Business rule evolution and measures of business rule evolution

Liwen Lin, Suzanne Embury and Brian Warboys
Department of Computer Science
The University of Manchester
Oxford Road, Manchester, United Kingdom
{l.lin, suzanne, brian}@cs.man.ac.uk

Abstract

There is an urgent industrial need to enforce the changes of business rules (BRs) to software systems quickly, reliably and economically. Unfortunately, evolving BRs in most existing software systems is both time-consuming and error-prone. In order to manage, control and improve BR evolution, it is necessary that the Software Evolution community comes to an understanding of the ways in which BRs are implemented and how BR evolution can be facilitated or hampered by the design of software systems. We suggest that new software metrics are needed to allow us to measure the characteristics of BR evolution and to help us to explore possible improvements in a systematic way. A suitable set of BR-related metrics will help us to discover the root causes of the difficulties inherent in BR evolution, evaluate the success of proposed approaches to BR evolution and improve the BR evolution process as a whole.

1. Introduction

A Business Rule (BR) is *"a statement that defines or constrains certain aspects of a business"* [13]. For example, a car rental company may have such a BR – each rental car must be serviced every 10,000 kilometres. BRs are major factors that affect the competitive of a business and they are therefore subject to frequent and unpredictable changes; these changes in turn lead to changes in the software systems that enforce the BRs.

However when software systems are large and complex, making changes to the BRs they implement can be too expensive and risky to be worth the attendant benefits. Thus the owners of such systems are discouraged from making changes to their BRs. In order to make evolution of BRs easier, we need to make substantial progress in understanding the characteristics of BR implementations and understanding how BR evolution is facilitated or hampered by the design features of the systems in which they are implemented.

One way to approach these issues is to use software metrics to precisely characterise the process of evolution for BRs. In particular, we could try to derive a relationship between different approaches to BR implementation and their attendant difficulties in evolution. Thus, we can deepen our understanding of how different software architectures can support BR evolution, and of what it is that makes BR evolution so difficult. This would then allow us to propose more effective solutions than those which currently exist. And, in addition to these benefits, BR-specific software metrics could be used to quantify the strengths and weaknesses of any proposed approaches, and thus allow us to make informed choices between them.

While many generic software metrics exist (*e. g.* complexity metrics, coupling/cohesion metrics [12]), there are few that are tailored specifically at the measurement of BRs. It seems likely however that some of the existing metrics can be adapted to our purposes. For example, it might be possible that traditional measures of software complexity can be used to measure some aspects of the complexity of BR implementations. But other aspects of BRs, such as the complexity of BR representation and effort required to evolve a BR, will not be clearly defined by existing metrics, and we therefore suggest that there is a need to define a new set of metrics, preferably agreed upon by a significant part of the business rule community, in order to allow us to capture the data on the characteristics of BR evolution that is needed in order to support further progress in this area.

In the remainder of this paper, we will describe the challenge of BR evolution and emphasise the importance and potential benefits of a systematic study into BR evolution.

2. The challenge of BR evolution

The challenge for BR evolution is to be able to support unforeseen changes to BRs quickly, economically and reliably. Unfortunately, in reality, the process of evolving BRs

Proceedings of the Sixth International Workshop on Principles of Software Evolution (IWPSE'03)
0-7695-1903-2/02 $17.00 © 2002 IEEE

is both time-consuming and error-prone [11]. For example, BRs are typically implemented in both application modules and in database structures, and are scattered throughout the source code of many application modules. The form of the BR is lost during the translation to conventional object-oriented or procedural software. What was once a concise and declarative statement of business behaviour is converted into a set of programming instructions, which are spread widely throughout the application and which have no easily discernable connection to one another. Indeed, it is not even easy to decide which lines of code are part of a BR implementation, once the translation has been made, and the documentation which would preserve this information rarely exists.

Consequently, such BR implementations are difficult to modify and keep consistent. Moreover, when implementation of some BRs are scattered across several program units, the process of evolving these rules requires not only changes to the program units that implement them but it may also require changes to other program units that interface with these program units. When the system is badly designed or long-lived, most program units may be related to each other in some way; the consequence is that it becomes impossible to evolve BRs without restructuring large parts of the system. The expense of such a change can be prohibitive.

2.1 The contemporary approaches

One widespread approach to improving the evolvability of BRs is to "externalise" their implementation from the applications that depend upon them. Over the past 30 years, several technologies and tools have been proposed which aim to do just this. These include integrity constraints in database systems [21], triggers in active database systems [17], constraints in object-oriented systems (*e.g.* coordination contracts [2, 3] and IBM's Accessible Business Rules (ABR) [14]), expert systems (*e.g.* OPS5 [8] and CLIPS [19]), and knowledge-based inference engines (*e.g.* ILog JRules [16], CommonRules [15], and Versata Logic Server [22]).

The common feature of all these technologies is that BRs are managed and enforced through some centralised rule management facility. BRs are given to the system as self-contained units (whether as declarative integrity constraints or procedural descriptions of condition-action rules) and the rule manager takes over the task of checking the actions of the system against the current set of BRs.

With such an approach, it is relatively easy to locate the rules that need to be changed and to discover what rules the system is currently enforcing. One needs only to examine the central rule repository. The rule management system will also offer facilities for deleting existing rules and adding news ones. These technologies can therefore claim to make BRs much easier to implement and evolve than if BRs are implemented directly in the source code of application programs.

2.2 Limitations of the contemporary approaches

these above mentioned technologies, however, can provide only a partial solution to the BR evolution problem, and they have some new disadvantages of their own. Most of these technologies are only suitable for use with restricted types of BRs. For example, rule engines implement their rules as a set of mid-tier services in software applications, using a special procedural rule language that requires the programmer to embed calls to the rule manager at appropriate points in the processing. This approach is suited only to certain forms of inference and data derivation rule. Worse still, the need to embed calls to the rule manager means that rules are not completely externalised, and much code may need to be examined and modified when BRs change. It also suffers from the disadvantage that such engines cannot easily be grafted onto an existing software system (especially a database-oriented application). To take another example, triggers in active database systems can be used to implement BRs that apply to the elements and values of the database. Due to the limitations of existing trigger languages, it is not possible to implement all BRs in this manner. In addition, even when complex BRs can be implemented by the trigger language, the result is often a large set of highly complicated triggers, which can be challenging to understand and evolve in their own right [10].

In addition to the above limitations, this group of technologies suffers from performance problems when rules are complex or when the rule set as a whole is large [10, 20]. These problems are so severe that they can be seen as the principal reason why companies have not adopted technology such as triggers and rule engines as widely as might have been hoped. The most important (and often the most volatile) BRs are concerned with key aspects of how the company which enforces them does its business. Poor performance cannot be tolerated in the parts of the system where speed of transaction processing translates into income for the company.

Finally, while these technologies can help to make BR evolution less error prone, they do not solve the problem completely - especially when rule sets are large. For instance, when introducing a new rule, developers still have to detect and locate possible conflicts between the new rule and the existing rules, something that can be a difficult task given the size of typical rule sets and the current paucity of software support. At the moment, the only support for conflict resolution that is provided by rule management technology is based on assigning priorities to rules. For exam-

ple, in active databases conflict resolution is limited to hard-coded rule ordering or numeric priorities [1], while CLIPS tag rules with specific priorities such as low, medium or high [19]. Ironically, such priority mechanisms typically mean that rules sets are even harder to evolve, as the complex interactions between rules at different priority levels are even more difficult to predict correctly.

While externalisation of rules and centralisation of rule management is an interesting and worthwhile avenue for exploration, there is clearly still scope for improvement. However, the most significant barrier in the way of such improvement is our lack of understanding of the factors that make BR evolution so difficult. Previous work has been based on intuitions and guesses about the problem, rather than on any systematic study of the real causes of errors and difficulties inherent in the process. Nor do we really understand how the architecture and design of a software system can facilitate or hamper the evolution of BRs. If we can make substantial progress with these issues, then research effort expended on developing tools and methods to manage and control BR evolution is more likely to yield significant benefits.

3 Measures of BR evolution

In the previous section, we have drawn attention to the following questions: how can we discover the root problems underlying BR evolution? How can we discover the relationships between the characteristics of BR implementations and the difficulties inherent in the evolution of BRs? Can we elicit an understanding of how software design structures constrain or enable BR changes?

One way to approach these questions is to use software metrics to measure the detailed aspects of the BR evolution process. For example, we might measure the complexity of the rule to be evolved, the fragmentation or dispersal of its code during its implementation or the degree of structure imposed on its implementation. Or, we might measure aspects of the process itself, such as the effort required to locate and evolve a business rule, or the number of errors introduced by the modification.

3.1 Benefits of measuring BR evolution

Measuring the characteristics of BR evolution through empirical studies will enable us to formulate and test new hypotheses about the root causes of BR evolution problems. For example, we are currently engaged in a study of the degree and effects of fragmentation of BRs during implementation. By fragmentation of a rule, we mean the separation of the initial high-level rule statement into the fragments that will be embedded into the various application programs and code modules, and then the further separation of these fragments into individual lines of code scattered throughout them. Clearly, different approaches to the implementation of a BR will result in different degrees of fragmentation for that rule. We hypothesise that the greater degree of fragmentation of BR implementation, the more effort is required to evolve the BR.

Using the GQM/MEDEA approach [5, 6, 7], we have derived and selected several software metrics to measure the fragmentation of a BR implementation and the effort required to evolve it. We are now undertaking an empirical study to collect data relating to these two aspects of BR evolution, and are looking for evidence of a relationship between them. Discovering the root problems of BR evolution in this way will enable us to propose new tools that can help programmers to control, or at least mitigate the bad effects of, these problems. Other than that, through quantitative measurement of those characteristics of BR implementation which cause difficulties, we can evaluate the strengths and weaknesses of various BR implementation technologies and compare them on a more systematic basis. As a result, we can choose the best technology to implement BRs in a way that meets our evolution needs.

For example, one factor of software systems that inhibits BR evolution is the cost and difficulty of restructuring existing software systems to adapt to dramatic changes in BRs, such as when a new set of government regulations is issued or the company embarks on a completely new way of doing business (e.g. through e-commerce). The question that arises is whether we can adapt existing proposals for more flexible software architectures for software evolution, such as ArchWare [4], FLEXX [18], and DESEL [9] projects to improve support for BR evolution. These architectures aim to support the design and development of evolvable software systems, but we do not know as yet how far they can solve the specific problems inherent in the evolution of BRs. Without a more detailed understanding of these problems, we cannot determine how far such frameworks are helpful in promoting more evolvable BR implementation approaches.

3.2 Difficulties of using metrics in BR evolution

In considering BR evolution, we must remember that it is not sufficient to take a short-term view of improving the evolution capabilities of our systems. What is highly evolvable for the current set of BRs may be extremely cumbersome to change in the light of the BRs being enforced in future versions of the system. What is required is a continuous approach to the improvement of the software system's ability to support BR evolution. However, such an approach would rely heavily on the collection and monitoring of software metrics relating to the implementation and evolution of BRs. Such a facility would exploit software measure-

ments to identify where evolution is required, which technology is most appropriate for use at that stage, select alternative BR structures for promoting continuous BR evolution and even evolve the evolution process itself to improve the reliability, efficiency and accuracy of BR evolution.

For all these reasons, therefore, it would appear that an agreed set of software metrics describing the various aspects of BR implementation and evolution would be of great value to researchers investigating the BR evolution problem. Unfortunately, however, it is not at all easy to measure the characteristics of BR evolution. The difficulties that attend metrics research in general would apply equally to any BR-specific efforts. For example, the measures of BR evolution would involve quantifying some attributes that depend on human behaviour and capabilities which cannot be controlled and measured easily. A classic example of such a metric would be the effort required to evolve a BR. Furthermore, many other aspects of BR evolution are very difficult to define precisely. For example, it is in practice non-trivial to define a notion of the complexity of a BR in a succinct and measurable way.

3.3 The next step

Although a large number of software measures have been proposed in the literature, their applicability to the specific problem of BR evolution is still in doubt. For example, can we use traditional software complexity metrics to measure the complexity of a BR implementation? How far do such metrics reflect our intuition about what makes a BR implementation complex or not? In addition, many software metrics have never been subjected to empirical validation, thus their credibility is undermined [7, 12]. Consequently, further study on metrics for BRs will be necessary, including the following elements:
(1)Provide a clear rationale of the definition of characteristics of BR evolution and the measures for these characteristics.
(2) Use of software metrics in a language independent way is essential so that measurement results using different languages are comparable.
(3) Realistic case studies are essential to investigate and validate applicable measures that capture attributes of BR evolution in a quantitative way.

4 Conclusion

The inability of current software systems to adapt to (largely unpredictable) changes to BRs presents an unsolved challenge for researchers in software evolution. Although substantial research has been carried out in this context, the results of this research provide only a partial solution to the problem, along with new sets of disadvantages to be overcome. In this position paper, we have proposed that new software metrics are needed to help us characterise the real problems inherent in BR evolution, in order to deepen our understanding of the way in which BR evolution is helped or hindered by software systems. With this increased understanding, we can propose new approaches to supporting BR evolution and can evaluate and compare the relative strengths and weaknesses of existing approaches in a more systematic way.

References

[1] A. S. Abrahams. *Developing and executing electric commerce applications with occurrences*. PhD Thesis, the University of Cambridge, Cambridge, 2002.

[2] L. Andrade, J. L. Fiadeiro, and M. Wermelinger. Enforcing business policies through automated reconfiguration. In *proceedings of automated software engineering (ASE 2001), M. Feather, M. Goedicke (eds), San Diego, USA*, pages 426–429, November 2001.

[3] L. F. Andrade and J. L. Fiadeiro. Coordination: the evolutionary dimension. In *Proceedings of technology of object-oriented languages and systems, TOOLS Europe 2001, W. Pree (ed), Zurich, Switzerland*, pages 136–147, March 2001.

[4] ArchWare. http://www.arch-ware.org/project.htm, 2002.

[5] V. R. Basili, C. Caldiera, and H. D. Rombach. Goal question metric paradigm. In *Encyclopaedia of software engineering, vol 1, J. J. Marciniak (Ed.)*, pages 528–832. John Wiley and Sons, 1994.

[6] V. R. Basili and D. M. Weiss. A methodology for collecting valid software engineering data. *IEEE transaction on software engineering*, 10:728–738, November 1984.

[7] L. C. Briand, S. Morasca, and V. R. Basili. An operational process for goal-driven definition of measures. *IEEE transactions on software engineering*, 28(12):1106–1125, December 2002.

[8] I. Brownstone, R. Farrell, E. Kant, and N. Martin. *Programming expert systems in OPS5: an introduction to rule-based programming*. Addison-Wesley, 1985.

[9] DESEL. http://www.personal.rdg.ac.uk/ sis99scc/desel/, 2002.

[10] A. K. Dittrich and E. Simon. Active database systems: Expectations, commercial experience, and beyond. In *Active rules in database systems, N. W. Paton (ed)*, pages 367–404. Springer, 1999.

[11] S. M. Embury and J. Shao. Analysing the impact of adding new integrity constraints to information systems. In *proceedings of the 15th conference on advanced information systems engineering, CAiSE'03, Vienna, Austria*, June 2003.

[12] N. E. Fenton and S. L. Pfleeger. *Software metrics: a rigorous and practical approach*. PWS publishing company, New York, 1997.

[13] D. Hay and K. Healy. Defining business rules - what are they really, GUIDE Business Rule Report. http://www.businessrulesgroup.org/, 2000.

[14] IBM. Accessible Business Rules. http://www.research.ibm.com/AEM/abr.html.

[15] IBM. Overview of IBM CommonRules. http://www.research.ibm.com/rules/commonrules-overview.html.
[16] ILog. ILog JRule. http://www.ilog.com/.
[17] N. W. Paton and O. Diaz. Introduction. In *Active rules in database systems, N. W. Paton (ed)*, pages 1–28. Springer, 1999.
[18] S. Rank, K. Bennett, and S. Glover. Flexx: designing software for change through evolvable architectures. In *Systems engineering for business process change, P. Henderson(ed)*. Springer, 2000.
[19] G. Riley. CLIPS: a tool for building expert systems. http://www.ghg.net/clips/CLIPS.html, 2001.
[20] I. Rouvellou, L. Degenaro, K. Rasmus, D. Ehnesbuske, and B. McKee. Extending business objects with business rules. In *Proceedings of the 33rd international conference on technology of object-oriented languages and systems (TOOLS Europe 2000), Mont Saint-Michel/ St-Malo, France*, pages 238–249. IEEE Computer Society Press, June 2000.
[21] J. D. Ullman and J. Widom. *A first course in database systems*. Prentice-Hall, 1997.
[22] Versata. Versata Logic Suite. http://www.versata.com.