Domain-Based Change Propagation Analysis: An Enterprise System Case Study

Amir Aryani, Ian D. Peake, Margaret Hamilton,
School of Computer Science and Information Technology, RMIT University
{Amir.Aryani,Ian.Peake,Margaret.Hamilton}@rmit.edu.au

Abstract—Change propagation has mainly been estimated by maintenance history or source code analysis. However, sometimes history and code are inaccessible, or impractical to analyse, such as for heterogeneous sources.

Previously we hypothesised that change propagation from modifying domain level components may be predicted purely from information available to domain users. We proposed domain-based change propagation analysis, enabling analysts and domain experts to predict conceptual coupling independent of implementation.

This paper reports on application of domain-based analysis to a significant (enterprise) system. We performed both domain-based analysis and a well known history-based analysis and compared the results. Like history-based approaches, domain-based analysis reveals coupling between software components, can assist to prevent errors in software maintenance, and predict change propagation. We conclude that it may be worth applying to certain kinds of systems where established approaches would be considered impractical.

I. INTRODUCTION

Predicting change propagation is a key software maintenance challenge [4]. Imagine a tool which enables domain experts to estimate change propagation within a software system without technical knowledge of software engineering and access to the source code.

For typical information systems and data driven business applications, domain experts have comprehensive knowledge about system functionality, and are the primary requesters of additive and corrective system changes. When domain experts think about enhancements or bug fixes to a system, typically the first step is to discuss the matter with software engineers responsible for relevant aspects of the system. The outcome of such a discussion in most enterprise environments is an estimate of the potential impact of the proposed change. If such estimations could be derived by domain experts, and were sufficiently reliable, this would assist software maintainers by saving time and effort on making decision about prospect changes. A tool assisting such estimates would be applicable to legacy systems with missing or outdated design artefacts, heterogeneous systems where source code dependencies are not easily traceable, or change history is unavailable.

In earlier work we hypothesised that change propagation in domain level components can be predicted from a model of the coupling between domain information. We introduced conceptual coupling as a measurement of commonality of domain information between two software components [1].

In this paper we refine our proposed approach and for the first time apply it to a significant, enterprise system. We use conceptual coupling as an indicator of how domain components share domain variables, and the probability of common functionality, and utilise such couplings to predict the change propagation between software domain level components.

We distinguish between the previously introduced notion of symmetric coupling and a new notion of asymmetric conceptual coupling to find sets of components most likely to be affected by a change. The proposed approach have been evaluated based on source code evolutionary couplings derived by a well known technique for mining version history [26]. The change propagation incidents extracted from the maintenance history of the system form the basis of the evaluation.

The rest of this paper is organised as follows: In section II we provide the methodology to derive conceptual coupling. Section III provides the results of a case study. Section IV discusses the related work, and finally section V closes with conclusions and further work.

II. CONCEPTUAL COUPLING

Here we summarise conceptual coupling, how it is modelled, how such models are derived and how they are used to predict change propagation.

System domain-level behaviour is the system specification from the point view of the domain user. This is an abstract view of the system consisting purely of functionality provided to the domain user and generally agnostic with respect to its implementation.

We use the following terminology for system domain-level behaviour:

- A domain variable is a variable unit of data which has a clear identity at the domain level.
- A domain function provides proactive or reactive domain-level behaviour of the system which includes at least one domain variable as an input or output.
- A user interface component (UIC) is a system component directly interacting with the system domain user and containing one or more domain functions.

1This is a pre-press version of a paper with the same title to appear in ICSM 2010, Timisoara, Romania. ©2010 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
For example, in a business software system, UICs are the software data entry forms, each form provides one or more functions to the end user, and the data fields visible on forms are the domain variables.

A. Notations and Definitions

Much of this section quotes [1] with the exception of the new definition of asymmetric weight (Definition 2).

For the remainder of the paper we use standard notation for binary relations. For \( R, Q \subseteq A \times A \), we denote by \( R.Q \) their composition, i.e., \( x.R.Q.y \) if and only if \( \exists z : x.Rz \land z.Qy \). The inverse of \( R \) is denoted by \( R^{-1} \) and \( I.D \) the identity relation. Moreover we abbreviate \( x.R = \{ y | x.Ry \} \).

The three key element types are modelled as follows:

- **Domain variables** are modelled by a finite set \( V \) of variable symbols.
- **Domain functions** are modelled by a finite set \( F \) of function symbols.
- **User Interface Components** (UICs) are modelled by a finite set \( C \) of (component symbols).

A binary relation \( USE \subseteq F \times V \) represents elementary dependencies between functions and variables as the input-output variables. Because we are only interested in domain functionality interacting with a domain user then \( f.\text{USE} \neq \emptyset \) for all \( f \in F \), i.e., a domain function uses at least one variable.

The associated function symbols are represented by the relation \( \text{HAS} \subseteq C \times F \). We require that \( c.\text{HAS} \neq \emptyset \) for all \( c \in C \), i.e., a component has at least one function.

We assume the system under analysis (SUA) is fixed, that is, \( V, F \) and \( C \) are fixed and so are their \( USE \) and \( HAS \) relations.

**Definition 1.** The dependency matrix \( M \) of a SUA is a matrix of binary elements \( M_{c,v} \) where \( c \in C \) and \( v \in V \),

\[
M_{c,v} = \begin{cases} 1 & : v \in c.\text{HAS}.\text{USE} \\ 0 & : v \notin c.\text{HAS}.\text{USE} \end{cases}
\]

**Definition 2.** The asymmetric weight function is a weight function where the weight \( w_a : C \times C \to [0..1] \) assigns probabilities to a pair of UICs by

\[
w_a(c, c') = \frac{|c.\text{HAS}.\text{USE} \cap c'.\text{HAS}.\text{USE}|}{|c.\text{HAS}.\text{USE}|}
\]

**Definition 3.** The symmetric weight function is a weight function where the weight \( w : C \times C \to [0..1] \) assigns probabilities to a pair of UICs by

\[
w(c, c') = \frac{|c.\text{HAS}.\text{USE} \cap c'.\text{HAS}.\text{USE}|}{|c.\text{HAS}.\text{USE} \cup c'.\text{HAS}.\text{USE}|}
\]

By conceptual coupling we refer to either \( w \) and \( w_a \) or both, as indicated by the context\(^2\).

\(^2\)In our previous work we use only symmetric weight and refer to it as conceptual coupling.

B. Example

In a Work Order system, the UICs WorkOrderCompletion (WO) and BarcodeWorkOrderCompletion (BWO) which together we use in a running example.

BWO has associated domain variables as follows:

\[
\text{BWO}.\text{HAS}.\text{USE} = \{ \text{AssetNo}, \text{AssetDescription}, \text{HistoryTypeDescription}, \text{HistoryTypeCode}, \text{DoubleTimeHrs}, \text{ExtraAmount}, \text{InternalExternalFlag}, \text{NormalTime}, \text{TimeAndAHalf}, \text{TradespersonNo}, \text{WorkDate}, \text{WorkOrderNo}, \text{TradeDescription}, \text{TradeCode}, \text{TradespersonName}, \text{ActionDetails}, \text{WorkDescription}, \text{ArrivalDateTime}, \text{CompletionDateTime} \}.
\]

WO has all BWO’s domain variables and some others:

\[
\text{WO}.\text{HAS}.\text{USE} = \text{BWO}.\text{HAS}.\text{USE} \cup \{ \text{HistoryDescription}, \text{ServiceDate}, \text{CostCentre}, \text{ExpenseCode}, \text{ServiceDate}, \text{WorkComments}, \text{WorkCompleteStatus} \}.
\]

(\( \text{WO}.\text{HAS}.\text{USE} \cap \text{BWO}.\text{HAS}.\text{USE} = \text{BWO}.\text{HAS}.\text{USE} \))

Thus:

\[
w_a(\text{WO}, \text{BWO}) = 19/26 = 0.73
\]

\[
w_a(\text{BWO}, \text{WO}) = 19/19 = 1
\]

\[
w(\text{WO}, \text{BWO}) = 19/26 = 0.73
\]

C. Change Propagation Analysis

Given a change request for modifying (fixing a bug or an enhancement) a UIC, we query other UICs which most likely will be affected by the given change.

When changing a given user interface component \( c \in C \), we define a function

\[
\text{AFC} : (C \times C \to \mathbb{R}) \times \mathbb{R} \to (C \times C)
\]

which generates a relation between a component \( c \) and other components most likely affected by a change to \( c \). That is \( \text{AFC}(f, \lambda) \) represents the set of most likely affected components by a change to component \( c \), defined as

\[
\text{AFC}(f, \lambda) = \{ (c, c') | c, c' \in C \land c \neq c' \land f(c, c') > \lambda \}
\]

where \( f \) is a function measuring the level of coupling between two components and the \( \lambda \) is a given threshold.

Using Definition 2 and 3, \( c.\text{AFC}(w_a, \lambda) \) and \( c.\text{AFC}(w, \lambda) \) represent the probable affected components based on conceptual couplings.

III. Case Study

Newly proposed approach requires empirical evaluation. We present the results of a case study aimed at comparing with the known history-based change coupling derived from the version control [24, 26].

In specific we seek to answer the following questions for our case study system:

- **Correlation.** To what extent can the conceptual coupling between pairs of UICs be correlated with history-based change coupling?
• **Error prevention.** Given a single transaction involving modification of multiple UICs, if a single UIC is missing from the transaction, how reliably can conceptual coupling be used to find the missing component?

• **Estimating change scope.** Given a change to a single component, how reliably does conceptual coupling determine what other components will most likely be affected by the change?

To answer these questions we use Zimmermann’s approach to calculating evolutionary coupling [26] as a basis for evaluating the reliability of predictions derived from conceptual coupling (see below).

### A. Case Study System: BEIMS

The software examined in this case study is a facility management system called BEIMS³. Mercury Computer Systems designed and developed the first version of BEIMS in 1989 using a 4GL. In 1998 the fifth generation of BEIMS was redeveloped for Windows platforms. It has been extended with more than 100 individual subsystems and custom developed programs. For our case study we chose the five core subsystems in BEIMS installed and used by all BEIMS clients. As demonstrated in Table I these together contain more than 100,000 lines of code. We looked at the maintenance history of these programs and studied how they have evolved over the last 12 years.

Given the rich maintenance history of BEIMS, we can evaluate to what extent the conceptual coupling between UICs correlates with what can be predicted from the change history of the software.

<table>
<thead>
<tr>
<th>Subsystem name</th>
<th>ID</th>
<th>Source code lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asset Management System</td>
<td>AMS</td>
<td>15,700</td>
</tr>
<tr>
<td>Cost Control System</td>
<td>CCS</td>
<td>10,959</td>
</tr>
<tr>
<td>Information Setup System</td>
<td>ISS</td>
<td>29,330</td>
</tr>
<tr>
<td>Planned Maintenance System</td>
<td>PMS</td>
<td>13,857</td>
</tr>
<tr>
<td>Work Order System</td>
<td>WOS</td>
<td>34,164</td>
</tr>
</tbody>
</table>

**TABLE I: BEIMS Core Subsystems**

### B. History-Based Change Coupling

The history of source code changes can reveal patterns in modifying software components, the result of software maintenance activities.

Let \( T \subseteq 2^{UIC} \) be a set of *transactions* where each transaction is defined by a set of *changed components*. Following standard data mining approaches [2, 26], define an *(association)* rule \( x_1 \Rightarrow x_2 \) for two disjoint sets \( x_1 \) and \( x_2 \) *(interpretation:* if a programmer changes \( x_1 \) then she has also changed \( x_2 \)). The *frequency* of a (changed component) set \( x \) in \( T \) is defined as

\[
freq(T, x) := |\{ t | t \in T, x \subseteq t \}|
\]

³The Building and Engineering Information Management System (BEIMS) has more than a third of the market share for facility management systems using by Australian and New Zealand universities, hospitals and casinos.

The *confidence* of a rule \( x_1 \Rightarrow x_2 \) (interpreted as the strength of the rule) is defined as

\[
conf(T, x_1 \Rightarrow x_2) := \frac{freq(T, x_1 \cup x_2)}{freq(T, x_1)}
\]

For assessment of results we follow standard definitions of *precision* \( (P_q) \), the percentage of a returned answer which was expected, and *recall* \( (R_q) \), the percentage of an expected answer which was returned [12]:

\[
P_q = \frac{|A_q \cap E_q|}{|A_q|}, \quad R_q = \frac{|A_q \cap E_q|}{|E_q|}
\]

In order to calculate the precision and recall for all queries for a given subsystem, we take the mean value of the precision and recall of individual queries:

\[
P_M = \frac{1}{n} \sum_{i=1}^{n} P_{q_i}, \quad R_M = \frac{1}{n} \sum_{i=1}^{n} R_{q_i}.
\]

### C. Evolutionary Coupling in BEIMS

For BEIMS, Microsoft Source Safe is used for source code version control. We used the sliding window technique proposed by Zimmermann and Weißgerber to recover transactions from Source Safe [27], since it does not track which files have been modified in a transaction (committed changes in conjunction). The history of changes to a given file can be derived from Source Safe as a set of change log records whereby each record represents a single *check in* (commit) command and contains user name, comment (message) and the differences between two subsequent revision of a file.

At the time of analysis there were 78,632 change log records for the five subsystems of BEIMS. Some of these records are the result of a labelling action in Source Safe. Labels have been used to tag all files with a given time (typically a released version of BEIMS) for the purpose of creating branches. Labelling records are not related to any modification to file contents and the transactions derived from them do not imply any code change coupling between files. We are only interested in the coupling between source code files result of conjunction maintenance, therefore we removed the labelling records to avoid false positive results. The remainder is 10,912 records, yielding 4,456 transactions.

### D. Domain-Based Analysis of BEIMS

We analysed the behaviour of all UICs for the five subsystems, only based on information provided in the software functional specification. The functional specification describes a subsystem from the perspective of a domain user including the behaviour of each screen at the domain level, that is, actions, interactions and provided information. The functional specification is derived from existing user manuals and expert user knowledge about system behaviour, as described earlier [1]. The analysis result is collected in the form of a dependency matrix (Definition 1) consisting of 68 UICs and 381 domain variables whereby for 731 elements \( M_{c,v} = 1 \). From the dependency matrix, we derived symmetric and asymmetric weights for pairs of UICs in each subsystem as summarised...
in Table II with the aggregated statistical information for each subsystem.

<table>
<thead>
<tr>
<th></th>
<th>Min</th>
<th>Max</th>
<th>Mean</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMS</td>
<td>$w_a$</td>
<td>0.00</td>
<td>1.00</td>
<td>0.11</td>
</tr>
<tr>
<td></td>
<td>$w$</td>
<td>0.00</td>
<td>0.50</td>
<td>0.23</td>
</tr>
<tr>
<td>CCS</td>
<td>$w_a$</td>
<td>0.00</td>
<td>1.00</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td>$w$</td>
<td>0.00</td>
<td>0.25</td>
<td>0.10</td>
</tr>
<tr>
<td>ISS</td>
<td>$w_a$</td>
<td>0.00</td>
<td>1.00</td>
<td>0.44</td>
</tr>
<tr>
<td></td>
<td>$w$</td>
<td>0.00</td>
<td>0.88</td>
<td>0.23</td>
</tr>
<tr>
<td>WOS</td>
<td>$w_a$</td>
<td>0.00</td>
<td>1.00</td>
<td>0.24</td>
</tr>
<tr>
<td></td>
<td>$w$</td>
<td>0.00</td>
<td>0.77</td>
<td>0.12</td>
</tr>
</tbody>
</table>

**TABLE II: Conceptual couplings**

E. Results: Correlation

The experimental hypothesis is that there is a correlation between evolutionary and conceptual coupling.

We evaluated the hypothesis in two stages. Firstly we examined the evolutionary coupling of UIC pairs with respect to conceptual coupling. Secondly we measured the correlation coefficient between the asymmetric/symmetric weight functions and the value of $conf$.

1) Average $conf$ versus conceptual coupling: In the first stage, we grouped the UIC pairs in all BEIMS subsystems by conceptual coupling—first asymmetric weight, then symmetric weight—and measured the average confidence ($conf$) for association rules between all pairs in each group.

Figure 1a illustrates the relationship between $conf$ and asymmetric weight. Each group $G_n$ consists of pairs $(c, c')$ where $n - 1 < w_a(c, c') * 10 \leq n$.

Figure 1b illustrates the relationship between $conf$ and symmetric weight. Each group $G_n$ consists of pairs $(c, c')$ where $n - 1 < w(c, c') * 10 \leq n$.

In general average $conf$ increases with respect to asymmetric or symmetric weight, i.e., pairs with stronger conceptual coupling have greater confidence level for evolutionary coupling. There are exceptions to this trend that we will discuss later in this section.

Table III shows the number of pairs in each group. In general the number of pairs decreases as conceptual coupling increases, i.e., most pairs are weakly conceptually coupled, and few pairs are strongly conceptually coupled.

As illustrated in Figure 1, the first exception in the trend of average $conf$ increasing with conceptual coupling is at $G_0$. In comparison to nearby groups ($G_1$, $G_2$) the expected average $conf$ for $G_0$ is a value close to zero; however, the actual value is 0.17 (true in both Figures 1a and 1b). That there are 414 asymmetric pairs (Figure 1a) in this group, suggests that not all change couplings can be derived from conceptual coupling. Change logs show some changes to the code are motivated by refactoring, and initiated by programmers where there are no bug reports or enhancement requests.

The second exception is visible in Figure 1b, which shows a correlation between the symmetric weight and average $conf$ maximised at $w \leq 0.6$ ($G_6$). The exceptions to this trend are
in $G_8$ and $G_9$, each with a single pair (Table IIIb) and lower confidence than $G_6$.

The first pair is \langle WorkOrderCompletion (WO), BarcodeWorkOrderCompletion (BWO) \rangle. BWO provides the same functionality as WO but instead of typing the work order information, a barcode scanner is used to read the work order and fetch the data. BWO and WO have a clear overlap in their functionality and a lot of duplicated source code. However, WO has been much used by BEIMS users, and subjected to more refinement and minor enhancements. Change logs show that BWO is more often ignored for minor BEIMS revisions, and more subject to changes in major revisions.

The second pair is \langle MaintenancePlan, AssignTaskToAssets \rangle. Both these UICs enable users to add and manage jobs related to assets. However, these UICs provide two different presentations of similar information: the first UIC allows the user to review and manage the jobs in bulk using a calendar view; the second UIC is more focused at the detailed level of individual tasks. The behavioural difference between these components is the main cause of disjoint sets of changes to their source code.

2) Correlation coefficient: In the next stage, we examined each subsystem individually, and measured the correlation between conf and weights. We used Pearson’s correlation coefficient $r_{x,y}$ as a measure of linear dependence between two variables $x$ and $y$, giving a value between $+1$ and $-1$ inclusive [22].

Table IV shows on average there is a positive correlation between weights and conf, however the correlation is not the same for all subsystems. The behavioural and architectural characteristics of these subsystems affect the pattern which in their source code is changed.

For example, Information Setup System (ISS) is in charge of defining primary data entities in BEIMS, leading to individual UICs containing detailed information unique to each UIC. Also there are number of cases where two UICs holding information about master-detail data entities. Such cases reduce the symmetric weight between UICs; however, the asymmetric weight function can reflect such relationships where one component holds the superset domain variables of another component.

The other example is Planned Maintenance System (PMS), where there is a negative correlation of $-0.04$ between the symmetric weight and conf, suggesting that evolutionary coupleings can not be derived from the symmetric weight function. Though, the asymmetric weight function for the same UIC pairs has a correlation of 0.42.

In summary, for all five subsystems, confidence level of evolutionary coupling and asymmetric weight are correlated with average Pearson’s correlation coefficient of 0.48. Notably, the level of correlation is not the same for all subsystems, and the variation is greater for symmetric weight.

\begin{table} [h]
\centering
\begin{tabular}{|c|c|c|}
\hline
Subsystem & $r(w, \text{conf})$ & $r(w_a, \text{conf})$ \\
\hline
AMS & 0.341 & 0.392 \\
CCS & 0.703 & 0.45 \\
ISS & 0.199 & 0.534 \\
PMS & -0.042 & 0.422 \\
WOS & 0.481 & 0.61 \\
\hline
Average & 0.3364 & 0.4816 \\
\hline
\end{tabular}
\caption{Correlation coefficient} \label{tab:corr_coef}
\end{table}

F. Results: Error Prevention

In this section we evaluate to what extent conceptual coupling can be used to prevent software bugs where a programmer changes multiple UICs but misses a single component.

If $\Psi \subset UIC$ represents a set of UICs modified in a given transaction, for each component $c \in C$ we test a query as $Q = \Psi - \{c\}$.

For a given $Q$ (and some suitable $f$ and $\lambda$), the prediction derived from conceptual coupling is: $A = \bigcup_{x \in Q} x.AFC(f, \lambda)$.

Our experimental hypothesis is that for some suitable $f$ and $\lambda$, $A = \{c\}$ always.

In order to evaluate this hypothesis, we tested queries for five subsystems of BEIMS using $f = w$ and $f = w_a$ (Definitions 2 and 3). As a benchmark for evaluation, we compared the effectiveness of conceptual coupling to evolutionary coupling with respect to error prevention, repeating the experiment above using $f = \text{conf}$.

In addition, we found cross-program transactions containing changes to multiple subsystems. Such transactions are the result of logical coupling between different BEIMS subsystems. These couplings are not visible at the source code level as there is no code dependency between these subsystems, but, more abstractly, these subsystems are connected at the domain level [6]. As all these subsystems are maintained by a single programming team, it is often the case that programmers are aware of such dependencies.

In the first stage, we tested the queries using the asymmetric weight function ($f = w_a$). Table V shows the results for the five systems. To avoid many false positive results (false warning), we set $\lambda = 0.9$, yielding an average recall of 0.09 and precision of 0.66. This means for only one in 11 queries the $AFC(w_a, 0.9)$ warns the programmer about the missing UIC, and more than half of the results are valid warnings. However, the detailed results show that for the threshold of 0.9, no results were returned for the CCS subsystem. The highest threshold that we can get to cover all subsystems is $\lambda = 0.3$, with the average recall of 0.52 and precision 0.32.

In the second stage, we tested the queries using $AFC(w, \lambda)$ (i.e. based on the symmetric weight function). Notably the maximum value for $w$ in all subsystems is 0.88 (Table II), so we selected the maximum threshold as $\lambda = 0.7$, yielding average recall of 0.05 and precision of 0.8. This means for only one in 20 queries $AFC(w, 0.7)$ returns any missing UIC, and 80% of the raised warnings to the programmer are true missing UICs. However, as represented in Table VI there are no results
for three subsystems (AMS, CCS, ISS). The highest threshold that can be achieved to cover all subsystems is \( \lambda = 0.2 \), with average recall of 0.46 and precision 0.35.

Finally, as a benchmark for evaluation, we tested the queries based on \( AFC(\text{conf}, \lambda) \). The results are represented in Table VII. For a strong threshold of 0.9, \( AFC(\text{conf}, 0.9) \) returns one out of 34 missing UICs with the precision of 0.93, however, the results only include the WOS, ISS and cross-program queries. To achieve a result for all subsystems the maximum threshold of 0.3 yields recall of 0.48 and precision 0.54.

In summary, for all five subsystems, on average 46% of errors arising from imperfect change propagation can be avoided (Table V) using only domain-level information. This is a promising result in compare to the 48% average error prevention using evolutionary coupling. In addition, comparison between results for asymmetric and symmetric weight functions suggests that the asymmetric weight function provides better recall; however, more precision can be achieved at the expense of recall using the symmetric weight function.

G. Results: Estimating Change Scope

In this section we evaluate to what extent change propagation can be estimated using conceptual couplings. Based on evolutionary couplings a set of components can be derived which are coupled to a given UIC with the confidence greater than a given threshold. The hypothesis is that such a set can be derived from the conceptual couplings.

For \( c \in C \), we define a query as a tuple \( q = \langle c, \lambda \rangle \) where \( \lambda \) is the minimum required level of confidence. The expected set of affected components by a change to \( c \) can be estimated using conceptual couplings. Based on evolutionary couplings a set of components can be derived.

\[
E = c.AFC(\text{conf}, \lambda)
\]

We used asymmetric and symmetric weight functions (Definitions 2,3) with the same threshold as \( \text{conf} \) to derive the following answers:

\[
A = c.AFC(w, \lambda), \quad A_a = c.AFC(w_a, \lambda)
\]
Asymmetric function

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>R_M</th>
<th>P_M</th>
<th>F_b</th>
<th>R_M</th>
<th>P_M</th>
<th>F_b</th>
<th>R_M</th>
<th>P_M</th>
<th>F_b</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMS</td>
<td>0.33</td>
<td>1.00</td>
<td>0.67</td>
<td>0.48</td>
<td>0.68</td>
<td>0.89</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>CCS</td>
<td>0.07</td>
<td>0.25</td>
<td>1.00</td>
<td>0.04</td>
<td>0.44</td>
<td>0.67</td>
<td>0.12</td>
<td>0.45</td>
<td>0.94</td>
</tr>
<tr>
<td>ISS</td>
<td>0.75</td>
<td>0.25</td>
<td>1.00</td>
<td>0.84</td>
<td>0.72</td>
<td>1.00</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>PMS</td>
<td>0.50</td>
<td>0.63</td>
<td>0.50</td>
<td>1.00</td>
<td>0.41</td>
<td>1.00</td>
<td>0.92</td>
<td>0.52</td>
<td>1.00</td>
</tr>
<tr>
<td>WOS</td>
<td>0.50</td>
<td>0.63</td>
<td>0.50</td>
<td>1.00</td>
<td>0.41</td>
<td>1.00</td>
<td>0.92</td>
<td>0.52</td>
<td>1.00</td>
</tr>
<tr>
<td>Average</td>
<td>0.29</td>
<td>0.44</td>
<td>0.75</td>
<td>0.53</td>
<td>0.53</td>
<td>0.83</td>
<td>0.67</td>
<td>0.66</td>
<td>0.97</td>
</tr>
</tbody>
</table>

Symmetric function

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>R_M</th>
<th>P_M</th>
<th>F_b</th>
<th>R_M</th>
<th>P_M</th>
<th>F_b</th>
<th>R_M</th>
<th>P_M</th>
<th>F_b</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMS</td>
<td>0.33</td>
<td>0.82</td>
<td>0.67</td>
<td>0.33</td>
<td>0.82</td>
<td>0.67</td>
<td>0.75</td>
<td>1.00</td>
<td>0.75</td>
</tr>
<tr>
<td>CCS</td>
<td>0.33</td>
<td>1.00</td>
<td>0.67</td>
<td>0.07</td>
<td>0.25</td>
<td>1.00</td>
<td>0.04</td>
<td>0.45</td>
<td>0.94</td>
</tr>
<tr>
<td>ISS</td>
<td>0.75</td>
<td>0.25</td>
<td>1.00</td>
<td>0.84</td>
<td>0.72</td>
<td>1.00</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>PMS</td>
<td>0.75</td>
<td>0.25</td>
<td>1.00</td>
<td>0.84</td>
<td>0.72</td>
<td>1.00</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>WOS</td>
<td>0.50</td>
<td>0.71</td>
<td>0.82</td>
<td>0.50</td>
<td>0.71</td>
<td>0.82</td>
<td>0.60</td>
<td>0.90</td>
<td>0.60</td>
</tr>
<tr>
<td>Average</td>
<td>0.15</td>
<td>0.73</td>
<td>0.40</td>
<td>0.15</td>
<td>0.73</td>
<td>0.40</td>
<td>0.50</td>
<td>0.71</td>
<td>0.82</td>
</tr>
</tbody>
</table>

TABLE VIII: Result: Estimating the scope of change propagation

Where $Q$ is the set of queries for a system, we measured the percentage of the queries where our approach can give at least one recommendation $Q^*$ as $\text{feedback} = |Q^*|/|Q|$.

Table VIII shows the results for the five subsystems of BEIMS with three threshold levels 0.7, 0.4, and 0.1, each resulting in a different level of $\text{conf}$ for change propagation.

A strong threshold of $\lambda = 0.7$ yields empty query sets for three subsystems AMS, CCS and PMS, where the maximum $\text{conf}$ values are 0.33, 0.60, 0.66 respectively. For the two other subsystems $AFC(w, \lambda)$ returns at least one UIE for three out of four queries (feedback=0.75), and on average for each query 29% of answers were correct with precision of 44%.

For a threshold of $\lambda = 0.1$, there are queries derived from all subsystems. Using the asymmetric weight function, 97% of these queries have been answered with an average recall of 67% and precision of 66%, meaning more than half of the expected answers been returned.

Using symmetric weight function with $\lambda = 0.1$ precision improves to 71% with the cost of reducing both feedback and recall. The results suggest that the asymmetric weight function is more effective for high and midrange thresholds, and the symmetric function can be used with lower thresholds. This is a tradeoff between precision and recall in favor of precision.

The predictive power of conceptual coupling is affected by the confidence level and the given threshold to the weight functions. Figure 2 shows the comparison between recall and precision for confidence levels 0.1, 0.4 and 0.7. In order to find the impact of the different thresholds on the results, the queries have been answered using the asymmetric weight function with a range of answer thresholds from 0 to 1 exclusive (horizontal axis).

For all query thresholds, increasing the answer threshold reduces the recall and increases the precision. For queries with stronger confidence level, the recall suddenly drops after some given threshold; however, for the queries created from low confidence level this change is gradual.

In summary, the results show that change propagation can be estimated based on conceptual coupling. Using the asymmetric weight and a threshold of 0.1, we achieved up to 92% correct estimation of change propagation in WOS, and in average 67% correct estimation for all five subsystems. Using the symmetric weight for estimating the change propagation, in average improves the precision in cost of recall, making the symmetric weight a more preferable choice where high level of accuracy is required.

H. Discussion

The results of this case study show how change propagation between software domain-level components can be predicted based on conceptual couplings. Also we demonstrated that such prediction could assist in avoiding bugs arising from imperfect software alteration.

We measured the correlation between conceptual and evolutionary coupling. The result suggests that there is a positive correlation between conceptual and evolutionary coupling for all five subsystems. For the symmetric function the correlation is high for two subsystems while it is low for the other ones.
The results in sections III-F and III-G show that the asymmetric weight function is more suitable for predicting change propagation at higher confidence levels, whereas in contrast the symmetric weight function is more suitable at lower confidence levels where it yields higher precision.

The effectiveness of this approach depends on the type of queries, and the behavioural characteristics of the system, and even though we used only domain information for change propagation analysis, the results are seemingly close to history-based change coupling, suggesting that a domain-based approach is a plausible alternative.

IV. RELATED WORK

To our knowledge not much work has been done in studying the impact of domain-based couplings between software components on change propagation. Poshiyvanyak and Marcus have defined many various coupling measures and shown that conceptual coupling can capture new dimensions of coupling which are not captured by existing coupling measures and so can be used to complement existing metrics [16,20]. Their work was for object oriented systems, and we recognize that much of what they discuss can be applied to large enterprise systems. However, their approach requires availability of the source code, and may not be usable by non-technical domain experts.

The logical couplings discussed by Gall, Hajek and Jaza-yeri have explained software evolution using product release history and change reports [6]. Their approach determines dependencies at the coarse granularity of programs and modules, and assumes longevity and adequate information about the maintenance history. Robbes, Pollet and Lanza extend logical coupling by introducing several new ways of measuring semantic changes and information recorded in the IDE [21].

There is a strong body of literature in studying software evolution and impact analysis based on mining software change history [3,5,7,8,13,14,17,23–26]. Such approaches, assuming the availability of the code change history, are effective in change propagation analysis. However, they cannot be applied to new systems where there is no adequate maintenance history.

Source code analysis is another established approach for change impact analysis. Petrenko and Rajlich discuss variable granularity for change impact analysis [19]. Godfrey and Tu investigate the evolution of open source systems [10]. Program slicing has been introduced as an aid to programmers performing software maintenance [9,11,15,18]. Such approaches require in-depth knowledge of software engineering, and access to source code, thus are not usable by non-technical domain experts.

V. CONCLUSIONS AND FURTHER WORK

In this paper we introduced a novel approach for change propagation analysis based on software domain level information. The aim is to provide a method to estimate the impact of a change to a system independent from source code, maintenance history and assuming limited knowledge of software engineering. Such an approach would be usable by system analysts and domain experts, and applicable to software environments where source code analysis or history-based change propagation analysis is not available.

Our approach is based on the hypothesis that change propagation between system domain components can be derived from the relationships between information available at the domain level. In our work we defined conceptual couplings as a measurement for commonality of data derived from software domain-level components.

With a case study of an enterprise system, we demonstrated how conceptual couplings can assist in preventing errors in software maintenance or predicting the scope of change propagation. The evaluation is performed based on the confidence level of evolutionary couplings derived by a well known technique for mining version history. The result suggests that for an enterprise system with many user interface components (e.g., management information systems (MIS)), the domain-based approach can be used for change propagation analysis, gaining practicality possibly at the expense of accuracy.

This approach is not applicable to systems which provide their functionality throw few UICs, or their functionality cannot be analysed by domain users, e.g., a computer operating system.

We envisage that the domain-based approach might also be used to complement history-based techniques and source code analysis methods, in a hybrid approach. Our case study shows that although overall evolutionary coupling and conceptual coupling are correlated, there are some exceptions. In such cases, each coupling measurement may reveal different aspects of a system’s behavioural or architectural characteristics. In this work we did not evaluate these cases qualitatively looking for complementarity, although we believe complementarity may well exist. A study evaluating complementarity is a clear candidate for future work.

For this method prediction quality depends on the optimal choice of the λ parameter: as λ increases, precision increases and recall decreases. Automatically finding the optimum value of λ for any given system will be necessary, and therefore the subject of future work.

We applied asymmetric and symmetric weight functions to different tasks and compared the results, demonstrating that asymmetric weight has higher recall (provides more results) whereas symmetric weight has better precision. However, whether these functions can be used in conjunction to achieve even better efficiency is a subject for further work.

We also propose to extend this work using information mined from bug reports and support records, leading to yet other forms of coupling with further potential benefits. Descriptive information recorded by users about application bugs, and required enhancements, may reveal complementary couplings between software components.

We have shown that there are statical or average correlations between predictions derived from domain-based and heuristic analyses. However the variation in the quality of individual predictions seems significant. A qualitative evaluation would
yield insight into the underlying causes of these variations.
Overall, the positive average correlation between domain-based and heuristic analysis results suggests that conceptual coupling at the software domain level can predict evolutionary coupling.

ACKNOWLEDGMENT
Part of Amir Aryani’s contribution to this work has been funded by an Australian Postgraduate Award (APA).
The authors would like to thank Mercury Computer Systems (Australia) Pty Ltd. for allowing access to BEIMS maintenance logs and domain information.

REFERENCES