in the change set after excluding the Initial CI provided by the analyst (i.e Change Set - Initial CI). The intersection of the predicted set and the occurred set, called PO, is the common CIs in both sets. For each constructed change set, we then calculate the recall and precision values for the predictions according to the following definitions [15]:

$$Recall = \frac{|PO|}{|O|} \tag{6.1}$$

$$Precision = \frac{|PO|}{|P|} \tag{6.2}$$

If no CIs are predicted (i.e P and thus PO is empty), precision is defined as 1 since there cannot exist any incorrect predictions in an empty predicted set. On the other hand, if the size of the change set is 1, and thus the size of the occurred set is 0, recall is defined as 1 since there are no CIs to predict.

In order to have a single measure that indicates the effectiveness of our predictions, we use the F-measure which is based on van Rijsbergen's effectiveness measure which combines recall and precision [34]. The F-measure is calculated according to Equation 6.3 which gives equal weighting to recall and precision.

$$F = 2 * \frac{precision * recall}{precision + recall} \tag{6.3}$$

The ideal F-measure is 1 where both recall and precision are 1. The F-measure accounts for undesirable situations with very low recall and high precision (or vice versa) by producing a low value. For example, if recall = 0.9 and precision = 0.1, the F-measure would be 0.18. On the other hand, if the recall and precision values were closer to the mid-range, for example, recall = 0.5 and precision = 0.7, the F-measure would be higher, around 0.58.

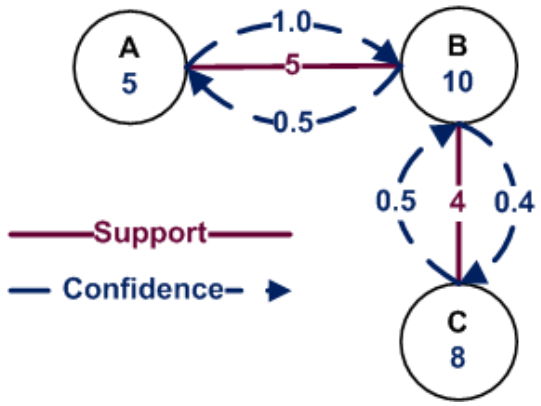

Figure 6.3: Tracking Changes between CIs

6.4 Model Used

In order to keep track of which CIs changed together in the past, we need a model that gives an indication of CIs which change together frequently versus those which do not. To accomplish this, we use two measures, *support* and *confidence*, from Zimmermann et al. [40].

Given a historical sequence of change sets, we start by counting the number of times each pair of entities (CIs in our case) appeared in a change order. For example, Figure 6.3 records that CIs B and C have occurred together in 4 change sets. We call this count the *support* between these two CIs. The support is drawn as an undirected edge between the two CIs. These counts are stored in the *support matrix* S as shown below. This example support matrix below corresponds to Figure 6.3.

$$S = \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{pmatrix} 5 & 5 & 0 \\ 5 & 10 & 4 \\ 0 & 4 & 8 \end{pmatrix} \end{array}$$

An entry S_{ij} shows how many times i and j changed together during the time period we are mining. For example, since A and B changed 5 times together, entry $S_{AB} = 5$. Note that S_{ii} is the number of times i changed with itself which is simply the count of how many times i changed in total. Also note that the support matrix is symmetrical since the number of times A and B changed together is the same as the number of times B and A changed together.

The support matrix S records the actual counts of co-changes. However, it is also useful to compute relative counts which take into consideration the total number of times each of these CIs have changed. For example, in Figure 6.3, A and B have changed 5 times together while the total number of times A and B have each changed are 5 and 10 respectively. This means that every time A changed, B changed with it as well while every time B changed, A changed with it only half the time. We call this relative count the *confidence* and calculate it as follows [40]:

$$C_{ij} = \frac{S_{ij}}{S_{ii}} \tag{6.4}$$

For example,

$$\begin{aligned} C_{BA} &= S_{BA}/S_{BB} \\ &= 5/10 \\ &= 0.5 \end{aligned}$$

This means that 50% of the time that B appeared in a change set, A also appeared in the same change set. The *confidence matrix* C for the example in Figure 6.3 is given below. Unlike the support matrix, the confidence matrix is not necessarily symmetric.

$$C = \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{pmatrix} 1.0 & 1.0 & 0 \\ 0.5 & 1.0 & 0.4 \\ 0 & 0.5 & 1.0 \end{pmatrix} \end{array}$$

6.5 Empirical Validation

To provide empirical results, we simulated the process described in Section 6.2 on a set of industrial data. We use the model presented in Section 6.4 to make the suggestions. We propose an initial procedure, our base case, and then propose three improved procedures which ideally filter out items mistakenly predicted by the base case. To evaluate these procedures, we test them with data from three year's use of a CMDB at CA. This section describes the data used, the experiment setup, and the results obtained.

Year	Number of Change Orders (COs)
2006	8,307
2007	9,784
2008	9,214
Average Num of COs Per Month	758
Total Number of COs Studied	27,305

Table 6.1: Number of Change Orders (COs) Studied in the GIS System

6.5.1 Description of the Data

Size of the System

The set of industrial data we used to evaluate our technique was provided by CA's Global Information Systems (GIS) team. Internally, CA uses its own Service Desk and CMDB products to manage its internal network and the services it provides to internal or external customers. The GIS team manages the CA network worldwide. In a change order, the set of CIs to be changed (i.e., the change set) is stored in a field named, 'Configuration Items'. However, some of the change orders in the CMDB had this field empty as they were not related to any CI. For example, some changes were more business or logistics related where an employee was changing their seat allocation or was requesting a new software on their laptop etc. In our work, we only considered IT related changes, and excluded change orders with empty change sets. Table 6.1 shows the number of change orders containing data in the 'Configuration Items' field per year in the GIS Service Desk during the three year period, January 2006 – December 2008, examined in this work.

From the three years of data, we determined the following facts about changes in the system. The average size of a change set is 4 CIs. At the end of 2008, there was a total of 37,906 CIs in the system. However, over the examined three year period, only 7,999 CIs were involved in the change reports (about 21% of the total number of CIs). The fact that not all the CIs have changes associated with them is not surprising since the CMDB keeps track of a large variety of CI types. These types range from software applications to hardware such as printers or LCD screens whose changes would not necessarily be logged in the CMDB. However, change orders are usually logged for business critical components that affect the services provided by the company.

Format of the Change Order Reports

In the CMDB, a Change Order has several fields including the requester, the assignee, the start date of the change, the description and the change set field (called the 'Configuration Items' field). Of the fields in a change order, we use only the change set field. The advantage of only using this one field is that change sets are easy to extract and easy to understand and, more importantly, as we will show, they may be useful as the basis for predicting future change sets.

6.5.2 Experiment Setup

To apply the change impact analysis process described in Section 6.2, we mined the change orders stored in the CMDB repository to extract the change sets. We use reports from three consecutive years: 2006, 2007 and 2008. We start by constructing the support and confidence matrices from January 2006, and use this knowledge to predict the change sets in February 2006 (i.e., learning period = Jan 2006 and testing period = Feb 2006). We then add the data from February 2006 to our matrices, and attempt to predict the change sets in March 2006 (i.e. learning period = Jan-Feb 2006 and testing period = March 2006), and so on until December 2008.

For each learning period, we calculate the support matrix and then calculate the confidence matrix according to Equation 6.4. To calculate the support matrix, S, we look at each change order in the specified learning date range, and increment the count of each CI that occurred in this change order as well as incrementing the count corresponding to each pair of CIs in the change order. That is, if a change order has CIs A and B, then S_{AA}, S_{BB}, S_{AB}, and S_{BA} are all incremented.

For each testing period, the aim is to use data (support and confidence matrices) from the corresponding learning period to try to reproduce the existing change sets given an initial CI in the set. Using the available data in this way assumes that the actual data available to us accurately reflects the ideal change set for a given change. This technique is commonly used in evaluating research ideas, such as the work done by Hassan and Holt [15] as well as that by Zimmermann et al. [42], when deploying the tool in a production system is not feasible. For each change in the testing period, we choose one of the CIs in its change set (Initial CI), and then suggest which other CIs should be changed along with it. To be able to compare results from the different filters we use, we fixed the Initial CI to be the first CI listed in a change set. We then suggest a list of CIs that should also be changed based on their corresponding values in the matrices.

To simulate the interaction of the analyst with DRACA, we check which of these suggestions actually exists in the occurred set. Any suggested CI that happens to also exist in the occurred set will be treated as accepted by the analyst. For each accepted CI, DRACA looks for more suggestions. This process continues until no more accepted CIs can be found, or no more suggestions can be found.

For example, consider this change set: $\{x, y, z, w\}$. In this example, the initial CI would be x making the occurred set $O = \{y, z, w\}$. Given x, assume DRACA suggests the following set of CIs: $\{y, r, w\}$. Since y and w are part of the occurred set, DRACA looks for more suggestions given that y and w are accepted. Assume DRACA now suggests $\{l, m\}$. Since none of the suggested CIs are in the occurred set, DRACA stops iterating making the predicted set, $P = \{y, r, w, l, m\}$. The intersection set would be $PO = \{y, w\}$. We can now compute recall and precision based on the sizes of sets: occurred $|O| = 3$, predicted $|P| = 5$ and intersection $|PO| = 2$. This means that for this change set, the following are the recall and precision values:

$$recall = \frac{|PO|}{|O|} = \frac{2}{3} = 0.667 \qquad precision = \frac{|PO|}{|P|} = \frac{2}{5} = 0.4$$

An experiment run consists of reproducing all the change sets from February 2006 to December 2008. For each experiment run, we calculate the average recall and precision values from all the change sets in the testing periods. However, to allow time for the learning process to stabilize, we exclude the first five testing periods, and calculate the average recall and precision for the change sets in July 2006 to December 2008. The F-measure for each experiment run is then calculated based on these average recall and precision values.

6.5.3 Experimental Results

We ran the above procedure on our three year dataset. We will first present the results from applying this procedure to a base case, which suggests each CI that has occurred in the past with a CI accepted by the analyst. We then present the results from running the procedure while applying three filtering techniques each of which remove some of the suggestions made in the base case. These techniques apply a support threshold, a confidence threshold, and exponential forgetting. We compare the performance of each of these filters to the base case.

Base Case

The base case begins by determining which items have occurred together in a change set in the past. It then predicts members of a change set as follows. When an item x is known to be in the current set (as determined by an analyst), the base case suggests each item y which has previously occurred in a change set with x. That is, S_{xy} is greater than zero. For example, using the support matrix in Section 6.4, given that CI B is in the change set, DRACA would also suggest A and C since they have non-zero entries in the support matrix.

This approach is simple and seems to be promising. When we ran our experimental procedure on this approach, it produced a recall of 0.9423. However its precision was only 0.0983, which is so low as to be of doubtful utility. This produces an F-measure of 0.178. We conclude that the base case makes too many suggestions, thereby producing high recall but too low precision. Consequently, we proceed to apply filters to prune out some of the base case's suggestions, hoping to gain precision at, ideally, a reasonable cost in decreased recall.

Filter 1: Support Threshold

We use the support matrix to count how many times each pair of CIs changed together. We use these counts as our first filter to refine the base case as follows. When an item x is known to be in the change set (as determined by an analyst), Filter 1 suggests each item y which has previously occurred in a change set with x more than t times, where t is a parameter called the support threshold. Otherwise, y is not suggested. This is based on the hypothesis that two CIs which have changed together 10 times in the past is stronger evidence that they may change together in the future when compared to two CIs that have changed only 5 times together in the past, for example. Our expectation is that such a threshold will eliminate enough suggested CIs to increase precision and thereby to improve upon the base case. Table 6.2 shows the effect of varying the support threshold. The graph in Figure 6.4 shows the plot of these results.

Higher thresholds prune out more suggestions that would have been made by the base case. If the threshold is zero, no pruning takes place and this procedure devolves to the base case; this can be seen in the first row of Table 6.2. As can be seen, as the support threshold increases, the precision increases with an accompanying decrease in recall. This, in turn, causes the F-measure to improve to about

Support Threshold	Recall	Precision	F-measure
0 (Base Case)	0.9423	0.0983	0.1780
10	0.7973	0.4674	0.5893
20	0.7488	0.6679	0.7061
30	0.7215	0.7572	0.7389
40	0.7032	0.7925	0.7452
50	0.6900	0.8307	0.7539
60	0.6793	0.8706	0.7635
70	0.6709	0.8866	0.7638
80	0.6627	0.8995	0.7635
90	0.6558	0.9095	0.7621
100	0.650	0.9208	0.7621
110	0.6451	0.9295	0.7616

Table 6.2: Recall/Precision/F-measure with Increasing Support Thresholds (Filter 1)

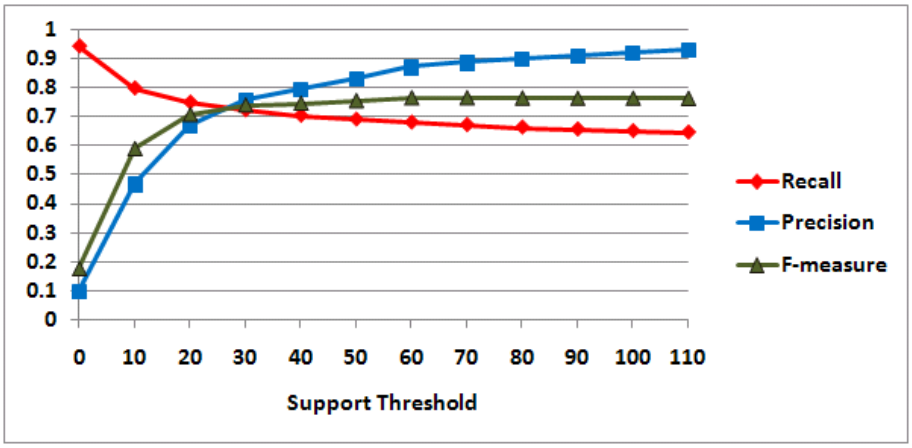

Figure 6.4: Recall/Precision/F-measure with Increasing Support Thresholds (Filter 1)

0.76 at a threshold of about 70, and then starts decreasing. At that point the recall is about 67% and the precision is about 89%.

Although these values seem to indicate that this filter is potentially useful in practice, in fact it cannot be expected to continue to work well. This is because the counts of pair occurrences (i.e., support) continues to grow with time, while the support threshold remains constant. Eventually a problem will arise, in that the increasing support counts will allow more and more suggestions, so precision will

Confidence Threshold	Recall	Precision	F-measure
0 (Base Case)	0.942	0.098	0.178
0.1	0.924	0.320	0.476
0.2	0.899	0.399	0.552
0.3	0.872	0.476	0.616
0.4	0.850	0.529	0.652
0.5	0.816	0.625	0.707
0.6	0.791	0.682	0.732
0.7	0.753	0.768	0.760
0.8	0.718	0.837	0.773
0.82	0.712	0.850	0.775
0.84	0.710	0.857	0.776
0.86	0.695	0.887	0.779
0.88	0.678	0.899	0.773
0.9	0.666	0.914	0.770
0.92	0.658	0.924	0.769
0.94	0.650	0.932	0.766
0.96	0.644	0.938	0.764
0.98	0.638	0.942	0.761
1.0	0.594	1	0.745

Table 6.3: Recall/Precision/F-measure with Increasing Confidence Thresholds (Filter 2)

be degraded.

Filter 2: Confidence Threshold

Filter 1 improved the F-measure of our predictions, but it, unfortunately, depends on the length of time the system has been running. The second filter (confidence threshold) is much like the first filter but avoids this problem by using relative counts. When an item x is known to be in the change set (as determined by an analyst), this procedure suggests each item y such that y's confidence with respect to x is greater than u (i.e., $C_{xy} > u$), where u is called the confidence threshold.

To illustrate how Filter 2 works, consider the example confidence matrix we presented earlier (see Page 51) in which $C_{BA} = 0.5$ and $C_{BC} = 0.4$. If B is known to be in a change set, and if the confidence threshold is 0.2, then both A and C

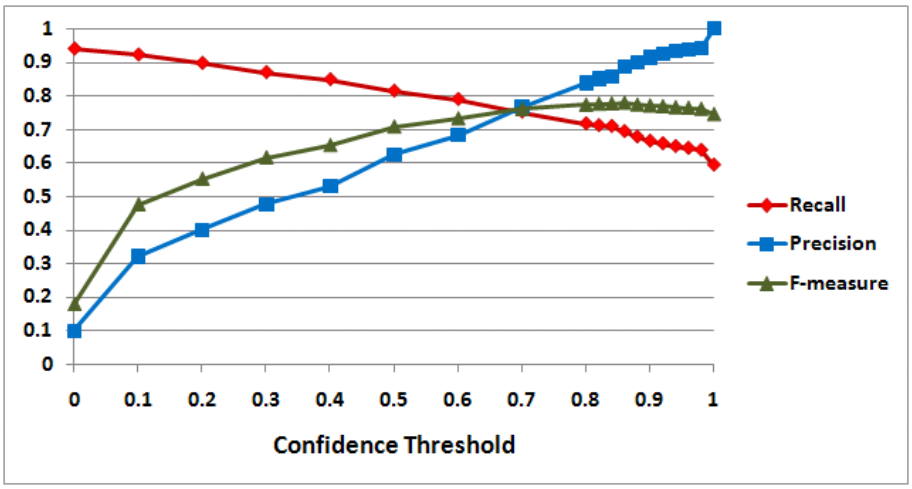

Figure 6.5: Recall/Precision/F-measure with Increasing Confidence Thresholds (Filter 2)

will be suggested as likely members of the change set because both C_{BA} and C_{BC} exceed 0.2. However, if the confidence threshold is 0.45, only A will be suggested.

Table 6.3 shows the effect of varying the confidence threshold on the recall and precision levels, and accordingly on the F-measure. Figure 6.5 shows the plot of these results. As in the case of the previous filter (support threshold), larger values of the threshold prune out more of the suggestions that would have been made by the base case. Similarly, if the threshold u is zero, this procedure devolves to the base case; this can be seen in the first row of Table 6.3.

The table and figure show that as the confidence threshold is raised, the F-measure keeps improving until a threshold of about 0.8 is reached and then starts falling. To find a more exact value for which the F-measure maximizes, we tested all confidence thresholds form 0.8 - 1.0 in intervals of 0.02. This showed that the F-measure reached a maximum value of 0.779 at a threshold of about 0.86, and then started falling. At that maximum value, recall was about 70% and precision was about 89%. These values are somewhat better than those from the previous filter, but more important is the expectation that this second filter will keep producing good recall and precision with the passage of time. These high values indicate that this filter is potentially useful in practice.

Half-life	Recall	Precision	F-measure
3 months	0.7074	0.8623	0.7772
6 months	0.6987	0.8782	0.7782
9 months	0.6982	0.8828	0.7798
12 months	0.6980	0.8847	0.7803
15 months	0.6965	0.8858	0.7798
18 months	0.7176	0.8453	0.7763
21 months	0.6961	0.8867	0.7799
24 months	0.6960	0.8868	0.7799
27 months	0.6965	0.8858	0.7798
30 months	0.6960	0.8868	0.7799
33 months	0.6959	0.8869	0.7799
36 months	0.6959	0.8869	0.7799
∞	0.6945	0.8869	0.7790

Table 6.4: Recall/Precision/F-measure withe a Confidence Threshold of 0.86 and Increasing Half-life, λ (Filter 3)

Filter 3: Exponential Forgetting

Our third filter is based on the idea that more recent information should count for more. Since we are dealing with dynamic systems, the relationship between CIs may change over time or the way they depend on each other may change. Therefore, more recent change sets should reflect the current state of the system better than change sets that took place a year ago, for example. When applying exponential forgetting, we use the concept of half-life to indicate how fast the forgetting occurs. The half-life measures after how much time will we only remember half the amount of any information. This determines the rate at which we forget previous information. A shorter half-life means quicker forgetting of information, while a longer half-life means retaining any learned information for a longer period of time.

As previously explained, in order to calculate the support matrix, we count the number of times CIs appear in the change reports. If no exponential forgetting is used, then every time a CI appears in a change report, we increment its count by 1 regardless of the time this change occurred at. Let us call the amount we increment by, in this case 1, the *impact*, I, of the new piece of information. With exponential forgetting, the impact of a new piece of information will depend on the current time and on the time this piece of information occurred at. Given the half life, λ, we

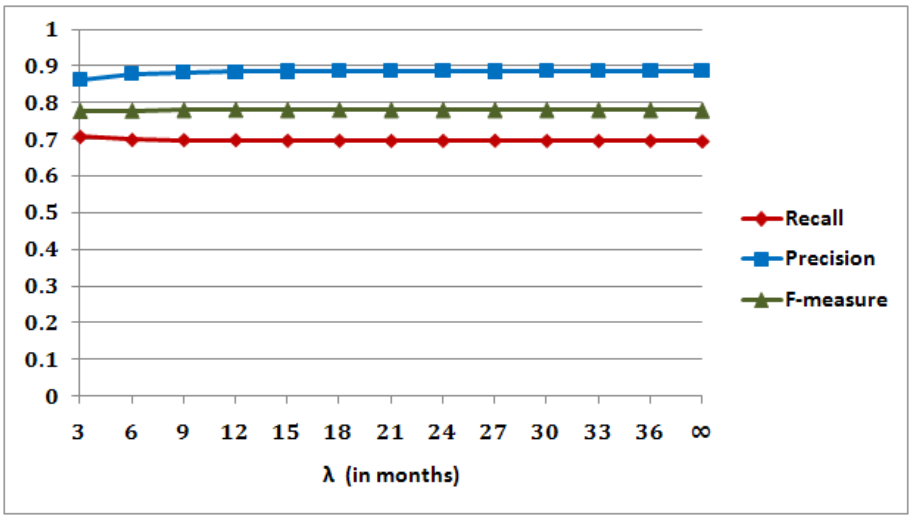

Figure 6.6: Recall/Precision/F-measure with a Confidence Threshold of 0.86 and Increasing Half-life, λ (Filter 3)

use the following formula to calculate the impact, I_{t_0} at the present time t_{now} of a change set that occurred at time t_0:

$$I_{t_0} = 2^{-\frac{(t_{now}-t_0)}{\lambda}} \tag{6.5}$$

For simplicity, in our work, we update the support and confidence matrices every month. Therefore, when predicting the change sets for April for example, we assume that t_0 is March 31st. The impact of the data learned from previous months in the support and confidence matrices is adjusted accordingly. After we are done predicting the change sets in April, the new t_0 time becomes April 30th, and we add the change sets in April to our matrices, and adjust their impact accordingly.

Before applying exponential forgetting, we filter using our best results from Filters 1 and 2. This is done by applying Filter 2 (which produced better results than Filter 1) with its confidence threshold set to 0.86 (the best setting of that threshold). Table 6.4 and Figure 6.6 show the results of exponential forgetting with that threshold active.

As shown, varying the half-life only slightly improved results. The best half-life seems to be 12 months with an F-measure of 0.7803.

6.5.4 Discussion of Results

The obtained results seems promising since they indicate that the change set data is highly predictive, with high values of recall and precision. However, a question that arose during these experiments is this: Why are the values of recall, precision and the F-measure so high? These values are higher than figures from experiments such as predicting change sets in source code updating [15, 42, 37]. This question became more intriguing with the bottom row of Table 6.3 which documents the situation when the confidence threshold is 1.0. This threshold implies that the procedure will make absolutely no suggestions. In other words, the procedure will predict that the change set contains nothing but the item originally provided by the analyst. As the table shows, with this threshold, the F-measure (0.745) is reasonably high.

Both the high recall (0.594) and high precision (1.0) contribute to this high F-measure. The perfect precision can be explained as follows. Since precision measures the percentage of suggestions made that were correct, making no suggestions at all means making no mistakes at all which produces a perfect precision of 1.0. This maximal possible value of precision helps make the F-measure be high.

However, it was the corresponding high recall value that came as a surprise. Achieving perfect precision is usually accompanied by a very low recall rate, but this was not what we observed. The corresponding observed high recall value (0.594) has the following explanation. In the analyzed data, we found out that 16,294 change orders out of the total of 27,305 change orders studied contained only 1 CI in their change set. This means that roughly 59% of the change sets in the data contain exactly one CI. Each such CI will be selected by the analyst as the initial CI leaving the occurred set empty. This means that the occurred set is empty about 59% of the time. If the occurred set is empty, recall is 1.0 by definition [15]. However, if the occurred set is not empty, and no suggestions are made, then recall is 0. Since 59% of the change sets have empty occurred sets, it follows that the average recall of all change sets is about 59% in the case in which the procedure makes no suggestions.

In our experiment, we determined that a value of 0.86 for the confidence threshold u produces the maximum F-measure value (0.779). We interpret this as follows. This u value (0.86) is close enough to 1.0 so it suggests no CIs most of the time which is correct for change sets of size 1. Additionally, when it does make suggestions, it suggests only high frequency pairs that have a high chance of being correct. This explains why the experiment has such simultaneous high recall and precision results.

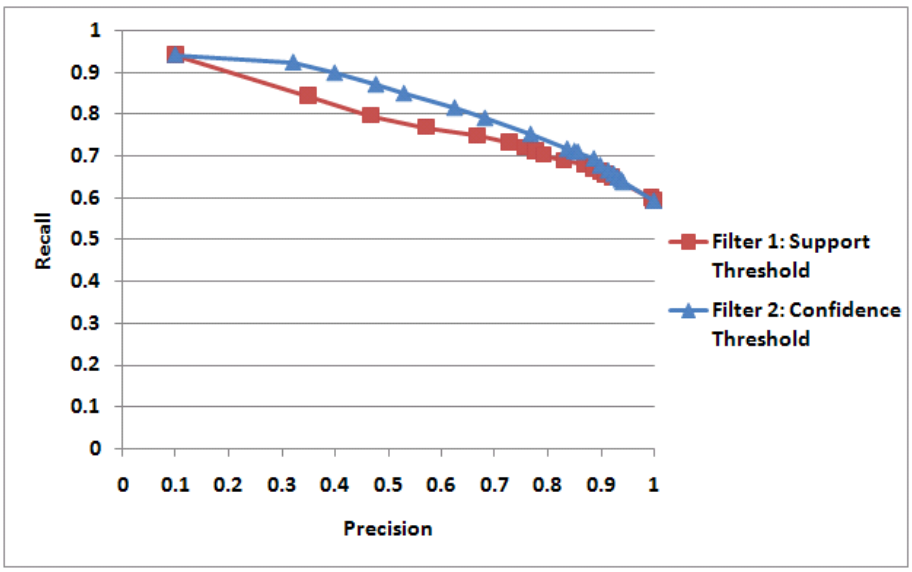

Figure 6.7: Comparing Recall and Precision for Filter 1 (Support Threshold) and Filter 2 (Confidence Threshold)

One interpretation of this situation is as follows. Filter 2 learned to perform well (as shown by its recall, precision and F-measure values) and it accomplished this by making no suggestions in many cases. Another interpretation would question the convention of defining precision to be 1.0 in the case of an empty suggestion set as this definition seems to inflate the value of precision. Another interpretation or approach would ignore all singleton change sets and would re-run the experiments using only change sets of size at least two. We favor the first interpretation, but recognize that the other interpretations have merit. Regardless of the interpretation, it appears that Filter 2 may adapt reasonably well to other historical data of change sets that either do or do not contain many singleton change sets; future work may confirm this position.

Figure 6.7 compares the recall-precision curves of the first two filters. These curves are based on the precision and recall columns of Tables 6.2 and 6.3. The top left point of these two curves corresponds to the base case. As the figure shows, the curve for Filter 1 (support threshold) lies below that of Filter 2 (confidence threshold). This indicates that in all cases Filter 2 outperformed Filter 1. Since Filter 2 produced the best results in our experiments, we used confidence to rank the set of suggested CIs that are displayed to the analyst.

Applying exponential forgetting with the optimal confidence threshold of 0.86 slightly improved results (raising the F-measure from 77.9% to 78.03% with a half life $\lambda = 12$ months). We believe that this is because the nature of the system data

we analyzed is more or less stable in the sense that once a change set occurs, it is likely that it will occur again in the future. With a more dynamic system, we expect that varying the half-life would produce more significant variances in the results.

6.5.5 Threats to Validity

Although our results seem promising, we still cannot conclude that our findings will apply to different systems. We presented our results from analysis of one system, which is the system used by CA's GIS team. We reviewed one other system, but it did not use change sets so we were not able to apply our technique to it. Ideally, in the future we may be able to analyze more systems which use change sets, but this may be challenging as it is not easy to gain access to industrial systems. Analyzing additional systems would allow us to better evaluate the utility of various filters.

6.6 Summary

In this chapter, we presented the collaborative change impact analysis process provided by DRACA. Given an initial CI that is provided by the analyst, DRACA suggests more CIs that might also be included in the current change set. Based on the CIs an analyst accepts in the change set, DRACA could suggest additional CIs. All the lists of suggested CIs are ranked according to their confidence measure. The support and confidence measures are used to model the strength of correlation between two CIs according to their co-change history.

We tested our process on a set of industrial data. By applying different filters on the suggested CIs, we were able to predict change sets with a combination of recall and precision as high as 69.8% and 88.5% respectively. The filters applied include a support threshold, a confidence threshold and exponential forgetting. The confidence threshold seemed to be the most effective threshold and greatly improved results.

Chapter 7

Conclusions

7.1 Summary of Topics Addressed

Enterprise IT management (EITM) is becoming increasingly important every day since most business services are delivered through an underlying IT infrastructure. Since any failure in the IT infrastructure results in financial losses for the business, properly managing an IT system is crucial. The Information Technology Infrastructure Library (ITIL) provides guidelines to processes and tools that can be used for EITM. A Configuration Management Database (CMDB) is one of these tools. A CMDB provides a basis for Configuration Management by tracking configuration items (CIs) and the relationships which in turn provides a basis for Change Management. Change Management involves controlling changes applied to the system. This includes change impact analysis which identifies all the CIs that need to be updated for a change to be correctly implemented. Configuration Management also provides a basis for root cause analysis since it provides the relationships between CIs which allows the propagation of problems to be tracked. Root cause analysis involves finding the ultimate cause of a problem accurately and efficiently. This allows analysts to spend less time on finding the cause of the problem, and more time to solving the problem resulting in a shorter mean time to recovery. This thesis addresses the challenges involved in providing useful decision support for root cause analysis and change impact analysis.

7.2 Summary of DRACA's Approach

This thesis presented DRACA, a Decision-support framework for Root cause Analysis and Change impact Analysis. In DRACA, root cause analysis is supported by a causality graph which is constructed from data mined out of the CMDB. We demonstrated how the data in the CMDB can be mined to produce a causality graph, and how this graph can be represented in a matrix. Through manipulations on this matrix, we produced a root cause matrix which records the probability that one CI is the root cause of an incident in another CI. This is done through three steps. The first steps produces the basic root cause matrix from the data in the CMDB. The second step considers the time of the reported incident along with the system's calendar information to adjust the current root cause matrix accordingly. The third step then assigns different weights for the different CIs according to the last time they have changed to produce the final root cause matrix. The final root cause matrix effectively provides a ranked list of the suspect CIs for the current incident.

To be able to provide decision support for change impact analysis, we used the hypothesis that CIs that have change together frequently in the past are likely to change again together in the future. Through mining change sets from the CMDB, DRACA suggests the CIs that might be included in a change set, and also provides a ranking of these CIs based on their pattern of recurrence in the past. By modeling this recurrence in support and confidence matrices, and by applying different filters, we were able to predict change sets with a combination of recall and precision as high as 69.8% and 88.5% respectively. We presented the effect of different filters we applied to the suggested set of CIs. These include a support threshold, a confidence threshold and exponential forgetting. The confidence threshold seemed to be the most effective threshold and greatly improved results. Although we mined our data out of the CMDB repository, this work can be applied on any repository with a different format as long as change sets are recorded.

7.3 Thesis Contributions

A summary of the contributions of this thesis are as follows:

- An exploration of data repositories (CMDBs) that have not been previously attempted in the mining software repositories research community.

- A causality model providing decision support for root cause analysis based on this mined data.
- A process for mining historical change information to suggest CIs for future change sets based on a ranking model. *Support* and *confidence* measures were used to make the suggestions.
- Empirical results from applying the proposed change impact analysis process to industrial data. Our results show that the change sets in the CMDB were highly predictive, and that with a confidence threshold of 80% and a half life of 12 months, an overall recall of 69.8% and a precision of 88.5% were achieved.
- An overview of lessons learned from using a CMDB, and the observations we made while working with the CMDB.

7.4 Future Work

There are some enhancements that could be added to DRACA's functionality to make it more useful in EITM. Additionally, more empirical studies could be performed to evaluate DRACA's performance. For the purposes of root cause analysis, we were not able to empirically evaluate our model due to the difficulty of obtaining suitable data. Obtaining industrial data is a real challenge. One of our future steps is, therefore, to continue searching for data sets on which we can test our technique. As another option, we are also considering simulation to test our model. That is, we will design a small system and store its configuration in the CMDB. We will simulate faults causing a failure cascade which eventually causes an incident according to the probabilities stored in our model. We would then use the probabilities in our model to identify the root causes of these incidents.

Currently, our root cause analysis process can suggest what possible root cause CIs for an incident, but cannot indicate how the failure chain occurs. That is if A had a fault which caused B to fail which subsequently caused C to fail, then DRACA would report A as a possible root cause to the incident in C. However, it does not currently present the possible cause-effect chain. This feature could be incorporated in DRACA by analyzing the the timing of all the events in related CIs to order them chronologically and construct a possible cause-effect scenario.

From the change impact analysis side, we have used simple heuristics to investigate if the nature of changes stored in a CMDB is predictive or not. Since

our results seem promising, we intend to expand on this work using more of the available information. As shown in Section 6.5.1, there are several input fields in a change order that we might use in the future to derive additional information to help predict change impact. For example, looking at the description of the change along with the CIs which changed can allow us to have a classification of the different types of changes, and to predict which CIs to change based on the nature of the change. Additionally, looking at which analysts perform which changes can allow us to recommend the best analyst to perform the current change.

We also plan to add more decision support to our tool. For example, DRACA could not only suggest other CIs to change, but also the best time to implement this change based on the availability schedule of all CIs involved in the change set. Additionally, to reduce the probability of a change causing a failure sometime later in the system, we could bundle change orders with incident reports in the CMDB to identify which changes induced incidents, similar to the work in [31]. This will increase the cooperation between root cause analysis and change impact analysis. Finally, we hope to get access to more data sets to be able to compare the performance of the different filters on systems of different nature.

From the overall perspective of our framework, once we are able to find a dataset on which we can apply both root cause analysis and change impact analysis, we would like to test the collaboration between both processes. That is, we could use the root cause analysis model to validate the change impact analysis model and vice versa. If a change in x requires a change in y for the change to be successful, it is very likely that an incident in y might be caused by a change in x.

7.5 Thesis Conclusion

The work presented in this thesis is unique in that it combines different aspects of root cause analysis and change impact analysis into one framework. The problems we address are real world challenges which we have learned about from discussions with CMDB users and CMDB experts. Working closely with CA Labs has given us an insight into the problems faced by CMDB users. DRACA lays a foundation for a framework that addresses these challenges.

References

[1] Archimate. `http://old.telin.nl/index.cfm?language=en&id=252&context=253`. 23

[2] Robert S. Arnold and Shawn A. Bohner. Impact analysis - towards a framework for comparison. In *ICSM '93: Proceedings of the Conference on Software Maintenance*, pages 292–301, Washington, DC, USA, 1993. IEEE Computer Society. 12

[3] Algirdas Avizienis, Jean-Claude Laprie, and Brian Randell. Dependability and its threats: a taxonomy. In *IFIP '04: Proceedings of 18th World Computer Congress: Building the Information Society*, pages 91 – 120, Toulouse, France, 2004. Kluwer Academic Publishers. 11

[4] Linda Badri, Mourad Badri, and Daniel St-Yves. Supporting predictive change impact analysis: A control call graph based technique. In *APSEC '05: Proceedings of the 12th Asia-Pacific Software Engineering Conference*, pages 167–175, Washington, DC, USA, 2005. IEEE Computer Society. 16

[5] A.T. Bouloutas, S. Calo, and A. Finkel. Alarm correlation and fault identification in communication networks. *IEEE Transactions on Communications*, 42(234 Part 1):523–533, 1994. 11

[6] B. Burns and C.T. Morrison. Temporal abstraction in Bayesian networks. In *AAAI Spring Symposium, Palo Alto, CA*, 2003. 21

[7] Gerardo Canfora and Luigi Cerulo. Impact analysis by mining software and change request repositories. In *METRICS '05: Proceedings of the 11th IEEE International Software Metrics Symposium*, page 29, Washington, DC, USA, 2005. IEEE Computer Society. 17

[8] Nicos Christofides. *Graph theory: An algorithmic approach (Computer science and applied mathematics)*. Academic Press, Inc., Orlando, FL, USA, 1975. 12

[9] Dale Clark, Pratul Dublish, Mark Johnson, et al. The Federated CMDB Vision. Technical report, A joint white paper from BMC Software, CA, Fujitsu, Hewlett-Packard, IBM, Microsoft, 2007. 10

[10] F.S. de Boer, M.M. Bonsangue, L.P.J. Groenewegen, A.W. Stam, S. Stevens, and L. Van Der Torre. Change impact analysis of enterprise architectures. In *IRI '05: IEEE International Conference on Information Reuse and Integration*, pages 177–181, 2005. 22

[11] Andreas Hanemann. A hybrid rule-based/case-based reasoning approach for service fault diagnosis. In *AINA '06: Proceedings of the 20th International Conference on Advanced Information Networking and Applications*, pages 734–740, Washington, DC, USA, 2006. IEEE Computer Society. 19

[12] Andreas Hanemann. *Automated IT Service Fault Diagnosis Based on Event Correlation Techniques*. PhD thesis, Universitat der Bundeswehr Munchen, 2007. 19

[13] Andreas Hanemann, David Schmitz, and Martin Sailer. A framework for failure impact analysis and recovery with respect to service level agreements. In *SCC '05: Proceedings of the 2005 IEEE International Conference on Services Computing*, pages 49–58, Washington, DC, USA, 2005. IEEE Computer Society. 23

[14] Salim Hariri, Guangzhi Qu, Tushneem Dharmagadda, Modukuri Ramkishore, and Cauligi S. Raghavendra. Impact analysis of faults and attacks in large-scale networks. *IEEE Security and Privacy*, 1(5):49–54, 2003. 18

[15] Ahmed E. Hassan and Richard C. Holt. Predicting change propagation in software systems. In *ICSM '04: Proceedings of the 20th IEEE International Conference on Software Maintenance*, pages 284–293, Washington, DC, USA, 2004. IEEE Computer Society. 16, 49, 53, 61

[16] Irene Katzela and Mischa Schwartz. Schemes for fault identification in communication networks. *IEEE/ACM Transactions on Networking (TON)*, 3(6):753–764, 1995. 11

[17] Stuart Kent. Model driven engineering. In *IFM '02: Proceedings of the Third International Conference on Integrated Formal Methods*, pages 286–298, London, UK, 2002. Springer-Verlag. 3

[18] Aman Kumar, Preethi Raghavan, Jay Ramanathan, and Rajiv Ramnath. Enterprise interaction ontology for change impact analysis of complex systems. In *APSCC '08: Proceedings of the 2008 IEEE Asia-Pacific Services Computing Conference*, pages 303–309, Washington, DC, USA, 2008. IEEE Computer Society. 22

[19] Lundy M. Lewis. A case-based reasoning approach to the resolution of faults in communication networks. In *INFOCOM '93: Proceedings of 12th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1422–1429, San Francisco, CA, USA, 1993. 18

[20] David A. Messineo and Malcolm Ryder. Why implement a configuration management database (cmdb)? seven fundamental use cases. 2008. 10

[21] Siavash Mirarab, Alaa Hassouna, and Ladan Tahvildari. Using bayesian belief networks to predict change propagation in software systems. In *ICPC '07: Proceedings of the 15th IEEE International Conference on Program Comprehension*, pages 177–188, Washington, DC, USA, 2007. IEEE Computer Society. 17

[22] Nachiappan Nagappan, Thomas Ball, and Andreas Zeller. Mining metrics to predict component failures. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 452–461, New York, NY, USA, 2006. ACM. 16

[23] Maitreya Natu and Adarshpal S. Sethi. Using temporal correlation for fault localization in dynamically changing networks. *International Journal of Network Management*, 18(4):301–314, 2008. 21

[24] Office of government commerce (ogc), ed.: Glossary of terms, definitions and acronyms v3. *IT Infrastructure Library (ITIL)*, 2007. 10

[25] Office of government commerce (ogc), ed.: Service support. *IT Infrastructure Library (ITIL)*, 2000. 2, 7, 8

[26] T.J. Ostrand and E.J. Weyuker. A tool for mining defect-tracking systems to predict fault-prone files. In *MSR '04: Proceedings of International Workshop on Mining Software Repositories*, pages 85–89, 2004. 16

[27] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988. 17

[28] Ali Razavi and Kostas Kontogiannis. Pattern and policy driven log analysis for software monitoring. In *COMPSAC '08: Proceedings of the 2008 32nd Annual IEEE International Computer Software and Applications Conference*, pages 108–111, Washington, DC, USA, 2008. IEEE Computer Society. 20

[29] Thomas Reidemeister, Mohammad Ahmad Munawar, Miao Jiang, and Paul A.S. Ward. Diagnosis of recurrent faults using log files. In *CASCON '09: Proceedings of the 19th Centre of Advanced Studies Conference*, November 2009. 21

[30] M. Renieris and S.P. Reiss. Fault Localization With Nearest Neighbor Queries. In *ASE '03: Proceedings of the 18th IEEE International Conference on Automated Software Engineering*, volume 1527, pages 17–00, 2003. 16

[31] Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? In *MSR '05: Proceedings of the 2005 international workshop on Mining software repositories*, pages 1–5, New York, NY, USA, 2005. ACM. 68

[32] Malgorzata Steinder and Adarshpal S. Sethi. A survey of fault localization techniques in computer networks. *Science of Computer Programming*, 53(2):165–194, 2004. 11

[33] Ahmed Y. Tawfik and Eric Neufeld. Temporal bayesian networks. In *TIME '94: Proceedings of the International Workshop on Temporal Representation and Reasoning*, 1994. 21

[34] C.J. van Rijsbergen. Information retrieval, 1979. 48, 49

[35] Wei Wang, Hao Wang, Bo Yang, Liang Liu, Peini Liu, and Guosun Zeng. A Bayesian knowledge engineering framework for service management. In *NOMS '08: Proceedings of IEEE Network Operations and Management Symposium, 2008*, pages 771–774, 2008. 19

[36] Yiqiao Wang, Sheila A. McIlraith, Yijun Yu, and John Mylopoulos. An automated approach to monitoring and diagnosing requirements. In *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 293–302, New York, NY, USA, 2007. ACM. 20

[37] Annie T. T. Ying, Gail C. Murphy, Raymond Ng, and Mark C. Chu-Carroll. Predicting source code changes by mining change history. *IEEE Transactions on Software Engineering*, 30(9):574–586, 2004. 16, 61

[38] Andreas Zeller. Isolating cause-effect chains from computer programs. In *SIGSOFT '02/FSE-10: Proceedings of the 10th ACM SIGSOFT symposium on Foundations of software engineering*, pages 1–10, New York, NY, USA, 2002. ACM. 15

[39] Alice X. Zheng, Jim Lloyd, and Eric Brewer. Failure diagnosis using decision trees. In *ICAC '04: Proceedings of the First International Conference on Autonomic Computing*, pages 36–43, Washington, DC, USA, 2004. IEEE Computer Society. 17

[40] Thomas Zimmermann, Stephan Diehl, and Andreas Zeller. How history justifies system architecture (or not). In *IWPSE '03: Proceedings of the 6th International Workshop on Principles of Software Evolution*, page 73, Washington, DC, USA, 2003. IEEE Computer Society. 17, 50, 51

[41] Thomas Zimmermann, Rahul Premraj, and Andreas Zeller. Predicting defects for eclipse. In *PROMISE '07: Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, page 9, Washington, DC, USA, 2007. IEEE Computer Society. 16

[42] Thomas Zimmermann, Peter Weisgerber, Stephan Diehl, and Andreas Zeller. Mining version histories to guide software changes. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 563–572, Washington, DC, USA, 2004. IEEE Computer Society. 17, 53, 61