Assessing Impacts of Changes to Business Rules through Data Exploration

Suzanne M. Embury, David Willmor and Lei Dang
School of Computer Science, University of Manchester,
Oxford Road, Manchester, M13 9PL, United Kingdom
s.m.embury|d.willmor|l.dang@cs.man.ac.uk

Abstract

The benefits of impact analysis in the maintenance and evolution of software systems are well known, and many forms of impact analysis, over different software life cycle objects, have been proposed. However, one form of impact from software change has yet to be explored by the research community: these are the impacts of changes to software on data. In particular, when the business rules enforced by a system change, it may be necessary to perform some clean-up or transformations on persistent data, in order to bring it into line with the new system functionality. Alternatively, the proposed new rule may itself have to be modified, if the data impacts are too costly to address. In this paper, we show how such impacts can arise, and propose an approach to assisting in their identification through user-driven exploration of hypothetical change-impact scenarios.

1. Introduction

One of the key roles of a business information system is to assist its owning organisation in enforcing business policy and complying with the business rules agreed upon by the organisation. A business rule is a statement that defines or constrains some aspect of an organisation [6]. For example, a retail organisation might have rules describing the discounts that can be offered to regular customers or the conditions under which further credit will be refused. Despite the name, business rules are not limited to business organisations. Governments use business rules to determine who shall receive social security payments, for example, while hospitals have rules regarding who may perform certain medical procedures and the circumstances under which certain drugs can be prescribed.

As can be seen from these examples, business rules are typically relatively small units of functionality. They are also often self-contained and apply across a range of business processes. Consider a set of rules describing how sales tax should be calculated for different kinds of products. These rules must be enforced by the sales processing systems, but also by any electronic catalogue systems, by marketing systems (when determining the marketing strategy for a new product), by stock management systems (where the organisation is purchasing rather than selling items) and by tactical planning systems (where sales tax rules may affect which items are economic for sale in an area and which are not). The wide scope of applicability of many business rules means that a single rule must typically be implemented by several computing systems, and within several components of each individual computing system [5]. Moreover, a given organisation will typically have hundreds or thousands of such rules to enforce.

Evolution of business rules is therefore a costly exercise [11]. Unfortunately, business rules tend to evolve rapidly. Keeping up with competitors often requires that new schemes be implemented rapidly — new account types, for example, special offers or new types of purchasing plan. All these require evolution of business rules. A small number of researchers have been investigating the tools and techniques software developers need in order to implement business rule changes. For example, impact analysis tools can be used to identify the software components that must be changed when a new business rule is to be added to a system [5]. Tools also exist which can identify conflicts between rules, so that new and evolved rules can be checked for consistency with the existing rule set.

However, a third type of impact exists, which has not yet been studied. These are data level impacts, where new rules force a reinterpretation of the data that describes the current and past state of the organisation. This is perhaps best explained through an example. Suppose that a new set of health and safety regulations are imposed by government. These regulations manifest themselves as business rules describing (amongst other things) the toxicity levels associated with certain product ingredients. When these rules are imposed on an existing product base, the organisation may discover that a number of their products are raised to a level of toxicity which requires a more expensive form of packaging than previously needed. We call this a data-level impact

0-7695-2703-5/06/$20.00 (c) IEEE

because it is manifest not at the software level but in terms of the changes required to the data (in this case, the product catalogue) in order to bring it into line with the evolved rules. In our example, the company has no choice but to abide by the regulations and must either upgrade their packaging or find less toxic ingredients for the problem products. Where rule evolution arises from business pressures rather than regulatory ones, however, the company may decide that the proposed evolution is not worth the attendant costs and may reject it outright or else consider a modified form of the originally proposed change. Discovering these cost implications before significant effort has been put into implementing the new rule set is clearly of vital importance.

In this paper, we examine this new form of software change impact in the context of business rule changes. We also describe the $\mathrm{WiA_{BR}}$ system, which aims to assist system owners in determining the impacts of proposed changes to constraint business rules [14], in terms of the databases associated with the systems. The intention is that $\mathrm{WiA_{BR}}$ would sit between the business-oriented decision making processes that lead to changed requirements for software systems and the code-level impact analysis techniques that are applied by the software developer. $\mathrm{WiA_{BR}}$ can therefore be seen as a "requirements-level" impact analysis technique, which shares some features of the more traditional source-code level impact analysis tools.

The remainder of this paper is organised as follows. In Section 2, we give an overview of research into software change impact analysis, and use this (in Section 3) to determine the tool functionality needed to assist in the identification of data-level impacts from business rule changes. In Section 4, we describe how these requirements can be implemented in practice, and in Section 5 conclude.

2. Software Change Impact Analysis

One of the challenges in maintaining and evolving software systems is in determining the full scope of a change. What appears to be a single, simple change to functionality may in fact require a great many interconnected changes to the source code of the system, as well as to other software life cycle objects. If the developers charged with implementing the change fail to identify all the components impacted by it, then the change will be applied in an inconsistent manner and anomalous behaviour will result.

Software change impact analysis (SCIA) tools attempt to address this problem by exploiting the exhaustive search capabilities of software to identify the components affected by a proposed change [1, 13]. Such tools come in many varieties, depending on the kinds of software life cycle objects that they can reason over and the types of proposed change they can work with. Probably the most common form of SCIA is that which operates at the source code level [9, 10, 16]. A change is to be made to one part of the code (for example, to a specific method or module) and the impact analysis tool determines the set of such components within the system that could be affected by it. This is typically done through the calculation of *dependencies*, based on an understanding of the semantics of the programming language in question. These dependencies are typically the same ones that are used in other forms of software engineering tool (especially program slicers and other program comprehension tools): that is, data dependencies, control dependencies and call dependencies [12].

While these tools are very useful, they cannot be applied until the initial change has been understood well enough to identify a starting set of software components to change. This is much too late, if the change originates as a request for a change in functionality (i.e. a change to the specification). Managers need to be able to assess the impacts of such changes quickly, before the details of how to implement them have been worked out, so that appropriate decisions can be made about which changes to implement and which are not sufficiently cost-effective [8]. Unfortunately, impact analysis at higher levels of abstraction than source code is extremely challenging, since the artifacts that the tool must analyse are often unstructured or semi-structured, and are written in natural rather than formal languages.

To perform SCIA over documents, such as specifications and design reports, it is necessary to use textual analysis techniques and information retrieval techniques. Other authors have shown how to compute impacts based on the high-level models that are commonly used to represent software requirements and design. For example, Briand *et al.* have shown how to calculate the impacts of changes to UML diagrams on other parts of the UML model [2]. Although UML does not have a complete formal semantics, it is sufficiently well-defined to allow rules to be developed which describe how a particular type of change (for example, the addition of a new message send to a sequence diagram) can disturb the consistency of other UML elements (such as class methods in class diagrams). de Boer *et al.* adopt a similar approach to the identification of impacts to enterprise architectures [3]. Again, the architectural models are expressed in a formal language that has a sufficiently rich semantics to allow impacts to be calculated with a fair degree of accuracy.

None of these SCIA tools, however, consider the effects of changes on data. Almost all modern information systems are based around one or more databases, which manage large amounts of data describing both the current and (sometimes) the past history or the organisation. If a change is proposed which would require large subsets of the data to be processed (cleaned) by hand, then it may not be cost-effective, regardless of the benefits it promises to bring. Or, sometimes, the impacts of a change on data can highlight

incompleteness in the change itself. For example, a change to discount policies may result in certain kinds of goods being sold at a loss. Once this is known, the initial change can be amended so that the problem product types are excluded from the most generous discount deals.

In the next section, we will examine this new form of impact, and derive the features required of a system to assist in their discovery.

3. Data-Level Impacts from BRs

Based on the characterisation provided by Queille *et al.* [13], we can expect an impact analysis method to have the following basic components :

- some representation of the system which is to be the subject of the analysis (the system model);
- some means of specifying proposed changes to the system;
- some means of representing the impacts of proposed changes; and
- some means of determining how changes propagate through the system model to impact other components.

For example, a classic source-code level IA tool will use a collection of dependency graphs as its system model, while the proposed change might be represented as the subset of the nodes in the graphs (i.e. the statements) which are known to be involved in the change. The graph semantics are used to identify how the change propagates between nodes, and the resulting impact set takes the same form as the proposed change — a list of impacted nodes [1]. For other forms of SCIA, the system model might take the form of textual similarity links between documents or UML models. Changes might be represented as pointers to amended paragraphs, or as the difference between two versions of a document or model. Impacts may be coarse or fine grained, indicating for example that some entire document must change (in some unspecified way) or that a specific method must be added to a class in a UML model.

What form do these components take when we are concerned with the impacts of business rule change at the data level? Clearly, the model of the system is the collection of business rules currently being enforced by the software plus the data to which they are applied. The type of change to be analysed in our specific context can take one of three forms:

- the addition of a new business rule to the system;
- the deletion of an existing rule from the system; or
- the amendment of some aspect of an existing rule.

So far, this much is straightforward. The more interesting question is: what is an impact, in this context? Of course, changes to business rules have an impact on the specification, design and source code of a software system, just as any change to functionality does. Such impacts have been studied elsewhere [5]. Here, our goal is to identify impacts on other business rules (i.e. where must changes be made to other business rules to accommodate the proposed change) and impacts on data (i.e. which records must be cleaned or deleted in order to be compatible with the change).

The exact nature of these impacts depends on the semantics of the business rule that is changed. The term *business rule* is very broad and encompasses a wide range of rule-like semantics, each of which must be considered separately when the impacts of a change are to be deduced. There have been several attempts to classify business rules [6]. We will adopt the classification proposed by Shao and Pound, in which three types of rule are distinguished: derivation business rules, constraint business rules and event business rules [14]. We will now consider the kind of impact associated with each of these classes of rule.

3.1. Derivation Business Rules

This kind of rule describes how a data item relates to, or can be computed from, other data. For example, rules describing how the total bill for an order should be computed, including sales tax and any discounts, are classic examples of derivation rules.

A data-level impact from a change to such a rule would occur when an incorrect or inappropriate value would be calculated by the new rule for some data item. For example, a rule giving a discount on products with certain features may be found in practice to apply to a much wider range of products than originally envisaged by the designers of the new rule. One way to fix this is to make the conditions of the rule more strict, of course, but another option is to change the classification of those products that ought not to be subject to the new rule. In this latter case, the proposed change to the business rule must be augmented with associated changes to data before the initial evolution requested can be considered complete.

Just as with traditional impact analysis, however, we must consider the *ripple effect* of change propagation. A proposed business rule evolution may not result in any negative data impacts directly, but since the values produced by one derivation rule may be used as inputs to other rules negative data impacts can still result. In such a case, however, we have even more options for propagating the change, in order to remove the negative effects. If there is a chain of business rule applications $< br_1, \ldots, br_n >$ where br_1 is the evolved rule and br_n is the rule which directly results in the negative data impact, then we can potentially remove

the undesirable behaviour by making further changes to any of the rules in the chain, or the affected data, or any combination of such changes.

3.2. Constraint Business Rules

This kind of rule places restrictions on the state of the organisation, as represented by the data. A typical example of a constraint business rule is one which specifies that customers who have failed to meet the deadline for payment more than twice are not eligible for membership of the organisation's loyalty scheme.

Constraint business rules impact the data of a system in a rather different way to that just described for derivation rules. Instead of computing new values, changes to constraint business rules mean that certain data values that were formerly considered legal and appropriate are now invalid, while other values that were formerly illegal are now considered acceptable. In some cases, this will be exactly the behaviour that the business rule change was intended to achieve. However, we can have a negative data-level impact when data that represents perfectly legal real world entities or events is ruled invalid, or when invalid real world entities or events are allowed by the system.

As with derivation rules, we can correct these unwanted effects by propagating the desired change to the affected data items. For example, if a proposal was made to impose the constraint rule just described (regarding eligibility for membership of the loyalty scheme) on an organisation, we might discover that some valued customers were now excluded from the loyalty scheme. Further investigation might reveal that the payment defaults in question were very old and ought to be deleted. Alternatively, a new business rule could be added requiring payment defaults over ten years old to be deleted. Once applied, this change would also remove the negative effect.

Constraint business rules can also participate in chains which lead to unwanted data impacts. Most obviously, such rules can operate over data that is computed as a result of the application of one or more derivation rules. However, they can also have more subtle effects. For example, if a derivation business rule operates on data that is constrained by other rules, then it should really only perform its calculations over the legal subset of that data. Thus, a constraint business rule can change the scope over which a derivation rule operates, and therefore can cause a ripple-effect from the middle of a rule chain as well as at the very end.

3.3. Event Business Rules

This final kind of rule places restrictions on the actions that the organisation can initiate. For example, an event business rule would be used to state the requirement that all refund requests of more than £1000 must be sent to a departmental manager for authorisation before they can be processed.

Event business rules do not themselves impact on data directly, but they can cause data anomalies if the processing of the event in question involves the execution of programs which make updates to the data. Although these are an important class of impacts for ensuring system correctness, they require a mixture of rule impact analysis and code impact analysis that is beyond the scope of this paper. Furthermore, this class of rule is far less common than the previous two. This is partly because, in practice, many event business rules are implemented as constraint business rules. Real world events (especially face-to-face interactions between people) are difficult for computer systems to control by themselves, but easy for computer systems to record and track. Therefore, an event rule of the kind given would more typically be implemented as a constraint business rule requiring a record of authorisation to exist before refund requests can be processed. We therefore do not consider them further in this paper.

4. Impact Analysis through Hypothetical Data Exploration

At present, data-level impacts of the kind described in the previous section are detected (if at all) thanks to the foresight of knowledgeable team members, who know enough about the data and the target system to be able to predict and test for these negative effects. However, just as in the case of more traditional impact analysis, much of the work of checking for impacts is highly repetitive and is well suited to automation. We cannot remove the human from the loop altogether, but we can provide tools which can help to increase the scope and thoroughness of the search and help the human to focus his or her attention on the areas where problems are most likely to arise.

What kind of tool can assist in the detection of business rule impacts on data? The ideal impact analysis tool would be able to deduce the impacts from only a description of the proposed change and the system model. In this case, however, such an approach is unlikely to be feasible, because of the scale of the data sets over which such reasoning must take place. It is also the case that it is not reasonable to expect the user to formulate precisely the characteristics that indicate an anomalous data impact in advance. If this could be done, there would be no need for an impact analysis tool. Instead, what is required is a safe environment in which the user can apply the proposed business rule change, and then explore its effects relative to the live data — a sandbox, in other words, where the user can examine the results of standard queries in the light of the change and diagnose possible anomalies.

The easiest way to achieve this would be to make a copy of the live data, which could be updated and amended as much as necessary to determine the full impact of the change. But where data sets are large, this is too expensive - especially if several competing rule changes have been proposed and the results of the impact analysis are needed to help choose between them. In such a case, several fresh copies would be required - one for each change scenario.

An alternative approach is given by a technique called *hypothetical relations* (HR, for short) [15, 7]. Although not designed for the purpose of impact analysis, this technique meets some of our requirements. The basic idea is as follows. A hypothetical relation is a view onto a real stored relation that can be queried as if it were the original relation. When records are hypothetically added to a relation R, they are diverted into a different relation that has the same schema as R, and is called R_{add}. Similarly, if a request is made to hypothetically delete a tuple from R, it is also diverted and becomes a request to add the same tuple to the special relation R_{del}. R itself is left entirely unchanged. Now, when the user wishes to retrieve data from the hypothetical version of R, what they actually see is the result of the expression:

$$(R/R_{del}) \cup R_{add}$$

By this means, the original relation appears to have been updated as requested, but to all other users it is unchanged.

This existing technique, therefore, gives us a way for users to experiment with the different effects of updating data, without risking damage to the live database that supports it. Now we must add to this the ability to add and delete business rules, and to explore the resulting effects on data. We will now explain how this can be achieved, focussing on the particular case of constraint business rule (CBR) changes, since space is necessarily limited.

The three types of change to CBRs can be collapsed to two, since a rule update is conceptually equivalent to a deletion of the original rule followed by addition of its new form. Therefore, we need to consider only the following two cases.

4.1. Insertion of CBRs

If a new rule is added to the target system, then certain of the currently legal data values will become illegal. In our system, we adopt a common means of expressing constraints over data for the new CBRs: namely, standard queries with added cardinality constraints. For example, the following statement expresses the rule that no customers may be overdrawn by more than £100:

```
CONSTRAIN NO c IN
   SELECT custNo FROM customer
   WHERE balance < -100;
```

This can be converted automatically into a query that returns all customers who are currently violating this rule, simply by removing the cardinality constraint. We now create a hypothetical database from our original customer database, and hypothetically delete the tuples that violate the proposed new rule. This will allow the user to examine the remaining legal values, and to issue queries to check that only the expected number of customers have been made invalid by the evolution. However, it will not help the user to understand the causes of any anomalies they should find — especially when multiple business rule changes have been proposed within the current scenario. In that case, we need to know which rules were responsible for the removal of which values.

In order to support this, we augment the differential tables used to model the hypothetical changes with an additional table, R_{viol}, with the following schema:

$$R_{viol}(RowId,\ CBRId)$$

This allows us to record, for each row in the R_{del} table, which constraint rule (if any) contributed to its removal. Since some rows may be ruled illegal as the result of more than one hypothetically added business rule, it is necessary to allow multiple rule IDs to be recorded for each row ID (i.e. the *RowID* column is not a key).

In order to populate this table for a relation `R` in hypothetical scenario `HS`, we use the following algorithm. `CBR` is the set of constraint business rules that have been hypothetically added to `HS`.

```
Initialise array V, with |CBR| elements
For each rule C in CBR do
   Extract violation query Q from C
   Evaluate Q against HS, and store
            resulting row id set in V[C]
End for
For each rule C in CBR do
   For each row id I in V[C] do
      If I not in Rdel then
         Hypothetically delete row I
                  from R
      End if
      Add row <I, C> to Rviol
   End for
End for
```

The key point to note here is that we determine the violating sets *before* we remove them hypothetically from the state. This allows us to collect all the rules that render a specific row illegal. If we had deleted the illegal tuples as soon as we identified them, they would not have been visible to later CBR queries, and therefore the set of rule culprits for the violations could be incomplete.

With this information, the user can choose to hide violations (by querying directly over the hypothetical relations) or to view them (highlighted to show their illegality). This is achieved by adding a condition to the normal hypothetical query mechanism, so that attempts to retrieve rows from a table R are redirected through the following expression:

$$(R/VRQ) \cup R_{add}$$

where *VRQ* is the query:

```
SELECT * FROM Rdel
WHERE RowId NOT IN
   (SELECT RowId FROM Rviol)
```

4.2. Deletion of CBRs

Deletion of a CBR has the effect of making formerly illegal data values now legal. From the user's point of view, the ideal behaviour would be for query answers that result from such newly legal rows to be flagged in some manner, similar to that described for violations above. However, when a CBR is deleted from a first level hypothetical scenario (that is, one that is defined directly on top of a real database, such as H1 and H2 in Figure 1) there only way that there will be any such rows will be if historical data is present in the database that violates the current set of business rules. In fact, this is relatively common in real systems, since many new rules are assumed to apply only to new business and not to past transactions [4]. It can also occur with second degree and higher hypothetical scenarios, when a rule which was added in an earlier scenario is deleted.

The procedure for making such rows visible is very simple, and is based around a slight modification of the *VRQ* query given above:

```
SELECT * FROM Rdel, Rviol
WHERE Rdel.RowId = Rviol.RowId AND
      Rviol.CBRId NOT IN ActiveCBRs
```

where `ActiveCBRs` is a list of the business rules that are not deleted in the current scenario.

4.3. The Impact Analysis Tool

Taken together, the functionality just described allows the user to construct and compare multiple hypothetical scenarios over the same data set, simply by diverting the updates for each scenario into a different pair of *add* and *del* relations, and applying the updated rules. This is illustrated in Figure 1, a screenshot taken from the BR impact analysis system we are implementing. The middle panel shows four hypothetical scenarios that the user has constructed from the original `shop` database. In hypothetical scenario H1, for example, the user is experimenting with new rules on delivery charges. She has created a further scenario based on H1, scenario H3, which includes all the changes applied to H1 and some additional rules regarding the minimum order value. The user can issue queries to each of the scenarios, so that she can compare their values and this assess their relative impacts on the live data.

5. Conclusions

We have presented the case for a new kind of software change impact that, to the best of our knowledge, has not yet been studied within the software engineering community. With the complexity of modern software systems, developers need tools that can assist them in making changes to those systems in a complete and consistent way. We have shown how, for some changes in functionality (namely, to business rules), changes to the data managed by a system may be required in order to apply the change consistently and completely. Tools are needed to identify this kind of impact, just as they are for the more conventional impacts.

One potential disadvantage of our approach is that it requires the current set of business rules enforced by a system to be entered into the tool before the full set of impacts can be found. Further experimentation is needed in order to determine whether the benefits of the approach outweigh this cost. We are currently also extending our tool to deal with impacts resulting from changes to derivation business rules, as well as exploring alternative evaluation techniques based on techniques from hypothetical reasoning.

Acknowledgements

The $\mathrm{WiA_{BR}}$ project is supported by a grant from the UK Engineering and Physical Sciences Research Council.

References

[1] S. Bohner and R. Arnold, editors. *Software Change Impact Analysis*. IEEE Computer Society Press, 1996.

[2] L. Briand, Y. Labiche, and L. O'Sullivan. Impact Analysis and Change Management of UML Models. In *Proc. of 12th Int. Conf. on Software Maintenance*, pages 256–265. IEEE Computer Society Press, 2003.

[3] F. de Boer *et al.* Change Impact Analysis of Enterprise Architectures. In *Proc. of IEEE Int. Conf. on Information Reuse and Integration*, pages 177–181, Las Vegas, USA, Aug. 2005. IEEE Computer Society Press.

[4] A. Earls, S. Embury, and N. Turner. A Method for the Manual Extraction of Business Rules from Legacy

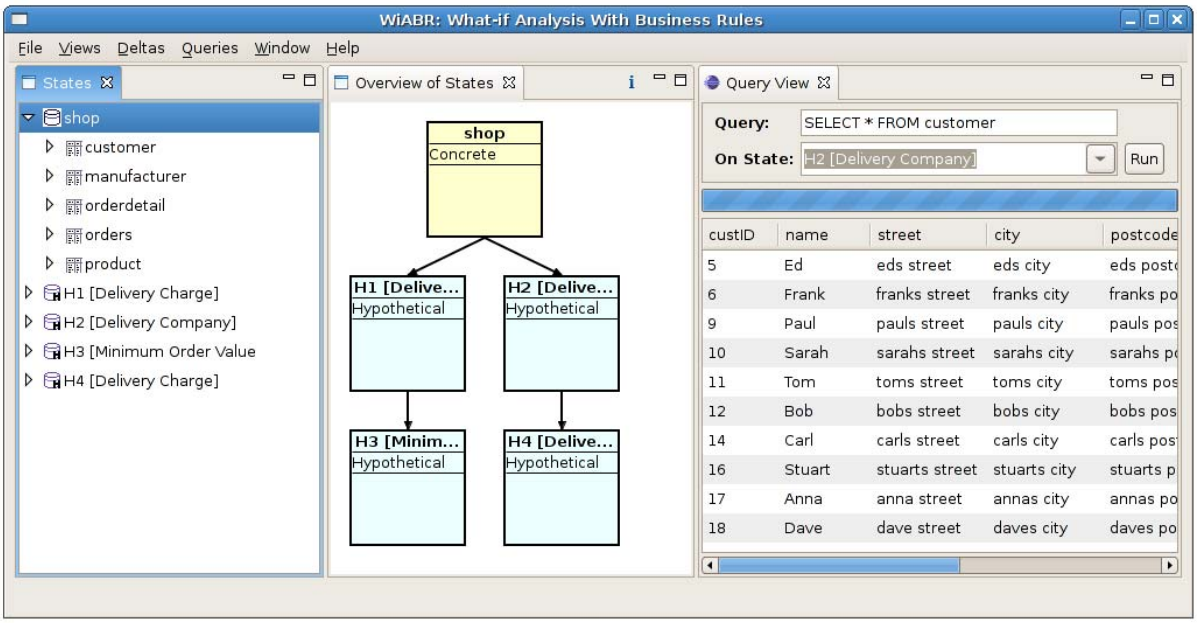

Figure 1. Example Screenshot from the BR Impact Analysis System

Source Code. *BT Technology Journal*, 20(4):127–145, Oct. 2002.

[5] S. Embury and J. Shao. Analysing the Impact of Adding Integrity Constraints to Information Systems. In *Proc. of 15th Int. Conf. on Advanced Information Systems Engineering*, LNCS vol. 2681, pages 175–192, Klagenfurt, Austria, June 2003. Springer.

[6] D. Hay and K. Healy. Defining Business Rules — What are they Really? Technical report, The Business Rules Group, Revision 1.3, July 2000. `http://www.essentialstrategies.com/publications/businessrules/`.

[7] U. R. Kulkarni and R. G. Ramirez. Independently Updated Views. *IEEE Transactions on Knowledge and Data Engineering*, 9(5):798–812, 1997.

[8] W. Lam and V. Shankararaman. Requirements Change: a Dissection of Management Issues. In *Proc. of the 25th EUROMICRO Conf.*, pages 2244–2251, Milan, Italy, Sept. 1999. IEEE Computer Society Press.

[9] J. Law and G. Rothermel. Whole Program Path-Based Dynamic Impact Analysis. In *Proc. of 25th Int. Conf. on Software Engineering*, pages 308–318. IEEE Computer Society Press, 2003.

[10] L. Li and A. Offutt. Algorithmic Analysis of the Impact of Changes to Object-Oriented Software. In *Proc. of 4th Int. Conf. on Software Maintenance*, pages 171–184, Monterey, CA, USA, Nov. 1996. IEEE Computer Society Press.

[11] L. Lin, S. Embury, and B. Warboys. Facilitating the Implementation and Evolution of Business Rules. In *Proc. of Int. Conf. on Software Maintenance*, pages 609–612. IEEE Computer Society Press, Sept. 2005.

[12] A. Podgurski and L. Clarke. A Formal Model of Program Dependences and its Implications for Software Testing. *IEEE Transactions on Software Engineering*, 16(9):965–979, Sept. 1990.

[13] J.-P. Queille, J.-F. Voidrot, N. Wilde, and M. Munro. The Impact Analysis Task in Software Maintenance: a Model and a Case Study. In *Proc. of the Int. Conf. on Software Maintenance*, pages 234–242, Victoria, BC, Canada, Sept. 1994. IEEE Computer Society Press.

[14] J. Shao and C. Pound. Reverse Engineering Business Rules from Legacy Systems. *BT Technology Journal*, 17(4):179–186, 1999.

[15] M. Stonebraker. Hypothetical Data Bases as Views. In *Proc. of the ACM-SIGMOD Conf. on Management of Data*, Ann Arbor, Mich, US, 1981.

[16] P. Tonella. Using a Concept Lattice of Decomposition Slices for Program Understanding and Impact Analysis. *IEEE Transactions on Software Engineering*, pages 495–509, 2003.