Feature Level Metrics Based on Size and Similarity in Software Product Line Adoption

András Kicsi, Viktor Csuvik

Abstract: Introducing software product lines is a natural way to cope with a large number of software variants and hard maintenance. This task can become more complicated with a fourth generation language, namely Magic in our case. Feature extraction is an important task of product line adoption, and the extracted features can amount to large proportions of the code and can be hard to contemplate, thus appropriate methods become necessary to ease the handling of the information gained. In this work we present some feature level metrics aiming to highlight valuable information on both the results attained through extraction and the features themselves which can be used in furthering the process of product line adoption. We present some metrics based on size and pairwise similarity of the features of four different variants of the same system. The knowledge of these metrics, properly measured and used can be vital in aiding product line adoption.

Keywords: spl, 4gl, feature, magic, metrics

Introduction

Software product lines (SPL) are a common way to cope with the reuse of large software systems. In case of an existing system product line adoption can significantly ease maintenance. SPL adoption is an extensive research branch of software development but literature mostly deals with traditional environments with less emphasis on fourth generation languages (4GL).

Our subject system is a pharmaceutical wholesaler logistics system which was started more than 30 years ago, and is used with great popularity ever since. This popularity has led to the introduction of more than 20 variants of the system which have varying development cycles and isolated maintenance. Our industrial partner is the developer of market leading solutions in the region, which are implemented in the Magic XPA 4GL language [7].

Magic as a language presents some unique challenges. As a fourth generation language development relies heavily on the user interface and produces no source code in the traditional sense. This presents an obstacle for most mainstream static analysis techniques. Magic as a language has also developed a lot in the last 30 years, it had gone through several major version changes introducing significant changes in the structure of the language. There are currently 19 active variants of our subject system written in four different main versions of Magic. A software written in Magic can consist of one or more projects. Projects have their own programs, which are the closest things to the methods of a traditional programming language. Programs can be called and they can call each other too. These programs build up the essence of the functionality provided by the project. We consider these as the building blocks of the features, one program can take part in more features and one feature is usually provided by the work of several programs. Programs can use several data objects through which they access the data stored. Since Magic is a data intensive language, these play a significant part too.

In our previous work [4, 5] we defined our process for feature extraction in the same environment. We presented our work with an information retrieval method based on Latent Semantic Indexing which relies more on the textual similarities located in the features and programs and a technique based on the program calls inside a system, which was based on static analysis and the building of a call graph. We have been provided with a multi-level feature list by domain experts which describes the features present inside the variants of the system. For the sake of clarity we only present the results attained with the highest level of features, of which we distinguish 10 different features. The main contribution of this work is the proposal of several potentially useful metrics on the extracted features of 4GL product line adoption which previously did not exist by our knowledge.

Proposed Metrics

In the current section we define our proposed metrics and display some results attained through their computation on four different variants of the system under study. We mention these variants simply as

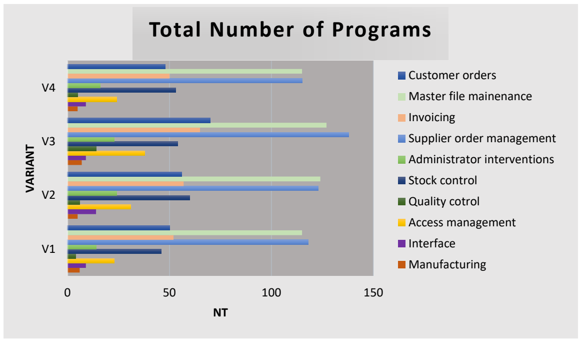

Figure 1: Number of programs by variant

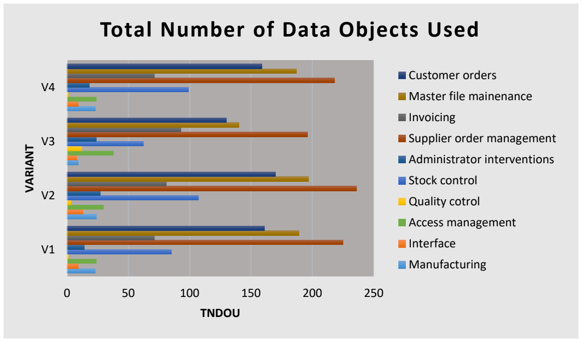

Figure 2: Number of data objects by variant

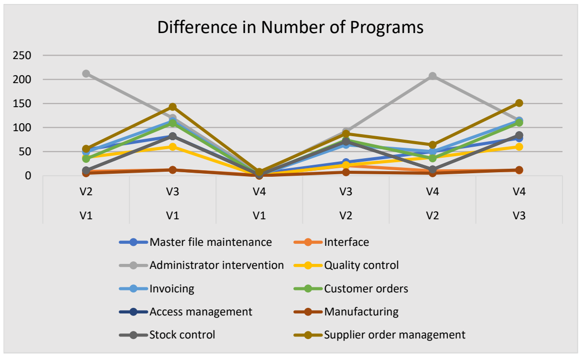

Figure 3: Difference in number of programs at each feature by pairs of variants

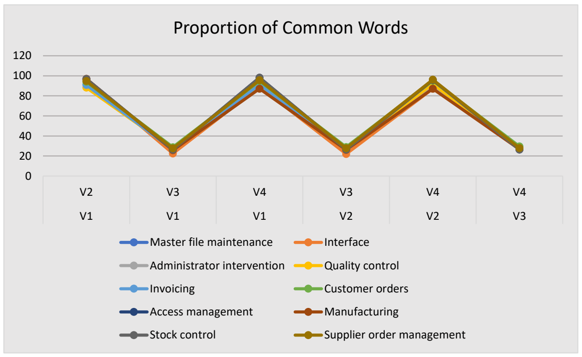

Figure 4: Proportion of common words at each feature by pairs of variants

V1, V2, V3 and V4. According to our previous knowledge V3 differs from these variants highly, while the others share a great number of programs and support a very similar set of functions but still vary somewhat in the specifics. The largest of the variants contains 4251 programs and uses 1065 data objects. Since we have experimented with different feature extraction outputs [5], we have chosen one of these for display, specifically the output achieved with our information retrieval based method which relies on textual similarity attained through Latent Semantic Indexing. The metrics proposed in this section were all measured but due to space limitations we only display the results of just a few of them.

Since at the feature extraction phase we worked on assigning programs to specific features the most straightforward metrics to think of here is the number of programs assigned to each feature. Results of this metric are displayed in Figure 1. On the other hand the number of features assigned to each program can be measured too. These metrics provide basic understanding of the size of features and the importance of the specific programs in their workings. These can be crucially important since we can get a picture of the complications resulting from changes done in a single program. Since a Magic application can have more than one projects, the number of these are also important on the feature level too. In our case the applications always had one single project.

Programs also access data objects to gain the data needed to perform their tasks. Reliance on data objects is another information that can be important for working with these features as sets of their programs. One such metric is the total number of data objects used, for which results can bee seen in Figure 2. In addition to this, since data objects are located in data sources and consist of columns and can even have indices, the number of these are measured too. In our case there are three different data sources, of which most features tend to use all three.

Feature similarity can be measured for each pair of features. Computing these metrics to features of the same variant can also have uses, for example if we are contemplating merging some features on the lower level, but the main importance lays in computing difference between the same feature of two different variants. Similarity can be interpreted in many ways in this case. Since we can handle features as sets of programs, the most straightforward approach is the number of common programs or the absolute difference in the number of programs which is displayed in Figure 3. The average number of programs or the proportion of size of the smaller feature relative to the larger one can also provide a quick and easy glance at similarity. These metrics are based on feature size and while they can indicate similarity we introduce some textual methods too. One of these is simply the number of unique words contained in their programs which is also based on size but works at a more conceptual level. We can also calculate the proportion of common words which is displayed in Figure 4. It is easy to see from this figure that V3 differs from the rest of the variants greatly which corresponds to our previous knowledge. Textual matching can also be applied as only partial matching when one term appears inside another. We computed both the number and proportion of these sub matches. For these textual metrics textual preprocessing was applied to the text of the programs.

Related Work

Software product line adoption has been an intensively studied subject during recent years [1, 10, 6]. The identification of specific features inside the software is usually one of the major parts of this process [3]. Reverse engineering of 4GL languages, particularly Magic is not extensive, but certain efforts at optimization are visible topics in the Magic community [2].

There already exist several quality measures specifically developed to Magic environments [9, 8]. Our current work deals with the extension of these efforts to the feature aspects of the applications, which can be a valuable asset in product line adoption.

Conclusion

In our current work we present a number of metrics used in attaining more thorough understanding of features extracted from different variants of a Magic 4GL software. In particular, we presented some fundamental metrics based on the size of extracted features and some similarity metrics for the pairwise comparison of features as well. We have computed these metrics on 4 different variants of the application with 10 top level features identified.

These metrics can be computed for each feature extraction output, particularly useful for reaching deeper understanding of the inner workings of the system. With size metrics we can get a good picture of the magnitude of work ahead of us in case of each feature as well as a hint to their complexity. With similarity metrics we can actually see some of the differences between variants and where those differences lay, thus pointing out the specific locations where differences are to be breached, focused on actual functionality.

Acknowledgements

This project was supported by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002).

References

[1] Wesley Klewerton Guez Assunção and Silvia Regina Vergilio. Feature location for software product line migration. In *Proceedings of the 18th International Software Product Line Conference on Companion Volume for Workshops, Demonstrations and Tools - SPLC '14*, pages 52–59, New York, New York, USA, 2014. ACM Press.

[2] John V. Harrison and Wie Ming Lim. Automated Reverse Engineering of Legacy 4GL Information System Applications Using the ITOC Workbench. In *10th International Conference on Advanced Information Systems Engineering*, pages 41–57. Springer-Verlag, 1998.

[3] Evelyn Nicole Haslinger, Roberto E. Lopez-Herrejon, and Alexander Egyed. Reverse Engineering Feature Models from Programs' Feature Sets. In *18th Working Conference on Reverse Engineering*, pages 308–312. IEEE, oct 2011.

[4] András Kicsi, László Vidács, Árpád Beszédes, Ferenc Kocsis, and István Kovács. Information retrieval based feature analysis for product line adoption in 4gl systems. In *Proceedins of the 17th*

International Conference on Computational Science and Its Applications – ICCSA 2017, pages 1–6. IEEE, 2017.

[5] András Kicsi, László Vidács, Viktor Csuvik, Ferenc Horváth, Árpád Beszédes, and Ferenc Kocsis. Supporting product line adoption by combining syntactic and textual feature extraction. In *New Opportunities for Software Reuse : 17th international conference, icsr.* Springer International PU, 2018.

[6] Crescencio Lima, Christina Chavez, and Eduardo Santana de Almeida. Investigating the Recovery of Product Line Architectures: An Approach Proposal. pages 201–207. Springer, Cham, may 2017.

[7] Magic Software Enterprises Ltd. Magic Software Enterprises. http://www.magicsoftware.com, last visited May 2017.

[8] Csaba Nagy, László Vidács, Rudolf Ferenc, Tibor Gyimóthy, Ferenc Kocsis, and István Kovács. MAGISTER: Quality Assurance of Magic Applications for Software Developers and End Users. In *26th IEEE International Conference on Software Maintenance*, pages 1–6. IEEE Computer Society, September 2010.

[9] Csaba Nagy, László Vidács, Rudolf Ferenc, Tibor Gyimóthy, Ferenc Kocsis, and István Kovács. Complexity measures in 4gl environment. In *Computational Science and Its Applications - ICCSA 2011, Lecture Notes in Computer Science*, volume 6786 of *Lecture Notes in Computer Science*, pages 293–309. Springer Berlin / Heidelberg, 2011.

[10] Marco Tulio Valente, Virgilio Borges, and Leonardo Passos. A Semi-Automatic Approach for Extracting Software Product Lines. *IEEE Transactions on Software Engineering*, 38(4):737–754, jul 2012.