Risk and Complexity Assessment on the Context of Language Migration

Santiago Bragagnolo, Abderrahmane Seriai, Stéphane Ducasse, Mustapha Derras

Risk and Complexity Assessment on the Context of Language Migration

Santiago Bragagnolo✉[1,2] santiago.bragagnolo@berger-levrault.com,
Abderrahmane Seriai[1] abderrahmane.seriai@berger-levrault.com,
Stéphane Ducasse[2] stephane.ducasse@inria.fr, and Mustapha Derras[1]
mustapha.derras@berger-levrault.com

[1] Berger-Levrault, Montpellier, France
[2] Université de Lille, CNRS, Inria, Centrale Lille, UMR 9189 – CRIStAL, France

Abstract. Language Migration is a highly risky and complex process. Many authors have provided different ways to tackle down the problem, but it still not completely resolved, even-more it is considered almost impossible on many circumstances. Despite the approaches and solutions available, no work has been done on measuring the risks and complexity of a migration process based on the technological gap. In this article we contribute a first iteration on Language Migration complexity metrics, we apply and interpret metrics on an industrial project. We end the article with a discussion and proposing future works.

Keywords: Migration · Challenges · Risk · Assessment · Metrics.

1 Introduction

With the fast evolution of programming languages and frameworks, companies must evolve their systems. This evolution may imply the full migration of their applications to new technological environments. Our work takes place in collaboration with Berger-Levrault, a major IT company selling information systems developed in Microsoft Access among others. Microsoft Access is ageing and not able to respond to the architectural needs of modern times, threatening the continuity of these information systems and pushing them into the classification of Legacy Systems.

To respond to the process of obsolescence, as explained by [5], we are working on a **software evolution** process of modernization by migration of Microsoft Access applications (source application) to a web architecture (target application). The technological choice of the target web architecture is Angular for the front-end and microservices for the back-end, in alignment with the migration policy of the company.

According to [5] a generic iterative migration responds to the iterative application following process steps: (i) Plan (ii) Understand system (iii) Understand destination (iv) transform knowledge (v) Produce Destination. Our article aims

to help to *plan* by measuring the potential complexity of the *transform knowledge* phase, by measuring the distance in between the origin system and the destination system.

Modernization is a risky and complex case of software evolution that could be near to impossible or just not worthy in certain circumstances. This complex and risky nature is not only due to the quality of the source project (as many other articles already spotted [1, 17]), but also due to the gap in between the origin and destination technologies. Much has been told about the impact of cohesion and coupling to the decomposability of software, and how this decomposability is key for iterative processes of migration ([2, 8, 10, 15, 21, 25]), but to the best of our knowledge there are not works on the measure of the technological gap, despite the conscious acknowledgement on the literature ([22]) of the correlation between the difficulty of a migration and the difference between origin and target technological platforms.

Following we explain the context and highlight the challenges of our migration project (section 2). We introduce our different counters and highlight their relationship with the complexity of the process of migration on different levels in our particular setup (section 3). We extract the proposed metrics on an industrial project, and propose an interpretation of the extracted numbers (section 4). After interpretation, we discuss the flaws of this kind of metrics and what it still to be done for having a functional set of gap metrics useful to predict complexity (section 5) We finish the article with a preview of future works (section 6) and a conclusion (section 7).

2 Challenges

The migration of Microsoft Access applications to a web technology is a difficult task. Indeed, it requires working on several levels of abstraction, namely: the architectural level and the source code level. In this section we present, in a non-exhaustive way, the difficulties inherent in the process of migrating VBA applications to a web technology.

It is important to remark that these challenges are not solved yet, but exposed to give context to the metrics. The metrics should contribute to the measurement of these challenges.

2.1 Software architecture challenges

Moving from a centralized/standalone application to a distributed application Microsoft Access is a development environment used to create database based applications. It comes with a programming language called Visual Basic For Application (VBA) and various libraries. This language and the libraries are used in a programming environment called Microsoft Visual Basic, which also comes with Microsoft Access. Microsoft Access applications are so-called “standalone” applications, i.e. they are developed to be deployed centrally, although they have

the ability to interact with remote data servers. Furthermore, a standalone application can access the resources of its deployment environment (user computer), such as the operating system used, the file system, printers, etc. In contrast, a distributed application rarely access users resources but network resources, assuming the existence of a network and shared resources. [20] is a good example of how radical may be this kind of modification. The difference between this two environments makes many original development assumptions to not be valid any more, requiring to be adapted or completely redeveloped during a migration process. Further, in this article (section 3), we propose the paradigmatic change and the dependencies counters which measure the amount of entities that are related with this aspect.

Moving from a monolithic application to a microservice application A monolithic application is often described as a single-tier system in which the user interface, business logic and data layer are combined into a single application. Microservices is one of the latest trends in software development that has emerged from service-oriented architecture styles. Microservices are expected to be specialized on specific concerns, small, highly cohesive, loosely coupled services, each independently deployable and communicating with communication mechanisms (such as REST or a message bus like RabbitMQ). In addition, there is a lot of microservices architectures and existing approaches for this kind of migration, that are not addressed in this paper, since we have not yet made the decision of which specific composition and migration approach to take. The contrast between this two architectures is dramatic. The original source code has been developed with the assumption of local synchronized execution with low-level shared resources (as memory and stack), while the destination proposes full distribution as assumption, requiring to be adapted or completely redeveloped during a migration process. Alongside it also requires the production of configuration files that allow to find a service composition that produces an equivalent work to the original software. [10, 25] have exposed with detail the case serving monolithic applications as web pages or services, identifying the multiple challenges that come from such a modernization. Further, in this article (section 3), we propose the paradigmatic change and the dependencies counters which measure the amount of entities that are also related with this aspect.

Moving from a desktop application GUI to a web application GUI Graphical user interfaces for Microsoft Access applications are developed using Microsoft Office GUI components. These components generally use the libraries provided by the Microsoft Windows operating system. Even when the GUI can be highly customized, it is no wonder that most of these applications respond to the unified aesthetics of the operative system, using not only the same look and feel, but the same navigation metaphor (from pop-up and dialogue windows to modal and non-modal sub windows). However, the GUIs of a web application typically use HTML and CSS (or derivations) used to customize the look-and-feel of the web page. The originality of aesthetics is encouraged and welcomed, as part of the company identity. Along with that, navigation methods such as pop-up and

dialogues are not welcomed and even banned by many web browsers. These two paradigms are completely different. Thus, a mapping work is necessary in order to be able to transform the VBA graphical interfaces into web interfaces. All of these differences together mean that the migration of graphical constructs, both visual and behavioural ([23]) are developed with assumptions that are not valid on the destination platform, and may require to be adapted or completely redeveloped during a migration process. [7, 12, 13, 18, 19] and many others have contributed in identifying the challenges and validation opportunities of GUI migration. Because of lack of time for further experiments we do not provide metrics for this challenge in this article.

2.2 Source code difficulties

VBA/Macro code to Typescript / Java VBA (Visual Basic for Applications) is a language similar to Visual Basic that requires a host application to run (Access in our case). Microsoft Access projects provide two kinds of source code: The "macros" language – a specific user-friendly language for Microsoft Access– and VBA, a language inspired by Visual Basic, adapted for use in the Microsoft Office context, and Microsoft Access. It is an interpreted language, and developed to run in the context of a database. Furthermore, VBA does not support namespaces nor packages, meaning that all the module functions and classes defined in a project are visible within the project. By other hand, TypeScript is a statically typed programming language that transpiles to JavaScript code. TypeScript allows for development in both the procedural and object-oriented paradigms. Regardless to support for both procedural and object-oriented, an Angular application (the technological target for the GUI), is expected to be written by using mainly object-oriented and component-oriented code. Java is an object-oriented programming language used to develop mainly object-oriented and component oriented software. Both languages are file-oriented, and require the usage of namespaces or packages and importing for visibility. All this code must be generated by instrumentation of algorithms as type inference. Also, the clustering of functions into classes is compulsory in java and mostly desirable in the context of an Angular project ([15, 26]). The difference between grammatical constructs, the existence of namespaces and the file oriented source code are really challenging chasms to cross, and in many cases leading to problems as the impossibility of expressing important semantic on the destination target or requiring to adapt the semantic into concepts that may lead to ambiguity or in extreme cases requiring the redevelopment of a component during a migration process([22]).

Moving the internal structural representation from VBA to Angular As an extension of the language difference, the graphical constructs are designed and developed differently. In Microsoft Access the development of graphical interfaces is done using a wizard (drag and drop). This drag and drop adds graphical controls to the built GUI. Each of these controls and the form itself can be bound to a database table directly. However, GUI development in Angular

is done using the Angular templating language. This language mixes HTML, HTML code generation directives and directives for linking to GUI behavioural code. Angular manual encourage the developers to use interface objects for storing the data produced by the user interaction, and use this objects afterwords as data transfer objects. Therefore, this migration implies the production of state-holding objects, the production of database CRUD code, and to produce calls to remote services thus none of these calls or objects exists in the origin system. [18, 23] have discussed on challenges related with the structural representation of the GUI. [11] also has discussed the user interaction impact over a software adaptation that can shed light also to the potential risks on invalidating the assumptions that were followed during the development of the original software.

3 Metrics

The various difficulties related to the migration of VBA applications (source of the migration) to a web architecture (target of the migration) highlight the incompatibilities that exist between VBA and its paradigm as well as web applications and their paradigm. In this section, we present a set of metrics that measure these incompatibilities and try to quantify the different gaps between source and destination technologies, in an intent to measure the inherent risks to the semi-automatic migration process. These metrics are the result of our experiences as well as our readings in the literature.

3.1 Risks related to the relevance of the source code analysis

Parsers has been used largely in the migration and reverse-engineering literature, as we have seen in different works as [3, 6, 8, 13, 23, 24] and many others. We know also from other studies that Microsoft access projects are complex to analyse [4]. Often, to count with a parser able to parse all possible program on a language is key for the automatic and semi-automatic approaches.

A parser is a program that analyses a string written according to the rules of a formal grammar and produces some kind of output. Among the set of tools we have developed to work with Microsoft Access projects, we have a VBA parser that takes as input a source code and produces a so-called Abstract Syntax Tree (AST). Our parser is based on the grammar of the VBA language as described in the Microsoft Access documentation. For ensuring the completeness we created tests for each of the grammar cases proposed by the documentation. We test also our tool by parsing the Microsoft Northwind Traders [3] example project, covering the full extension of the program.

Risk Despite our efforts to ensure completeness, we found that at least in one of the companies projects, our parser fails to produce an AST in %30 of the modules / class modules, due to unexpected usage of different grammatical constructions.

[3] https://docs.microsoft.com/en-us/powerapps/maker/canvas-apps/northwind-install

The lack of documentation coverage of these grammatical formulas threatens the validity of our semantic analysis since we have to interpret ourselves what these grammatical composition means, opening the door to ambiguity and misinterpretation. The measuring of this risk contributes to the understanding of the *source code related challenges*, depicted in subsection 2.2

Parsing error counter In order to quantify the parsing errors, we use the following Parsing error counter metric In the context of interpreting VBA source code with our parser, we have defined the "SyntaxError" counter. This gives us the amount of parsing errors due to a syntax error (especially those mentioned in the previous paragraph). The higher the value of this counter, the more complex the migration will be.

3.2 The risks of language translation

Programming languages often respond to specific formal grammar, that define the limits and possibilities of a language. These grammatical constructions allow expressing semantics that respond to a proposed programming paradigm. Many approaches to software migration are based on the interpretation of the semantics of the different source artefacts (contained in the language/application to be migrated) and the expression of an equivalent semantic in the target language/-paradigm. Many times the grammatical constructions and or the paradigmatic concepts are incompatible, leading to the inability to express an equivalent semantic.

The risks of mapping entities from one formal grammar to another One of the most important steps in a migration process is the transformation of grammatical entities from the source language to their equivalents in the target language. This consists in associating each element of the source language dataset with one or more elements of the target language dataset. This task is not trivial, because it is possible to have elements of the source language dataset that have no equivalents in the target language.

Risk The lack of equivalent grammatical entity causes loss of semantics during the transformation.[22] . VBA is a language that has a particularly rich grammar. Thus, the same semantics can be expressed in several ways, this is even true when it comes to information flow control or error handling. Therefore, we find it important to map the elements of the VBA dataset that do not have equivalents in the destination environment and to count them when analysing the code of VBA applications. The measuring of this risk contributes to (i) the understanding of the *VBA/Macro code to Typescript / Java related challenges*, since the lack of equivalence implies, depicted in subsection 2.2, and (ii) the complexity of error management, what gives insight of the complexity of the *architectural challenges depicted* in subsection 2.1

Incompatible grammatical construct Counter Different languages provide different ways to control the flow of execution of a program. Control flow management includes normally conditional branching (such as if, else if and switch), loops (such as for, while and repeat) and error management (such as try / catch). There are other less popular and harder to predict such as conditional and unconditional jumps (also known as go-to statements). There are many VBA grammatical entities that are used for error handling and control flow and for which we do not have an equivalent in Typescript / Java, we mention: Resume Label, Resume Empty, Error Resume Next, OnError GoTo, Resume Next, Property, PropertyAccessors. This grammatical entity counter counts the amount of appearances of these entities in a given source code. The higher the value of these counters, the more complex the migration.

The risks of paradigm shift A programming paradigm is a way of approaching computer programming and conceiving problem solutions and their formulation in an appropriate programming language. It thus provides (and determines) the developer's understanding of the execution of his programme. For example, in object-oriented programming, developers may conceive the program as a collection of interacting objects, while in functional programming a program may be seen as a sequence of stateless function evaluations. VBA proposes a hybrid paradigm programming language that aims the development of information technology systems focused on human-machine interaction with direct impact on an integrated database, we find many concepts that do not exist in the paradigms of our target languages.

Risk While VBA allows the usage of functions and procedures, all our destination require the code to be expressed in object oriented fashion. This requires the identification of concerns, and clusterization of variables and functions into potential classes. This kind of problem does not have automatic solutions, since most of the approaches are based on heuristics and multiple results are possible, exposing seriously the consistency of the produced code ([15, 16, 26]). Furthermore, VBA includes many first-class citizen that do not exist in the target environments such as Tables Queries and Macros. Each of these cases are mean to be transformed into something else, risking the loss of semantics and of consistency.[22] . The measuring of this risk contributes to understand (i) the complexity of the *VBA/Macro code to Typescript / Java* related challenge, depicted in subsection 2.2, and (ii) the complexity of *Moving from a monolithic application to a microservices-based application*, depicted in subsection 2.1.

Counter for incompatible entities Among the all the first class citizens of VBA language, we find the followings that have no equivalent on our target platforms: Modules, Tables, Queries, Macros. For measuring the paradigm shift complexity we opt for counting the amount of entities belonging to these categories. The higher the value of this counter, the more complex the migration.

3.3 The risks of using libraries as dependencies

A VBA project interacts with other projects or libraries. These are called dependencies. They can be of three types: (i)"BuiltInDependency": dependencies that are part of Microsoft Access (standard) (ii)"BinaryDependency": dependencies provided by a third party or by Berger-Levrault (iii)"MicrosoftAccessDependency": dependencies developed in Microsoft Access Each kind of dependencies is different, and may not have an equivalent on the target technology. These dependencies need to be taken into account in our migration process.

Risk There are many risks associated with dependencies update or migration [9]. In this article we consider the fact that in different runtime and languages there is a high probability that the same library is not going to be available, [8]. Therefore, each library used by the program, is likely to be changed. The measuring of this risk contributes to understand the general complexity, since affect *all the proposed challenges* depicted in section 2.

Dependency counter In the context of interpreting VBA source code with our parser, we have defined a counter for each type of dependency. The higher the value of these counters, the more complex the migration.

4 The eGRC use case

eGRC is an extremely complex project. Indeed, we have seen that it contains all the types of risks that were mentioned in the previous sections. Therefore, in order to get a clear picture of the proportion of these risks in the eGRC project, we have activated all the metrics mentioned above in our VBA code parser. To understand these metrics, we provide graphs that explain the proportion of risks in eGRC and its sub-projects. As a demographic, eGRC counts with *900.000 LoC* distributed in between *1232 widgets*, *564 reports*, *271 function-modules*, *491 class-modules* and *18 macros*. These different modules are implemented by using a total of *21 libraries*, *1172 queries* and *1437 tables*. This project has in charge a heavy load of business rules since it is in charge of the management of several public services such as electoral planning, civil status and cemetery management.

4.1 The Pareto chart

As largely explained on books as [14], the purpose of the Pareto chart is to identify important individuals in a sample of data. The diagram follows the Pareto principle (also known as the 80-20 principle), which is an empirical phenomenon found in some fields: about 80% of the effects are the product of 20% of the causes. The Pareto chart consists of two graphs: a histogram of frequencies on the measured variable (grouped by project complexity in our case), where the individual values are represented in descending order by bars, and an accumulation line. In our case, we want to identify the 20% of eGRC sub-projects that

cover the 80% of the risks. This will help us to focus only on the important sub-projects. For this reason, we have decided to use the Pareto chart.

4.2 Study of the complexity of syntactic errors

The objective is to show the coverage of our parser by making explicit the grammatical entities that are not recognized by our code analysis tool. The histogram shows the number of syntactic errors for each of the eGRC sub-projects. Example: The magact sub-project has just over 85 syntax errors. The accumulation line shows us the percentage of the cumulative frequency. Example: If we solve the syntax problems in the magact sub-project, we will cover 15% of the total number of syntax errors in the eGRC project, and if we solve the syntax problems in both the magact and magelereu sub-projects, we will cover just over 30% of the total number of syntax errors in the eGRC project. To get 80% coverage we need to solve the syntax problems in the first 7 sub-projects.

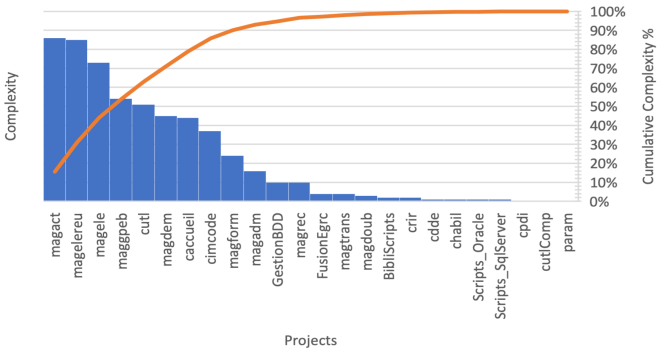

Fig. 1: Study of the complexity of syntactic errors.

4.3 Study of the complexity related to the differences in the source (VBA) and target (Typescript/Java) grammar

The objective is to show the degree of mismatch between the grammar of VBA and Typescript/Java. This consists of counting the number of grammatical elements in the source language (VBA) that have no equivalent in the target languages (Typescript/Java). The histogram shows us the number of occurrences of grammatical elements for which we have no equivalent in each of the eGRC sub-projects. Example: The magact sub-project has just over 9000 occurrences of grammatical elements with no Typescript/Java equivalent. The accumulation line shows us the cumulative frequency percentages. Example: if we solve the equivalence problems in the magact sub-project, we will cover 28% of the total

number of elements without Typescript/Java equivalents in the eGRC project, and if we solve the problems of elements without equivalents in both the magact, and cimcode sub-projects, we will cover a little more than 45% of the total number of elements without Typescript/Java equivalents in the eGRC project. In order to achieve 80% coverage, we need to solve the problems of grammatical elements without equivalents in the first 6 sub-projects.

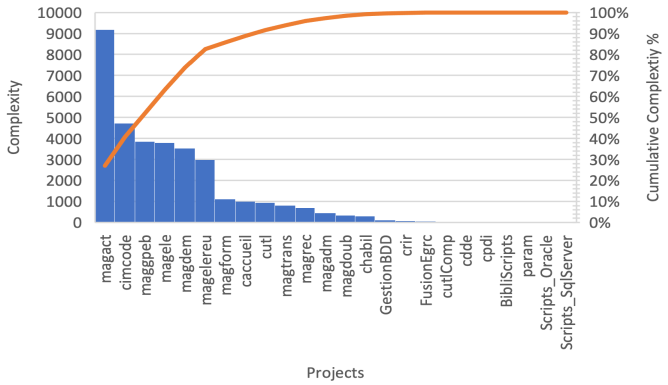

Fig. 2: Study of the complexity related to the differences in the source.

4.4 Study of the complexity related to the paradigm shift

As mentioned in a previous section, many of the notions related to the hybrid paradigm of VBA have no equivalent in the object and component oriented paradigm: Modules, Tables, Queries, Macros, etc. The objective is to show the degree of mismatch between the VBA and Typescript/Java paradigms. This consists of counting the number of paradigm elements in the source language (VBA) that do not have equivalents in the target languages (Typescript/Java). The histogram shows us the number of occurrences of paradigm elements for which we have no equivalents in each of the eGRC sub-projects. Example: the magact sub-project has just over 650 occurrences of paradigm elements without Typescript/Java equivalents. The accumulation line shows us the cumulative frequency percentages. Example: if we solve the equivalence problems in the magele sub-project, we will cover more than 25% of the total number of elements without Typescript/Java equivalents in the eGRC project, and if we solve the problems of elements without equivalents in both the magele, and magelereu sub-projects, we will cover a little more than 40% of the total number of elements without Typescript/Java equivalents in the eGRC project. In order to achieve 80% coverage, we need to solve the problems of paradigm elements without equivalents in the first 8 sub-projects.

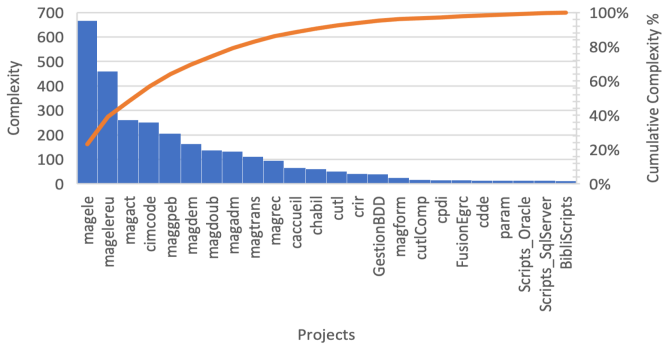

Fig. 3: Study of the complexity related to the paradigm shift.

4.5 Study of the complexity of the use of dependencies

The objective is to show the degree of use of dependencies in each of the eGRC sub-projects. The histogram shows the number of occurrences of dependencies in each of the eGRC sub-projects. Example: The magact sub-project has 17 occurrences of dependencies. The accumulation line shows us the percentage of the cumulative frequency. Example: If we solve the dependencies in the magact, magelereu, magform, maggpeg subprojects, we will only cover 30% of the total number of occurrences of dependencies in the eGRC project. All dependencies must be handled in the same way, this can be very time-consuming.

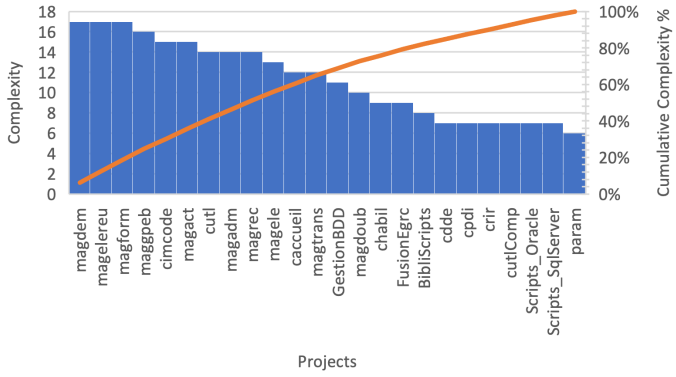

Fig. 4: Study of the complexity of the use of dependencies.

5 Discussion

The metrics we proposed are based on our understanding of the problems encountered in the literature, and some found in our own migrating experience. Regardless the link with previous empirical experiments, our work still shallow, since all the proposed metrics measure complexity in **Nominal Scale** units. Validability and Reliability have not being tested nor enhanced, the experiments using our metrics still few, and our work on the generalization of these metrics still far.

Nominal Scale metrics are useful to understand how many entities do we have in a continuum. This is useful to get an idea of how many of these entities we are bound to find but nothing more. More work is required to be able to establish the contribution of each of these variables to the complexity of the migration.

Validability and Reliability are two of the most important fundamental aspects of measurement [14], and there are required to be measured, validated and empirically proofed. While validability can be enhanced and validated by more exact means, the reliability of the metric requires empirical validation, what implies the requirement of statistical samples on the usage of such metrics.

Uses nevertheless, we have put to work these counters already to be able to select elements from many experiments still to be done. The required expertise to respond to a survey on the utility our context is quite unique.

Our metrics work has been already leveraged by our selves on the detection of projects and files that are more likely to be interesting for the study of layer violation on Microsoft Access elements.

6 Future Work

From this point we plan to wide-up the measurement of complexity to other dimensions of migration such as architectural migration complexity, or the third-party migration complexity. We expect also to work on the unification of the measurement units and on the empirical analysis of the reliability of the metrics in the context of an industrial migration.

7 Conclusion

In this article we contribute a first iteration on Language Migration complexity metrics. We also contribute the interpretation and study the application of such metrics on our current migration project for obtaining a general overview. We contribute also our first use case of these metrics, what is our first step into the iterative enhancement. We discuss on the scales, and our lack of studies on validability and reliability, remarking their importance. We conclude that our metrics give a some understanding of the potential risks during the effort measurement of a migration, but this understanding still fuzzy. More work must be done in the refinement of the proposed metrics to make them useful.

Bibliography

[1] Adjoyan, S., Seriai, A.D., Shatnawi, A.: Service identification based on quality metrics object-oriented legacy system migration towards soa. In: SEKE: Software Engineering and Knowledge Engineering, pp. 1–6, Knowledge Systems Institute Graduate School (2014)

[2] Ahmad, A., Babar, M.A.: A framework for architecture-driven migration of legacy systems to cloud-enabled software. In: Proceedings of the WICSA 2014 Companion Volume, WICSA '14 Companion, Association for Computing Machinery, New York, NY, USA (2014), ISBN 9781450325233

[3] Angulo, G., Martín, D.S., Santos, B., Ferrari, F.C., de Camargo, V.V.: An approach for creating kdm2psm transformation engines in adm context: The rute-k2j case. In: Proceedings of the VII Brazilian Symposium on Software Components, Architectures, and Reuse, p. 92–101, SBCARS '18, Association for Computing Machinery, New York, NY, USA (2018), ISBN 9781450365543

[4] Bragagnolo, S., Anquetil, N., Ducasse, S., Abderrahmane, S., Derras, M.: Analysing microsoft access projects: Building a model in a partially observable domain. In: International Conference on Software and Systems Reuse (ICSR'20), no. 12541 in LNCS (Dec 2020)

[5] Bragagnolo, S., Anquetil, N., Ducasse, S., Seriai, A., Derras, M.: Software Migration: A Theoretical Framework (A Grounded Theory approach on Systematic Literature Review). Empirical Software Engineering (2021)

[6] Bragagnolo, S., Rocha, H., Denker, M., Ducasse, S.: Ethereum query language. In: 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), pp. 1–8 (may 2018)

[7] Bragagnolo, S., Verhaeghe, B., Seriai, A., Derras, M., Etien, A.: Challenges for layout validation: Lessons learned. In: International Conference on the Quality of Information and Communications Technology, QUATIC'2020 (Sep 2020)

[8] Brant, J., Roberts, D., Plendl, B., Prince, J.: Extreme maintenance: Transforming Delphi into C#. In: ICSM'10 (2010)

[9] Cossette, B.E., Walker, R.J.: Seeking the ground truth: A retroactive study on the evolution and migration of software libraries. In: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, pp. 55:1–55:11, FSE '12, ACM, New York, NY, USA (2012), ISBN 978-1-4503-1614-9

[10] De Lucia, A., Francese, R., Scanniello, G., Tortora, G.: Developing legacy system migration methods and tools for technology transfer. Software: Practice and Experience **38**(13), 1333–1364 (2008)

[11] DeLine, R., Zelesnik, G., Shaw, M.: Lessons on converting batch systems to support interaction: Experience report. In: Proceedings of the 19th International Conference on Software Engineering, p. 195 to 204, ICSE '97, Association for Computing Machinery, New York, NY, USA (1997), ISBN 0897919149

[12] Di Santo, G., Zimeo, E.: Reversing guis to ximl descriptions for the adaptation to heterogeneous devices. In: Proceedings of the 2007 ACM Symposium on Applied Computing, p. 1456 to 1460, SAC '07, Association for Computing Machinery, New York, NY, USA (2007), ISBN 1595934804

[13] Garcés, K., Casallas, R., Álvarez, C., Sandoval, E., Salamanca, A., Viera, F., Melo, F., Soto, J.M.: White-box modernization of legacy applications: The oracle forms case study. Computer Standards & Interfaces pp. 110–122 (Oct 2017)
[14] Kan, S.H.: Metrics and models in software quality engineering. O'Reilly (2006)
[15] Kontogiannis, K., Martin, J., Wong, K., Gregory, R., Müller, H., Mylopoulos, J.: Code migration through transformations: An experience report. In: Proceedings of the 1998 Conference of the Centre for Advanced Studies on Collaborative Research, p. 13, CASCON '98, IBM Press (1998)
[16] Martin, J., Muller, H.A.: C to java migration experiences. In: Proceedings of the Sixth European Conference on Software Maintenance and Reengineering, pp. 143–153, IEEE (2002)
[17] Mateus, B.G., Martinez, M., Kolski, C.: An experience-based recommendation system to support migrations of android applications from java to kotlin (2021)
[18] Moore, Rugaber, Seaver: Knowledge-based user interface migration. In: Proceedings 1994 International Conference on Software Maintenance, pp. 72–79, IEEE Comput. Soc. Press (1994), ISBN 978-0-8186-6330-7
[19] Moore, M.M.: Rule-based detection for reverse engineering user interfaces. In: Proceedings of WCRE'96: 4rd Working Conference on Reverse Engineering, pp. 42–48, IEEE (1996)
[20] de Souza, P., McNair, A., Jahnke, J.H.: Network-centric migration of embedded control software: a case study. In: Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research, pp. 54–65 (2003)
[21] Su, X., Yang, X., Li, J., Wu, D.: Parallel iterative reengineering model of legacy systems. In: 2009 IEEE International Conference on Systems, Man and Cybernetics, pp. 4054–4058, IEEE (2009)
[22] Terekhov, A.A., Verhoef, C.: The realities of language conversions. IEEE Software **17**(6), 111–124 (Nov 2000), ISSN 0740-7459
[23] Verhaeghe, B., Etien, A., Anquetil, N., Seriai, A., Deruelle, L., Ducasse, S., Derras, M.: GUI migration using MDE from GWT to Angular 6: An industrial case. In: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER'19), pp. 579–583, Hangzhou, China (2019)
[24] Williams, J.R., Paige, R.F., Polack, F.A.C.: Searching for model migration strategies. In: Proceedings of the 6th International Workshop on Models and Evolution, p. 39 to 44, ME '12, Association for Computing Machinery, New York, NY, USA (2012), ISBN 9781450317986
[25] Zhang, Z., Yang, H.: Incubating services in legacy systems for architectural migration. In: 11th Asia-Pacific Software Engineering Conference, p. 196 to 203, IEEE (2004)
[26] Zou, Y., Kontogiannis, K.: A framework for migrating procedural code to object-oriented platforms. In: Proceedings Eighth Asia-Pacific Software Engineering Conference, p. 390 to 399, IEEE (2001)