Appendix To Software Migration: A Theoretical Framework A Grounded Theory approach on Systematic Literature Review

Santiago Bragagnolo, Nicolas Anquetil, Stéphane Ducasse, Abderrahmane Seriai, Mustapha Derras

Noname manuscript No.
(will be inserted by the editor)

Appendix To Software Migration: A Theoretical Framework

A Grounded Theory approach on Systematic Literature Review

Santiago Bragagnolo · Nicolas Anquetil · Stephane Ducasse · Abderrahmane Seriai · Mustapha Derras

Received: date / Accepted: date

Abstract Software migration has been a research subject for a long time. Major research and industrial implementations have been conducted, shaping not only the techniques available nowadays, but also a good part of Software evolution jargon. To understand systematically the literature and grasp the major concepts is challenging and time consuming. Even more, research evolves, and it does based on the assumption that there is a single meaning that we all share redounding in the pollution of words with multiple and many times opposite meanings. In our quest to understand, share and contribute scientifically in this domain, we recognise this situation as a problem. To tackle down this problem we propose a taxonomy on the subject as a theoretical framework grounded on a systematic literature review. In this study we contribute a bottom-up taxonomy that links from the object of a migration to

Santiago Bragagnolo
Université de Lille, CNRS, Inria, Centrale Lille,
UMR 9189 – CRIStAL France,
Berger-Levrault
E-mail: santiago.bragagnolo@berger-levrault.com

Nicolas Anquetil
Université de Lille, CNRS, Inria, Centrale Lille,
UMR 9189 – CRIStAL France,
E-mail: nicolas.anquetil@inria.fr

Stephane Ducasse
Université de Lille, CNRS, Inria, Centrale Lille,
UMR 9189 – CRIStAL France,
E-mail: stephane.ducasse@inria.fr

Abderrahmane Seriai
Berger-Levrault, France
E-mail: abderrahmane.seriai@berger-levrault.com

Mustapha Derras
Berger-Levrault, France
E-mail: mustapha.derras@berger-levrault.com

the procedure nature migration, passing by migration drivers, objectives and approaches. We contribute a classification of all our readings, and a list of research directions discovered on the process of this study.

Keywords Software Reengineering · Migration · Modernisation · Taxonomy

1 Appendix I: Detailed Reporting

According to the different parts that appear on most of the migration projects we discover with this SLR, we propose our reporting to be split as follow: legacy system, drivers, options to migration, target, migration process, iterativity and incrementality.

1.1 Legacy Systems

[21] points that about legacy systems, their **lack of documentation** "Legacy systems are poorly documented, and poorly understood. Many times the only thing left are the source code and binary files". And how **complexity** is what turns a system into a legacy system: "Software systems eventually to become legacy systems is the fact that a great deal of the complexity in software systems is purely accidental". And explains that "Accidental complexity of a system and make the system harder to understand.", implying the impact on maintenance.

[16] remarks the **critical nature of legacy systems** "Many businesses are choosing to reengineer their critical applications to better fit the needs of the organisation and to take advantage of new technologies", as well as remarking the legacy systems to be in charge of being responsible of the organisation needs.

[13] Points out the **difficulty of maintenance** related to legacy systems "Some components of the system are not owned by any member of the development team and are therefore very difficult to maintain", and the inability of software teams to take over this complexity: "Not surprisingly, the team is reluctant to perform radical changes to its structure since this may affect negatively its overall performance.". Finally it also sustain the **critical nature** of legacy systems, and it **business value**: "Legacy software systems are software systems that have been in operation for many years, have evolved to meet changing organisational demands and computing platforms, and are often mission critical for the organisation that owns and operates them."

[27] implies the **lack of maintainability** with "The target migrant system is intended to be more maintainable than the original system and to possess an open programmatic interface (API)". and recognise the impossibility of more reasonable methods, because of the **lack of documentation** "The method is based on documentation and informal information, such as user manuals, requirement and design specifications, and naming convention. However, for legacy systems, the external information is not always available".

It does also implies by "A procedural legacy system can be migrated to an object-oriented platform" and "The main objective to leverage the business value of legacy software systems into Web-enabled environment" that legacy system **lack some technological feature**.

[14] stands also for legacy systems to be **critical systems** "Candidates for language conversion are usually the most critical systems of a business; thus, an emphasis must be put on the reliability of the conversion process"

[26] stands also for the critical aspect "Legacy systems refer to mission critical software systems that are still in operation", as well as the **decadence** of the state "Quality and expected operational life is constantly deteriorating due to prolonged maintenance and technology updates".

[24] claims that the legacy systems has been built to serve as **monolithic** applications "When legacy systems were built, they served as monolithic applications containing all the business rules and processing logic". And that they **lack some technological feature** that would give some new flexibility "Legacy systems can be leveraged in both service requester side and service provider side", "The legacy system for ACCA course management is based on client/server architecture.All the business logic is implemented in the client side. It is a typical fat-client application". It does also stand by the **decadence of the code** "In the legacy code refinement phase, an optimisation removed dead code, refactored several classes and functions, and refined the interfaces.", and it stands as well for the **lack of proper documentation** "Most of the code suffers from poor documentation.".

[7] implies the **reliability** of the legacy systems, and their **critic nature** "Many legacy systems are business critical and can operate up to 24 h a day". It also relates their status of legacy system to a **legacy language** "They are written in some legacy language such as COBOL". It also classifies the legacy systems according to their de-composability "Legacy system is classified as being decomposable, semi-decomposable, or non-decomposable". By other hand [20] "Legacy systems are old computer systems or application program that continues to be used". This systems may be considered as **decadent systems**, since they "have suffered from many problems, such as outdated technologies decayed architectures, lack of documents", implying finally their **expensive maintenance cost** "These problems make the maintenance cost higher and higher".

[1] implies the**lack of some architectural variable or technological feature** "Therefore, during legacy systems migration, the evolved architecture must satisfy the elasticity requirements". It does also remarks the **complexity and heterogeneity** of legacy systems "In a practical context it is common to have legacy code represented in different programming languages and developed on different platforms." and their **critical nature** for the organisation "increasing number of organisational business-critical applications so called legacy systems are being migrated to cloud computing"

Finally, [12] remarks the **decadence** and **complexity** of legacy systems "A number of key organisations are sustaining the decades old complex legacy systems despite their types of services and operating environment.", and their

architectural nature "For any traditional software system, following a layered architecture, three main constituent layers are presentation layer for user-application interaction"

1.2 Drivers

As drivers we point out the reasons that are used as a reason to allocate a large amount of resources for migration.

[15] hypothesise that "your company is moving from PCs to workstations, and the program must be migrated. Compounding the problem is the fact that the user interface technology in the PC version is proprietary and will have to be replaced" would be an unavoidable problem. Or maybe "Other variants include grafting a graphical user interface (GUI) onto a batch application, upgrading character-oriented display software to bit- mapped workstations, and keeping software up-to-date with respect to industry standards"

[21] remarks the importance of human resources "Can you bring a new person into the project, and, given nothing but readily available project materials, expect the new person to come up to speed on the system?", implying that ease the understanding of a software, would be a motive. It does also points to other potential motive for conducting some large engineering task on existing code "the problems with existing OO systems or libraries have two rather orthogonal aspects: **getting better code** and **using old code in new ways**".

[13] Remarks the reason for management to find solutions: "Managing such systems is difficult because of frequent breakdowns, spiralling **maintenance costs** and **shortages of qualified personnel** who are willing to work with obsolete programming languages and operating platforms.". It does also remarks the expected outcomes of the project, as a reason for the project to be accepted "There are requirements such as the migrant code must **run at least as fast as the original code**, or the migrant system must be **easier to maintain**".

[27] Points the general internet revolution architectures related to the "With the widespread use of the Internet and pervasive computing technologies, distributed object technologies have been widely adopted to construct network-centric architectures, using Web Services, CORBA, and DCOM. **This use has triggered a plethora of research** with the main objective to leverage the business value of legacy software systems into Web-enabled environments"

[14] Indicates relationship with competitiveness "Electronic commerce over the Internet plays an important role in **today's economy**. To stay competitive in the global marketplace, companies have to offer their services and products to current and prospective customers online through Internet client". It does also points the possible access to the computational capability of the clients: "As soon as value-added services are to be offered, it is desirable to have the

Internet client not only access data from the company's information systems, but also perform computations on the data"

[19] Indicates a **hardware and software lack** as a driver "Herzberg Institute of Astrophysics(HIA) is interested in migrating the application to the network-enabled, component based platform of our industrial partner".

[26] claims as reason "To **leverage business values entailed in such systems**, a possible solution is to migrate selected parts of such systems to modern platforms and designs", and points out the possibility of reuse and unlock specific technology "With properties, such as information hiding, inheritance and polymorphism inherent in object-oriented designs, essential parts of such a reengineered system can be **reused or integrated** with other applications using network centric Web technologies, enterprise integration solutions, or distributed systems."

[9] claims the need to unlock new running environments for widening usage: "heterogeneity demands for GUIs adaptation to a variety of **hardware and software platforms**" .

[7] brings some business factors "often other non-technical factors influence the decision as to how to deal with legacy systems, factors such as the need to move to a modern Internet based infrastructure in order to **remain competitive in the global market**".

[5] briefly explains the industrial partner pain: "These include difficulty in **hiring qualified programmers, limited availability of third-party libraries, lack of vendor support for the language tools, limited or delayed support for new technologies**, and enterprise integration after an acquisition". It does also details one specific point "third-party libraries that their products relied on were **becoming unsupported and libraries for new functionality were not available**.". And it points out the strongest arguments "But the most compelling reason for their switch was ProfitStars' **desire to utilise the latest .NET technologies** with their legacy applications. Another contributing factor for the migration was that they were **acquired by** Jack Henry & Associates, which primarily used C# in their program"

[6] links the evolution of libraries as driver "software **libraries are susceptible to the same environmental pressures** to change that all software systems face. As a result, library maintainers need to evolve their systems in ways that sometimes result in an **incompatibility between the old and new versions of their API**. This raises a dilemma for both API developers and client developers: whether to migrate to the new API version and endure the adaptive effort, or to refuse to migrate"

[4] Claims cost reduction and architectural variables to be the reason of a large migration phenomena "Since cloud computing aims at **improving the quality of delivered services** concerning rapid elasticity and high availability, as well as at reducing costs of software operation by a **'pay-as-you-go' pricing model**, there is an increasing need to move legacy software into the cloud of services"

[1] uses as argument a citation to a survey "A recent study of Capegemini – based on interviews with 460 business and IT executives – has concluded that migration of legacy applications to cloud-based system is driven by the organisational needs to achieve **business agility and cost efficiency**". It also exposes research drivers for the migration field "Our focus on architecture-driven migration was motivated by the potential benefits" and to propose an approach "that enable a **systematic cloudification** of existing on premise legacy software to clouds (A Framework for Architecture-driven Migration of Legacy".

[12] points many architectural variables to be main drivers "the advantages like **elastic storage, load balancing services, auto scalability over a distributed network**, and virtual private cloud". Alongside with availability "Microsoft's data centres are the one where the data gets stored safely and with **24x7 availability** support." . All variables of the cloud architectural paradigm.

[10] accuses as driver the passage of time "evolution is motivated by different reasons such as the **obsolescence of a technology**", the users requirements "The pressure of users", and some extraordinary events such as company fusion "the need to build a **single coherent information system** when companies merge".

[22] points "**GWT is no longer being updated with only one major release since 2015**. As a consequence, Berger-Levrault decided to migrate its applications to Angular 6 (GUI Migration using MDE from GWT to Angular 6".

1.3 Proposals profiling: Contexts, proposals and definitions: What is to migrate?

In this subsection we give context, proposition and how do they perceive migration, based on their claims.

[15] Takes place in the *context* of multi-platform GUI migration.

Proposes to map origin and destination widgets based on features and requirements, and use it for assessing the generation of the new GUI.

Recognises migration as knowledge mapping and code generation "The migration system then determines which widget has been selected, retrieves the defining properties and constructs a request for CLASSIC. CLASSIC, in turn, infers a list of replacement candidates"

[11] Takes place in the *context* of discussing and unifying the research directions of programming languages and software engineering.

Proposes To leverage each other technological developments and research problems. (Such as language interpretation usable for language compiling and for software static analysis)

Recognises migration as a problem to be addressed with technologies emerging from programming language theorisation and implementation.

[16] Takes place in the *context* of text-based UI to GUI migration.

Proposes to map code/ast patterns to kinds of widgets. By example the pattern in the listing should match a kind of menu widget.

```
choice := getanswer(lastchoice);
case choice of
        'a': answer
end;
```

Recognises migration to be a reengineering process to enable new technologies. "With the advent of client-server technology, open systems, and high-powered graphical workstations, many businesses are choosing to reengineer their critical applications to better fit the needs of the organisation and to take advantage of new technologies." "If the user interface components can be extracted from the computational code and expressed in an abstract model, then maintenance and future migrations can be made much simpler, since only the user interface components would need to evolve".

[**8**] Takes place in the *context* of the shifting of control paradigm of a software, going from batch to interactive usage.

Proposes to use source code analysis to analyse and understand the main control and flow assumptions used during the software development. Some of this assumptions are related with Execution duration, Incremental processing (in an interactive software we have stages to persist of a process, in a batch process all intermediate information is volatile), Error handling, Error reporting, etc. And then conduct adaptive reengineering for modifying the system into an interactive system. The article provides a large amount of advices on how to conduct the adaptive reengineering.

Recognises migration as an evolutionary change that implies at least the adaptation of the system "One of the most common evolutionary changes is migration from batch to interactive use. This minimally requires adding capability to invoke the system's operations one at a time and to provide visibility into the intermediate results."

[**17**] Takes place in the *context* of analysing software engineering migration related problem.

Proposes a collection of strategies and techniques.

Recognises migration as a reengineering process "Software reengineering is the process of examining and altering a subject system to reconstitute it in a new form. The spectrum of reengineering activities includes re-documentation, restructuring of source code, transformation of source code, abstraction recovery, and reimplementation. "

[**13**] Takes place in the *context* of PL/IX to C language migration.

Proposes to map data types and AST node types for code transliteration.

Recognises migration as a risky large code transliteration with impact on three levels: code, paradigm usage enhancement and architecture. "Unfortunately, re-engineering a large system not only requires a very high commitment of human resources, but also introduces a number of risk factors such as integration errors, introduction of faulty code and non-compliance with global constraints on performance, maintainability, etc." "At lowest levels, migration

takes the form of transforming (or, "transliterating") the code from one language into another." " level At higher levels, the structure of the system may be changed as well to make it, for instance, more object-oriented. " "At still higher levels, the global architecture of the system may be changed as part of the migration process."

[27] Takes place in the *context* the creation of a framework for tackling down procedural to object-oriented paradigm migration.

Proposes the usage of an XML language agnostic model to represent syntactical constructs of the origin source code (allowing the merging of different languages sources). It proposes also the usage of different heuristics over the unified representation of the origin to infer different possible object-oriented paradigm constructs. From hard concepts such as classes and methods, to more soft like inheritance and polymorphism. It also proposes devices to choose in between different possible results, and resulting target generation.

Recognises migration Migration is multiple-possible results process. A chance of quality enhancement. A device to enable new technologies. "To facilitate the reengineering of legacy services in web enabled environments, our approach proposes a framework where components of a procedural legacy system can be migrated to an object-oriented platform" "When more than one design decision is possible, a collection of source code features is used to assist the user to select the object model that optimises specific metrics and quality characteristics"

[14] Takes place in the *context* of C to Java language migration

Proposes a three step transliteration: (i) unifying to Kernighan and Ritchie C style (ii) data type / type conversions analysis (iii) transliteration .

Recognises different kind of migrations: Dialect conversion (transliteration), API Migration (adaptation), Language Migration (transliteration, adaptation API, adaptation paradigm) "Dialect conversion is the conversion of a program from one dialect of a programming language to another dialect of the same programming language" "API migration is the adaptation of a program to a new set of API" "Language migration is the conversion from one programming language to a different one possibly involving dialect conversion and API migration."

[19] Takes place in the *context* of embedded system for telescope control adaptation to enable networking. In this context of adaptation there is some hardware adaptation and migration imposed by the required technology.

Proposes a process that contemplates the enabling of networking on an existing heterogeneous embedded system. The adaptation process includes the unification of different embedded systems, the hardware adaptation and or exchange, compiler compatibility and eventual migration, and source code adaptation. "Our goal for this case study is to convert HIA's embedded legacy software into a network-centric component and attempt to learn from this process in order to automate it. This component will then be integrated into a network that provides HIA with more effective functionality. " "Evaluate hardware and upgrade if necessary." "Integrate the delivery network service" "Turbo C had no problem compiling the code. Microsoft Visual C++ had several problems

due to old DOS include files. Since AspectC code used the Borland C++ 5.5 compiler, that compiler was also tested on the code" *Recognises migration* as large complex process of adaptive reengineering.

[24] Takes place in the *context* of monolithic C++ to Service oriented Architecture served in Java.

Proposes to mine possible services by clustering by conceptual properties. Modify the candidates to be able to be called from java. Wrap them with java code. Serve the project on Tomcat.

Recognises migration as a reengineering process aiming to restructure the artefacts composition. "the assessment in order to consider many aspects altogether. Reengineering decisions are made according to this assessment." "The sub-domain identification determinate the boundaries of sub-domains and decomposes the whole problem domain. The further sub-domain analysis focuses on modelling a particular part of the problem domain".

[26] Takes place in the *context* of Procedural to object-oriented paradigm migration.

Proposes to Model and apply unified rules to transform the code from procedural to object-oriented, and choose best possible outcome by applying quality metrics comparisons.

Recognises migration as a transformation, that may not be deterministic. "we consider a migration process as a state transition system, denoted by a sequence of transformations that alter a system being migrated" "During the object model extraction process, many alternative object models can be considered" "Similarly, functions and procedures in the original system become primary **candidates** for methods and are attached to the aforementioned identified classes"

[9] Takes place in the *context* of migrating AWT GUI to XIML description for being able to adjust content and support multiple devices with different sizes (phones, pocket-pc, desktop)

Proposes to Interpret the AWT GUI definition of the project, transform it into XIML description, allow the generation of multiple versions according to target device.

Recognises migration as a transformation and adaptation reverse engineering process Transformation and adaptation : The adaptation and rendering phases follow the reverse engineering phase. During the transformation phase, for each XIML widget description produced in the previous phase, the developer prepares multiple XIML descriptions, each one suitable for the platform that can be used to show the user interface"

[25] Takes place in the *context* of threading library migration, from RTLinux to ThreadX.

Proposes to refine an ontology able to give semantic the different entities involved in the libraries, where the ontology instances are meant to be AST nodes, and being able to map the origin nodes to the equivalents on the destination semantic representation.

Recognises migration as a "knowledge intensive activity" "As an inherently knowledge intensive activity, software migration requires a great number of

knowledge covers from expertise to experience in the application domains." [**7**] Takes place in the *context* of an architectural migration, migrating existing software developed as client / server to a web-based environment.

Proposes For the front end, to recreate the Cobol screens into JSP pages. For the backend, the instrumentation of a middleware that bridges the communication in between a light java server and the existing software logic. The software logic would require to be modified and adapted to the interaction with the middleware. The java server access the middleware as a DLL. And the middleware provides a basic set of communications and synchronisation between the COBOL application and the Java server.

Recognises migration as a complex processes with many different approaches, and highly dependant of the internal qualities of the source code and architecture. "Generally, the migration of a legacy system is a complex task, which is influenced by several concerns. One concern, pointed out by Brodie and Stonebraker, is that the migration of a legacy system depends on its decomposability."

[**3**] Takes place in the *context* of enhancing the understanding of source code for monolith to SOA migration.

Proposes to use object-oriented design pattern recognition for understanding and assessing how to rather leverage or deal with those design patterns implications during migration.

Recognises migration to be an adaptation that requires structural insight. "What components can be mined to derive these services? What changes are needed to accomplished the migration?" " Independently from the approach or methodology adopted during the migration process, to understand a system it is crucial to know its software architecture and to recover high level representations of the source code"

[**20**] Takes place in the *context* of enhancing the performance of a migration project.

Proposes to parallelise tasks based on the dependance analysis of the expected tasks.

Recognises migration as a reengineering process "These problems make the maintenance cost higher and higher. Methods are promoted for reengineering legacy systems such as "Chicken Little Strategy" and "Butterfly Methodology" These methodologies all have deficiencies in data migration and access"

[**5**] Takes place in the *context* of translation from Delphi to C#

Proposes to use a rule based rewriting engine that allows to define specific transformation directives based on syntactic node recognition and/or pattern matching.

Recognises migration as the translation of a source code from an origin language with usage of language dependant libraries to a target language with their own libraries. "Our approach for migrating is to build a set of transformation rules that convert the code" "If a library in the old language is substantially different from the new library, often it is easier to create a new component in the new language that behaves like the old one"

[**23**] Takes place in the *context* of the migration of entities from one metamodel version to a new version (such as UML).

Proposes to find the required modifications to reach the acceptance of the new metamodel as a search problem.

Recognises migration as a search problem. "Search-Based Software Engineering (SBSE) is predicated on the fact that it is often easier to determine whether one solution to a problem is better than another, than it is to develop an optimal solution to that problem. Software engineering problems can be reformulated as an optimisation problem, and meta-heuristic algorithms search over the space of possible solutions to the problem.".

[**6**] Takes place in the *context* the study of a corpus of API Change.

Proposes to analyse the changes on an API and to map it to the nature of the impact of these changes on projects that depends on this API. According to the nature of the impact it proposes a categorisation of the solutions based on the difficulty of the adapting process. It also proposes to recommend the possible required adaptations (this recommendations seems to be useful in only 20% of the cases).

Recognises migration The adaptation of existing code from the usage of on library API to the usage of another API of the same library.

"We were also interested in understanding how changes in libraries potentially impacted the effectiveness of techniques that are intended to detect such changes, or to adapt source code to them"

[**18**] Takes place in the *context* of understanding how do industry deals with migration to service architectures.

Proposes to conduct a survey, gathering the point of view of consultants and engineers involved in migration projects.

Recognises migration as a reengineering transforming process. "Traditionally, there are two generic categories of migration lifecycles: the ones addressing incremental migration (e.g. chicken little), and those with complete, sudden, migration (e.g. cold turkey).". "All elicited migration approaches include transforming the pre-existing applications as-a-whole to new target services. Transformation here entails wrapping the legacy system without decomposing it.".

[**4**] Takes place in the *context* of migrating software to cloud environments.

Proposes a unified process to tackle down the planning and strategy of the project.

Recognises migration as a model transformation and code generation process. "Developing and applying model transformations to obtain platform-independent representations of the legacy software from which representations can be derived that are optimised to cloud-based capabilities", "Generating source code that realise the migrated software."

[**1**] Takes place in the *context* of migration of software to cloud environment.

Proposes a theoretical process framework that specialises the ISO reengineering horseshoe, to be applied to the particular case of migration to cloud environment.

Recognises migration as an architecture transformation and reengineering process "The architecture transformation process evolves the legacy architecture towards a service-driven cloud architecture."

[12] Takes place in the *context* defining a roadmap for legacy system modernisation to cloud architecture.

Proposes a roadmap with processes based on systematic review.

Recognises migration as a meta-process with many possible processes to choose according to reasons to migrate (expected architectural benefits), and possible modernisation approaches (different according to the effort and cost)

[10] Takes place in the *context* of the GUI migration of Oracle Forms projects to Java.

Proposes the extraction of a model from origin, passing to a technological agnostic model, then to a target technological model, finally to generate the code.

Recognises migration as a modernisation, model transformation and code generation process. "Steps involved in any modernisation: (1) Understanding and (2) transforming ". "We defined a multi-tiered architecture for the *Forms Modernisation* project. The transformation takes Oracle Forms code as input and produces a **modernised** application as output that implements this architecture in Java technology."

[2], Takes place in the *context* of the implementation of an engine to transform knowledge models to platform specific models (under the understanding of the Architecture driven modernisation from OMG) .

Proposes an engine based on rules for mapping from knowledge model to platform specific model, with the specific implementation of the transformation from model to java platform specific model. "The goal of the activity is twofold; the first one is to define which PSM (Java Model, C# model, Service-Oriented Model, etc) will be used as the target metamodel because the forward transformation will generate instances of this PSM. The second one is to choose a reverse-engineering tool/parser that generates a PSM from source code. The output of this activity are the PSM and the tool for reverse engineering."

Recognises migration as a model transformation and after code generation.

[22] Takes place in the *context* of GWT to Angular 6 GUI migration.

Proposes to use an MDE approach, based on extracting the GUI visual model and behavioural model, transform it into a pivot technology agnostic model, and generate the visual and behavioural code.

Recognises migration as an adaptable model transformation and code generation process "Our approach should be as adaptable as possible to different contexts. For example, it can be used with different source and target languages.". "Our approach should produce code that looks familiar to the developers of the source application. As far as possible, the target code should keep the same structure, identifiers and comments".

1.4 Options to Migration

In this subsection we analyse the claims on options to migration.

[11] Exposes a way to solve the migration issue as a programming language possibility "Another example of a binding-time change is the migration of a system to a new implementation language. This can be **addressed to a limited extent via linguistic mechanisms**—for example, by **making C11 essentially upwards compatible with C**. However, this is clearly only a partial solution: Even though C11 is essentially upwards compatible with C, there are still interesting issues that arise in such a conversion process, in particular, how to discover places in the code where the improved features of C11 can be exploited (such as the ability to have C11 templates)"

[13] Points out some proposals coming from management areas "Not surprisingly, management is looking for alternatives, which sometimes take the route of totally replacing the legacy system in question with a new one, or re-engineering it."

[7] proposes different kind of possibilities (including migration) before stating that migration is the most suitable for their case: "**Typical solutions** include discarding the legacy system and building a replacement system, freezing the system and using it as a component of a new larger system, and modifying the system to give it a new lease on life." Then the article remarks kinds of modifications " **Changes** may range from a **simplification of the system** through a reduction in size and complexity, to preventive maintenance operations such as **re-documentation, restructuring, and reengineering**, to an adaptive maintenance process entailing **interface modification, wrapping, migration**. These alternatives are not mutually exclusive, and the decision as to which approach, or combination of approaches, to take is generally based on an assessment of the quality and business value of the system."

[5] points that "Several approaches have been tried to reimplement large software projects.". The main option to migration cited is "The most common approach is the **Redesigning rewrite**. In this, software development on the original system halts, or at least is significantly reduced, while the team manually reimplements the system in the new language. This reimplementation may be a complete redesign or a manual conversion of the existing source"

[12] Makes migration one of different *modernisation* approaches. Between the other approaches it proposes the following three: "**Replacement** involves the usage of commercially available **off-the-shelf** (COTS) systems and software components to reduce the time and cost of re-writing all of them. According to the structure of the legacy components, the fraction of reuse is decided, ex. a well-defined structured system can undergo an incremental replacement of its components with lesser risk of failure.". "**Black-box Wrapping** involves the new interface to an older system components for easier access. In black box fashion, only interface is changed and no internal functionality is changed. When the size of legacy system is comparatively smaller and rewriting the code results in an expensive way. A high quality business code containing legacy systems is good for wrapping in a quick way. It includes

the collection of user interactions and then reverse engineering and analysis of navigation paths to gather the information about interface and information exchange" . "**Software reengineering** is an umbrella term for processes such as **reverse engineering, refactoring and forward engineering** of an already existing application. It involves the understanding the software (specifications, design and implementation), reimplementing the system with new functionalities and in turn improving the performance and maintainability of the application. Reengineering requires the adjustment and analysis of legacy system including activities such as redesigning, restructuring, re implementing the software system. It involves creation of evolution plan and execution of it. Architecture selection and evolution of system are the main issues to be considered". For this last option, proposes two flavours "*System Re-engineering* It involves the process to reengineer the entire system full in one go. System includes all the data files, platform, source code etc. after the validation of the newer functionality, the implementation of the system is performed. Older legacy systems are retired instantly if they are centralised and can be subjected to periodic upgrades if decentralised." and "*Partial or Incremental Re-engineering* includes basic criteria as re-integration. For small software changes, which do not require alteration in the source code language, reengineered modules are re-integrated into the existing system. But this is applicable in some of the systems where re-structuring is not needed"

[10] Enumerates experiences on modernisation "In cooperation with industry partners, we have carried out projects to tackle different modernisation challenges: (1) Migration from Oracle Forms to Java; (2) Restructuring of Java Enterprise Edition (JEE) applications from monolithic architectures to microservices; and (3) Maintenance of Ruby on Rails (RoR) applications developed by Agile practitioners".

1.5 Processes

[13]

Responds to the *language migration* from PL/IX to C language migration

Process Activities (i) Decompose the general system (ii) Select the migration recognised components (iii) Transformation of its data structures (iv) Generation of supporting utilities (constructs not available on the destination language) (v) Generation of the new system by source code transformation.

Iterativity It achieves iterativity by giving criteria to prioritise and organise the different increments. "Subsystems which degrade in quality and performance due to the lack of resources are also good candidates for migration" "Subsystems between 5-10KLOC are ideal candidates since they can be manually examined and inspected"

Incrementality The proposal achieves incrementality as a risk reduction strategy, by understanding decomposing and selecting pieces of the origin system to be migrated. "Another constraint often adopted in order to reduce the risk of migration projects is that the process must be incremental, i.e., can be

conducted so that certain components of the legacy system are selected and migrated."

[26] and **[27]** To respond to the *procedural to object-oriented* paradigm semi-automatic migration (without runtime modification), by model transformations, proposes

Process Activities (i) the extraction of a procedural representation of the origin source code. "The first phase focuses on representing the procedural code of the system being analysed in terms of an annotated Abstract Syntax Tree (ASTs) or entity relationship model." (ii) the application of transformation on the extracted model "We are interested in transforming a system from its original procedural language implementation to an object-oriented design without altering its external behaviours." (iii) measure and evaluate quality "Each transformation is selected to alter source code features in the original system, and is assessed according to its potential impact on the desired qualities for the target system. Consequently, the resulting system is evaluated against the desired goals"

Incrementality is required to reduce risks "To keep the complexity and the risk of the migration of large system into manageable levels, we provide an incremental approach". It relies on decomposability of the origin system "incremental approach that allows for the decomposition of legacy systems into smaller manageable units". An incremental process requires merging the increment-results "Then an incremental merging process allows for the amalgamation of the different partial object models into an aggregate composite model for the whole system "

Iterativity organises the order of migration increments "the proposed quality driven migration approach is applied iteratively to every cluster. In each cluster, the migration process is divided into a sequence of states and transformations"

Evaluation is done in the form of experiments "WELTAB has been analysed and migrated to an object-oriented platform, from its original C implementation.", migrating to C++ "Finally, the migrant C++ source code is automatically generated, by using the extracted object model"

[20]

Responds to the *meta* problematic of efficient migration process planning.

Process Activities (i) Recognise system components to migrate (ii) Recognise data shared in between components (ii) Schedule the migration process in terms of data dependancy

Iterativity The approach claims iterativity based on the ability of the method to recognise data dependancy and effective prioritisation. The method is able to recognise which tasks can be taken over in parallel, according to the data dependancy. "The approach has three phases and is iterative and incremental, where in each cycle, one transformation rule is developed/evolved"

Evaluation The article proposes to measure Performance Effect Percentage from the proposed roadmap and a sequential roadmap. The article claims the usage of this process during a large migration. The evaluation uses information

gathered from the process, and it compares it to a the expected equivalents from a traditional approach, based on a simulation.

[5]

Responds to the *language migration* from Delphi to C#.

Process Activities (i) Select an independent part of system to translate (ii) Develop/reuse and refine a translating rule (ii) Translate the selected part of the system

Iterativity The project finds iterativity on the division of independent and testable milestones. "The project was broken into 11 milestones that could be independently tested and monitored for their progress. They had a logical progression from simple external utility programs to the main software assets."

Incrementality The project finds incrementality by determining the size of the tasks, and how this tasks will affect the next tasks. To keep some of the initial milestones from including too much source, we had to break some dependencies. We may have needed a particular method or class in some unit (file) for the milestone, but we did not need all classes and methods in the unit for the milestone"

[18] In the context of *understanding* architectural migration to *service oriented architecture*, the article exposes

Process Activities (i) understanding the origin and target software "To gain understanding about the As-Is and To-Be states all enterprises extract the relevant portion of enterprise architecture (EA) of both legacy systems and target service-based systems". Accomplished by interviewing "This is because the knowledge about the preexisting system mainly resides in the stakeholders' minds" (ii) analysing the gap "Gap analysis aims at understanding the gaps between the legacy system and the target service-based system. " (iii) conducting forward engineering "forward engineering includes analysis, design and implementation of software services" (iv) transforming legacy assets to services. "All elicited migration approaches include transforming the preexisting applications as-a-whole to new target services. Transformation here entails wrapping the legacy system without decomposing it."

Planning Planning is an essential part "Not surprisingly, migration in industrial approaches starts with lifecycle planning" . Required for costs and risks management "The panel emphasised that what is especially important for them is to explicitly know the costs and risks of each migration activity. Associating costs and risks to core activities makes the core an even more powerful tool for planning how to do migration."

Incrementality Is to be planned "The plan must reflect important decisions such as the number of increments, or the order by which existing assets are migrated." Achieve by detecting independent elements to migrate "As such, they can select independent increments to migrate legacy elements that can be migrated independently of each other" Increments are related with migration drivers "Not surprisingly, the decisions related to migration increments are made in-line with business goals"

[1] Defines *architectural migration* theoretical framework to cloud architecture.

Process Activities (i) Architecture Migration Planning: (i.i) Feasibility Study. (i.ii) Requirement Analysis. (i.iii) Decision on Cloud Providers (i.iv) Architecture Migration Strategy (ii) Architecture Recovery (ii.i) Consolidation of the Legacy Source Code (ii.ii) Pattern and Style Extraction from Code Model (ii.iii) Legacy Architecture Description (ii.iv) Legacy Architecture Consistency Conformance (iii) Architecture Transformation (iii.i) Architecture Change Implementation (iii.ii) Architecture Property Preservation (iii.iiii) Applying Architecture Transformation Patterns (iv) Architecture-based Development (iv.i) Cloud-service Architecture Description (iv.ii) Application of Cloud Patterns and Styles (iv.iii)Code Generation and Consistency Conformance

Incrementality This article achieves incrementality by task-size subprocesses. "By incremental migration, we mean a decomposition of the coarse-grained architectural migration process into a set of fine-grained sub-processes and activities that can be tackled in a stepwise manner to enable architectural migration."

[7] Responds to *an architectural migration* to enable web technologies on a client / server system.

Process Activities (i) Assessing the current systems (ii) Defining the target environment (iii) Identifying problems and risks (iv) Pre-processing (v) Wrapping (vi) GUI-Reengineering (vii) Integration and Deployment

Incrementality Achieves incrementality by delaying a more dramatic migration, but yet allowing modernisation by white-box-wrapping .

"Legacy information systems or part of them can be encapsulated in a modernised set of legacy components, thus enabling the integration with newly developed or purchased applications through the wrapper interface. This approach also enables an incremental migration of the original system". "The black-box wrapping techniques have the advantage that they can be reused without intimate knowledge of the component's internals. Unfortunately, incremental migration strategies are not effectively supported as the wrapped component can be neither customised nor adapted "

Evaluation The article proposes a pilot project using the process and technology proposed. It measures the results of the pilot

[10]

Responds to *GUI* semi-automatic migration, by using a model driven engineering approach.

Process Activities (i) Extracting a model from the origin "A parser receives the legacy application as a set of legacy artefacts and produces a low-level model (so-called Technology Specific Model). " (ii) Refining an agnostic model from the first model "A transformation takes the Technology Specific Model as input and generates a model that conforms to the Technology Agnostic Metamodel" (iii) Configure the target architecture "We classify this information into the following concerns: data access, quality attributes, and configuration process" (iv) Generates target code "model-to-text transformation creates the new code from the elements in the technology agnostic model."

Iterativity responds to the process of refinement of the migrating process "each iteration, the developer can configure how to transform a set of legacy

artefacts (third step), and generate, complete, and test the corresponding new code (steps 4 to 6), until satisfying the scope of a given modernisation project"

Evaluation "The evaluation had two parts: a proof of concept and a pilot study. Asesoftware provided five applications that were taken as the dataset to perform the proof of concept". The case of the study is a based on comparison "Our study is aimed at analysing two methods for Oracle Forms application transformation: "manual" and "(semi)automated through the white-box method", for the purpose of comparison with respect to their productivity and quality from the perspective of practitioners"

[2]

Responds to *Knowledge model to platform specific model* automatic transformation, by model transformation based on model driven engineering standard.

Process Activities (i) Defining the Code Snippet to be Represented by the Models (ii) Generating the PSM origin (iii) Generating the KDM instance (iv) Transforming KDM instance onto PSM target (v) Generating destination snippet.

Iterativity & Incrementality The approach claims iterativity based on the development one at the time of the transformation rules. The approach claims incrementality on increments of rule-size, because each rule can be developed independently. "The approach has three phases and is iterative and incremental, where in each cycle, one transformation rule is developed/evolved. "

Evaluation

The evaluation is based on the correctness on the transformations. For measuring the proposal is to apply the transformations to randomly picked projects from GitHub that respond to some basic criteria such as being developed in Java. Afterwards the evaluation follows by applying the different proposed rules, and measuring the results.

References

1. 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). DOI 10.1145/2578128.2578232. URL https://doi.org/10.1145/2578128.2578232
2. 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, SBCARS '18, p. 92–101. Association for Computing Machinery, New York, NY, USA (2018). DOI 10.1145/3267183.3267193. URL https://doi.org/10.1145/3267183.3267193
3. Arcelli, F., Tosi, C., Zanoni, M.: Can design pattern detection be useful for legacy systemmigration towards soa? In: Proceedings of the 2nd International Workshop on Systems Development in SOA Environments, SDSOA '08, p. 63 to 68. Association for Computing Machinery, New York, NY, USA (2008). DOI 10.1145/1370916.1370932. URL https://doi.org/10.1145/1370916.1370932
4. Bergmayr, A., Bruneliere, H., Izquierdo, J.L.C., Gorronogoitia, J., Kousiouris, G., Kyriazis, D., Langer, P., Menychtas, A., Orue-Echevarria, L., Pezuela, C., et al.: Migrating

legacy software to the cloud with artist. In: 2013 17th European Conference on Software Maintenance and Reengineering, pp. 465–468. IEEE (2013)
5. Brant, J., Roberts, D., Plendl, B., Prince, J.: Extreme maintenance: Transforming Delphi into C#. In: ICSM'10 (2010)
6. 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, FSE '12, pp. 55:1–55:11. ACM, New York, NY, USA (2012). DOI 10.1145/2393596.2393661. URL http://doi.acm.org/10.1145/2393596.2393661
7. 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). DOI https://doi.org/10.1002/spe.870. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.870
8. 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, ICSE '97, p. 195 to 204. Association for Computing Machinery, New York, NY, USA (1997). DOI 10.1145/253228.253267. URL https://doi.org/10.1145/253228.253267
9. 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, SAC '07, p. 1456 to 1460. Association for Computing Machinery, New York, NY, USA (2007). DOI 10.1145/1244002.1244314. URL https://doi.org/10.1145/1244002.1244314
10. 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 (2017). DOI https://doi.org/10.1016/j.csi.2017.10.004
11. Gunter, C., Mitchell, J., Notkin, D.: Strategic directions in software engineering and programming languages. ACM Comput. Surv. **28**(4), 727 to 737 (1996). DOI 10.1145/242223.242283. URL https://doi.org/10.1145/242223.242283
12. Jain, S., Chana, I.: Modernization of legacy systems: A generalised roadmap. In: Proceedings of the Sixth International Conference on Computer and Communication Technology 2015, ICCCT '15, p. 62 to 67. Association for Computing Machinery, New York, NY, USA (2015). DOI 10.1145/2818567.2818579. URL https://doi.org/10.1145/2818567.2818579
13. 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, CASCON '98, p. 13. IBM Press (1998)
14. 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)
15. Moore, Rugaber, Seaver: Knowledge-based user interface migration. In: Proceedings 1994 International Conference on Software Maintenance, pp. 72–79. IEEE Comput. Soc. Press (1994). DOI 10.1109/ICSM.1994.336788. URL http://ieeexplore.ieee.org/document/336788/
16. 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)
17. Müller, H.A.: Reverse engineering strategies for software migration (tutorial). In: Proceedings of the 19th International Conference on Software Engineering, ICSE '97, p. 659 to 660. Association for Computing Machinery, New York, NY, USA (1997). DOI 10.1145/253228.253799. URL https://doi.org/10.1145/253228.253799
18. Razavian, M., Lago, P.: A lean and mean strategy for migration to services. In: Proceedings of the WICSA/ECSA 2012 Companion Volume, WICSA/ECSA '12, p. 61 to 68. Association for Computing Machinery, New York, NY, USA (2012). DOI 10.1145/2361999.2362009. URL https://doi.org/10.1145/2361999.2362009
19. 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)

20. 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)
21. Taivalsaari, A., Trauter, R., Casais, E.: Workshop on object-oriented legacy systems and software evolution. SIGPLAN OOPS Mess. **6**(4), 180 to 185 (1995). DOI 10.1145/260111.260276. URL https://doi.org/10.1145/260111.260276
22. 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). DOI 10.1109/SANER.2019.8667989. URL https://hal.inria.fr/hal-02019015
23. 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, ME '12, p. 39 to 44. Association for Computing Machinery, New York, NY, USA (2012). DOI 10.1145/2523599.2523607. URL https://doi.org/10.1145/2523599.2523607
24. 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)
25. Zhou, H., Kang, J., Chen, F., Yang, H.: Optima: an ontology-based platform-specific software migration approach. In: Seventh International Conference on Quality Software (QSIC 2007), pp. 143–152. IEEE (2007)
26. Zou, Y.: Quality driven software migration of procedural code to object-oriented design. In: 21st IEEE International Conference on Software Maintenance (ICSM'05), pp. 709–713. IEEE (2005)
27. 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)