sciendo

VOLUME 16, N° 1 2022
Journal of Automation, Mobile Robotics and Intelligent Systems

Migrating Monoliths to Microservices
Integrating Robotic Process Automation into the Migration Approach

Submitted: 21th July 2021; accepted: 29th of March 2022

Burkhard Bamberger, Bastian Körber

DOI: 10.14313/JAMRIS/1-2022/8

Abstract:
This research should help scholars and practitioners to manage the transition of monolithic legacy application systems to microservices and to better understand the migration process, its steps, and its characteristics. It should also provide guidance on how best to approach the migration process. We performed a systematic literature review and analyzed migration approaches presented by other research. We propose leveraging Robotic Process Automation technology to extract business logic and create and deploy bots, which are then used to mimic microservices. In essence, this represents a novel use case of integrating RPA technology into the migration approach in order to reduce uncertainty and risk of failure.

Keywords: *Microservice, RPA, Monolithic Architecture, Reverse Code Engineering, Migration*

1. Introduction

Organizations increasingly rely on information technology to create value. New digital business models, changes to business processes, and automation of tasks previously performed by office workers require investment in hardware and software. Management is then tasked with deciding how to best allocate resources in the field of information technology. Capital expenditure and implementation costs associated with introducing new or updating existing application systems are material. Therefore, management needs to assure that the organization's information systems adequately support its business strategy.

Many application systems, however, were introduced years ago and have been continuously customized and upgraded. These "legacy systems" were designed following a monolithic architecture style that dates back to legacy mainframe computers [29]. Replacing or updating existing legacy systems to address changes in business strategy or higher system load requirements due to increased transactional volume is a key challenge, especially since monoliths often lack scalability.

Monolithic applications are self-contained, consist of a single code base, include every single functionality, and are easily implemented [34]. Modularity, however, is not considered as a design principle [35]; therefore, it functions within a monolith collectively sharing resources on the host system, which limits scalability [32]. Developing, maintaining and changing monolith applications also becomes increasingly difficult and slow, as they tend to grow in size and complexity [44]. Cloud services make automatic scaling easy and cost-efficient; however, large monolithic applications cannot take full advantage of these functionalities [28]. Amazon, Netflix, LinkedIn, Soundcloud and other leading technology companies were among the first to transition to microservices [16].

Microservices represent a fundamentally different architectural design principle. Prioritizing decentralization over centralization is the common pattern guiding the development of distributed applications on cloud platforms [36]. Suites of small, independent services, sharing as little as possible, each running in its own process, and communicating with lightweight mechanisms are key characteristics of this architectural design style [16] [17]. Each single service capsules small deployable chunks of application logic, built around business capabilities [24] which is separately developed and deployed by a small, dedicated team [24]. This should allow for agile development and operation [10] resulting in "high availability and redundancy, automatic scaling, easier infrastructure management and compliance with latest security standards…" [8].

Therefore, microservice-based applications are advantageous from a flexibility, scalability, complexity, agility, and maintainability perspective [36]. In microservices, business processes are split up into separate, manageable components, as task logic is codified within distinct, easily identifiable services. By comparison, task logic is hard to locate and change within monolithic systems. Since business processes are made up of multiple "logically-related tasks performed to achieve a defined business outcome" [14], changes in corporate strategy necessitate adaptions to the application logic. Therefore, in creating new application systems, microservices are favored over monolithic principles, as early adopters have demonstrated. In contrast, when legacy systems were designed, applications were almost exclusively built according to monolithic architectural principles. If monoliths require major modifications to address new business requirements or improve scalability, management needs to decide whether to (a) buy and implement new software, (b) build new application from scratch,

(c) patch-up the legacy system by modifying the existing code base to reflect the desired changes or (d) migrate the monolith toward microservices to improve flexibility, scalability, agility and maintainability. This paper researches option (d), the migration of monoliths to microservices.

Robotic Process Automation (RPA) is a technology designed to automate business processes or task sequences without changing existing back-end systems [46] by building and deploying digital agents that mimic activities of human users in a variety of different application systems [30]. This lightweight automation approach via the application´s user interface opens up previously untapped automation potentials due to its ease of use, speed of implementation, and cost-effectiveness (Czarnecki & Fettke, 2021).

Despite the compelling advantages of microservices, migrations are rare [35]. This may be rooted in a perceived risk and uncertainty surrounding the migration. Our literature review revealed that except for comprehensive migration approaches presented by Maisto et al. (2020) and Megargel et al. (2020), most research focuses on code reverse engineering, a multitude of methods aimed at extracting business logic from the monolith's code base. We describe how migration processes are approached and which steps are critical. Rather than reengineering business logic from legacy code, we propose to use RPA to create and deploy bots, which mimic microservices. This not only provides an alternative solution to the critical extraction phase, but the bots also serve to improve and speed up the testing of microservices. This integrated approach should help reduce risk associated with migrating monoliths to microservices and, therefore, be of interest to both academics and practitioners.

2. Research Questions and Methods

Migrating from monolith to microservices is a complex undertaking. It requires a thorough understanding of the elements and characteristics of both the starting point (monolith) and end (microservices). A migration approach needs to address how to transform crucial elements, what steps to take and which steps warrant special attention, as they are mission critical.

Our key research questions are: (1) What migration approaches have other researchers or practitioners presented? (2) What are the benefits and challenges associated with each approach? (3) What alternative solutions or which modifications help reduce uncertainty and risk of failure?

Microservices is a relatively new design in software architecture, having gained popularity in the wake of cloud technology. Migrating monoliths to microservices has received limited attention from academia and migration to microservices is a rare phenomenon in practice [35]. There are few earlier studies; however, research efforts are at a preliminary stage. Therefore, qualitative, exploratory research seems most appropriate to establish an understanding of the migration process, its steps and characteristics, clarification on proposed approaches, as well as new ideas complementing existing approaches. Furthermore, it can help to structure, clarify, and prioritize future research and assist practitioners with resource allocation.

This research applies a design science approach, aiming at introducing new and innovative artifacts as well as the process of creating artefacts [42]. There are three stages of the research process [21]: First, we established the relevance of our research by inquiring and documenting the state of the art process of migration. Then, we modified and enhanced an existing artefact (migration approach), evaluated earlier versions and refined them. Lastly, we assured research rigor by leveraging the existing knowledge base as well as personal experiences and shared knowledge with professionals in the software industry. As a result, we designed an innovative artefact to help solve the practical problem of migrating from monolith to microservices.

3. Literature Review

In this section, we summarize migration approaches by other researchers or practitioners. Our steps of identifying, selecting and documenting relevant sources followed the process proposed by Onwuegbuzie et al. (2012) and O'Brian and McGuckin (2016). We first performed an internet search on the Google and Google Scholar platform by using the search string "migrating monolith to microservices." Based on these results, we modified and applied various alternative search strings and controlled for different spellings and synonyms. Additional test searches were then performed in the EBSCO, WISO, De Gruyter and Springer repositories. As a result of our preliminary searches, we identified a literature review by Silva Filho and Figueiredo Carneiro. Their search was conducted on May 4, 2018, covered a period of 10 years, and yielded 95 studies, of which only 12 contributions addressed monolith to microservice migration strategies. Of those, five articles focused on extraction techniques: Chen et al. (2017), Escobar et al. (2016), Baresi et al. (2017), Jamshidi et al. (2017), and Aiello et al. (2016).

We decided to build on these findings, limited our search to the period May 2018 through August 2020, again refined the search strategy, and performed our final search in the repositories listed above on September 1st 2020. Furthermore, we searched Scopus and Web of Science as well as ResearchGate to assure we did not miss relevant contributions. 48 sources were identified, of which 11 were categorized as highly relevant. Two sources provided a comprehensive migration approach: Maisto et al. (2020) and Megargel et al. (2020). Another nine sources addressed various extraction methods: Taibi and Systä (2020), Li et al. (2019), Abdullah et al. (2019), Ma et al. (2019), Nunes et al. (2019), Bucchiarone et al. (2020), Alwis et al. (2019a), Pigazzini et al. (2019), and Henry and Ridene (2020).

In total, the literature review performed by Silva Filho and Figueiredo Carneiro and our own search yielded 16 highly relevant sources as summarized in table 1 and 4:

Tab. 1.: General Migration Approaches (own illustration)

General Migration Approaches	Authors
Three phases	Maisto et al. (2020)
Six phases	Megargel et al. (2020)

General migration approaches provide a comprehensive, sequential phase model, which describes each stage of the migration from monolith to microservice.

Maisto et al. (2020) proposes a three-step model, starting with the decomposition phase, in which the application´s source code is analyzed and candidates for microservices are identified. Next, in the microservice production and ranking phase, designers receive a set of guidelines and a priority index. Communication stubs provide designers with development proposals for microservices, re-engineering the monolith´s functionalities. After all microservice candidates are defined, the new architecture is established, existing code is modernized, and new microservices are generated. Finally, the new microservices architecture is evaluated and services are deployed to the cloud [34].

Megargel et al. (2020) take a broader perspective, presenting a six-phase model, consisting of 14 steps as summarized below:

Tab. 2. Six migration phases proposed by Megargel et al. (2020)

Phase 1: Decoupling Monolith 1. Add Service Layer / Façade 2. Add Service Mediation Layer	**Phase 4: Deploy Microservices to Cloud** 9. Implement API Gateway 10. Deploy Microservices
Phase 2: Develop Local Microservices 3. Identify Microservices 4. Develop Interface Definitions 5. Develop Microservices	**Phase 5: Implement Microservices on Cloud** 11. Migrate Data to Cloud 12. Parallel Run in Cloud 13. Swing Channels to API Gateway
Phase 3: Implement Local Microservices 6. Migrate Data 7. Testing / Parallel Run 8. Swing Channels to Microservices	**Phase 6: Decommission Monolith** 14. Unplug Monolith

First, the monolith is decoupled by introducing a layer between the frontend user interface layer and the backend business logic layer. The service mediation layer is added to provide run-time control over the channel-to-service mapping. With this capability, it is possible to swing the entire channel to consume microservices. Next, local microservices are programmed using standard development and testing tools. In addition, design time governance tools to manage the microservices design lifecycle are recommended. As a result, microservices reflect the same business logic and data scheme as the original function within the monolith. Implementation of local microservices starts with data migration, such that the channel invokes monolith and microservices, allowing both running in parallel. Reconciling data generated by both systems is used for testing each microservice before swinging the channel to exclusively invoke the microservice. This loop is repeated for every single microservice or a batch of services until all are implemented locally. The “swing” to microservices can be effected without changing a single line of code, because both use exactly the same interface [35]. The remaining phases relate to cloud deployment and implementation as well as the eventual decommissioning of the monolith.

Both migration models are summarized below and condensed into five migration phases:

Tab. 3. General Migration Approaches (own illustration)

Maisto et al. (2020)	**Megargel et al. (2020)**	**Migration Phase**
	Decouple monolith	**Decouple**
Decomposition Identify microservice candidates via decomposition	**Develop local microservices**	**Extract**
Microservice production and ranking Define architecture Create documentation describing all functionalities Ranking / prioritizing Re-engineering	Develop interface definitions Develop microservices	**Develop**
Cloud deployment Evaluation of microservice architecture	**Implement local microservices** Migrate data Testing / parallel run Swing channels to microservice **Deploy microservices to cloud** **Implement microservices on cloud**	**Test / deploy**
	Decommission monolith	**Decommission**

Maisto et al. (2020) focus on the extraction and development phase. They recommend extracting process information from code by identifying classes and methods that make up the project, assuming that for each class a corresponding microservice can exist. In contrast, Megargel et al. (2020) place emphasis on the testing and deployment phase, even though they state that identifying microservice candidates in the extraction phase "... is both the most tedious step and the most critical step in the entire migration process." [35]. Irrespective of the different emphasis both contributions place on various steps of the process, we derived five stages, which describe the steps to follow in a migration project: decouple, extract, develop, test/deploy and decommission.

The remaining highly relevant sources discuss **extraction methods**. By analyzing existing code, granular information on business logic can be extracted in order to generate microservices via code reverse engineering, providing the same functionality as the monolith:

Tab. 4. Extraction Methods (own illustration)

Extraction Method	**Author**
Data Flow Driven	
• via business process mining	Taibi and Systä (2020)
• via dataflow-driven semi-automatic decomposition approach	Li et al. (2019)
• via detailed dataflow diagram	Chen et al. (2017)
• a black-box approach that uses the application access logs and unsupervised machine-learning algorithm	Abdullah et al. (2019)
Graph Dependencies	
• via GSMART	Ma et al. (2019)
• via Java-call-graph	Nunes et al. (2019)
• via visualising dependencies between components or layers.	Escobar et al. (2016)
Sematic Similarities	
• via semantic similarity of functionalities	Baresi et al. (2017)
• via a text-based meta-modelling framework	Bucchiarone et al. (2020)
Pattern Driven	
• pattern-driven Architecture Migration (V-PAM)	Jamshidi et al. (2017)
• structure the architecture by software functions and their interactions	Alwis et al. (2019a)
Tool Supported	
• Arcan	Pigazzini et al. (2019)
• Service Cutter	Aiello et al. (2016)
• Blue Age Analyzer	Henry and Ridene (2020)

Data flow-driven extraction methods build on transaction data generated by the monolith. Data analytics combined with pattern recognition help to identify service candidates. Several contributions are based on this principal idea: a data flow-driven semi-automatic decomposition method using fine-grained Data Flow Diagrams (DFD) to cluster service candidates [31] [12]; a black-box approach that uses application access logs and unsupervised machine-learning algorithms to map URL partitions with similar performance and resource requirements [1]; and a method using business process mining to identify service candidates [43].

Graph dependencies methods provide a visual representation of the dependencies between elements of the code to identify service candidates. Java-call-graphs are generated by collecting data using a static code analyzer, and then assessing communication rates in order to identify classes that have a high coupling. The architect generates a dendrogram using hierarchical clustering algorithms. The generated information is then visualized to assist the architect in informed experimentation until a fair balance is achieved between the microservices service covered and the communication rate [37]. GSMART (Graph-based and Scenario-driven Microservice Analysis, Retrieval and Testing) generates a different type of visual representation, while Service Dependency Graphs (SDG) visualize relationships and accelerate the development of new microservices [33]. Alternatively, dependencies between components or layers of applications (business and data layer) can be visualized [15].

Semantic similarity-based extraction methods use algorithms trained to detect linguistic patterns in order to identify relationships between sections and lines of code, using a text-based metamodeling framework [11] or a reference vocabulary, to identify potential candidates as groups of cohesive operations and associated resources. This should help in decomposing the monolith and also generate insights about granularity and cohesiveness of obtained microservices [7].

Pattern-driven methods use empirical data generated by observations from prior migration projects. Cloud architecture migration patterns, migration process frameworks, and variability models are complemented by secondary source analysis and derived to compose a migration plan (Jamshidi et al. 2017). Alternatively, the architectual structure of the monolith can be decomposed using queuing theory and business object relationship analysis [3].

Finally, various *software tools* are available to support architects in decomposing mostly Java-based applications. These tools use a variety of methods discussed above: "Arcan" analyses the monolith´s static structure, generates dependency graphs, and uses algorithms to detect and extract specific topics from the code without human supervision. These topics could help identify service candidates. Algorithms, such as Latent Dirichlet Allocation (LDA), Seeded Latent Dirichlet Allocation (SLDA), and a semi-supervised variant of the original LDA algorithm, extract topics [40]. "Service Cutter" extracts coupling information and engineering artefacts, such as domain models and use cases, to find and score densely connected clusters. The resulting candidate service cuts promise to reduce coupling between, and promote high cohesion within, services [2]. "Blue Age Analyzer" uses queuing theory and business object relationships to identify candidates. It automatically identifies all entry points into the system and organizes the dependencies into concentric rings. Microservice candidates appear as local trees starting from the outside [19].

In summary, except for the comprehensive three-phase approach presented by Maisto and the six-phase approach put forward by Megargel, all other contributions focus on extraction methods rather than a holistic view of the migration process.

4. Evaluation of Migration Approaches and Extraction Methods

Next, we may address the second research question on the benefits and challenges associated with each of the approaches summarized in the previous chapter:

General migration approaches as put forward by Maisto et al. (2020) and Megargel et al. (2020) provide a comprehensive model on how to approach migration projects. Their phased model provides insight on how to perform the steps associated with each phase. However, the extraction phase is the most challenging and critical, but the authors only offer limited guidance on how to identify and implement microservices. The value of both approaches predominantly lies in the comprehensive framework and the orientation it provides for migration projects. However, the critical extraction phase would need to be complemented by extraction methods summarized above, or by leveraging RPA technology, such as a novel RPA use case we present in section 5.

Most research on migration projects focuses on **extraction methods**. *Data-driven* extraction methods are business-focused and provide quick information on processes and variants as they are performed in the organization. However, this black box approach does not detect hidden business logic and the quality of the analysis greatly depends on the input data. In addition, the architectural structure of the code is disregarded and there is no reuse of code. *Graph dependencies* depict the architectural structure and provide transparency on input-output relationships; however, this requires a lot of manual input. *Semantic-based* extraction is highly automated; however, results also need substantial manual rework, especially if coding and naming conventions of the legacy system are inconsistent. *Pattern-driven* approaches are solely based on professional experience in comparable migration projects and, therefore, are highly subjective in nature. There is no transparency on how architectural, business, and process perspectives guide the migration effort. Finally, dedicated *extraction tools* often feature a combination of different methods and work best with Java, though dedicated tools only provide limited assistance to architects.

In essence, the idea of extracting business logic from the legacy code base is appealing, since recreating and documenting business processes, even when equipped with dedicated extraction tools, is a tedious task developers are ill equipped to perform. Since business logic needs to be documented on the most granular (click) level, only process owners and dedicated staff would be able to provide this input. Therefore, code reverse engineering potentially speeds up the project and reduces the need for developers to interview and solicit input from domain and process experts. In addition, the potential reuse of existing code is helpful and works best if the monolith is comparably small and of limited complexity. Taking a look "under the hood" of the monolith may provide a good starting point for these tasks and reduce the need of having to revert to business personnel to mine for business logic and recreate detailed process documentation.

Even though there is a wide range of extraction methods, all but the data driven approach require access to the source code of a legacy system. This is not always possible, as many legacy systems were made from off-the-shelves software packages, bought many years ago. Vendors no longer support these applications. Some legacy ERP or CRM systems are run as terminal solutions. Therefore, the extraction methods discussed above typically are limited to proprietary software, where developers have access to code.

In addition, legacy systems are usually outdated, and their features and functionality do not reflect current and anticipated business requirements. Therefore, re-engineering, i.e., recreating outdated business logic and software functionality by using a new architectural style, does not address the full potential of a major migration effort.

Ultimately, code reverse engineering is building a new structure on an existing, presumably outdated foundation. There is the potential to save time and resources if existing elements of code are reused; however, on the flip side are limitations in terms of outdated structures hampering progress and innovation in designing the new microservice-based application.

5. Integrating RPA into the Migration

The above evaluation of migration approaches has revealed substantial challenges; therefore, we may now address our last research question: *What alternative solution or modifications may help reduce uncertainty and risk of failure in migration projects?* We propose a novel approach, in which RPA bots mimic microservices. This requires certain modifications to standard RPA design principles (section 5.1). We propose a new systematic framework on how to integrate RPA into the standard migration process (section 5.2). The following is our evaluation as to whether the integrated migration approach helps reduce uncertainty and risk of failure (section 5.3).

5.1. Modifications to traditional RPA

For a seamless integration of RPA bots into the migration process and the ultimate replacement of the bot by a corresponding microservice, we propose modifications to the RPA process:

Tab. 5. Standard RPA approach and required modifications for bots to mimic microservices (own illustration)

RPA Approach	Modifications
Identification Process identification based on business and technical criteria	
Description Detailed process documentation	Split processes and define interfaces
Development Iterative implementation, testing, technical and commercial acceptance	Apply microservice principles to bot architecture
Deployment Roll out bot to vendor-specific RPA platform, test against productive systems	Control via RESTful API, parallel testing
Decommission	Swing bot to microservice, decommission bot

The first step in RPA projects is **process identification**, which is supported by technical tools such as Process Mining, Task Mining, and Process Discovery (Reinkemeyer 2020, p. 185). If RPA technology facilitates the migration to microservices, process identification does not require any modification.

Next, processes selected for automation are analyzed and documented via vendor tools such as AM Muse [5] or UI Task Capture [45]. The resulting **process descriptions** and additional information on application programming interfaces form the basis for creating the bots. Discussions between application managers and IT may result in amendments to the development roadmap. In the case of synergies or redundancies, tasks or parts of a process can be automated by using existing interfaces or simple extensions to the interface. However, since monoliths usually do not provide technical interfaces, bots can simulate the interface and thus allow for quick automation. Furthermore, the process documentation as well as transaction data generated by the bot is useful for developing microservices.

Bot **development** is mission critical. Since the bot will mimic a microservice, and eventually be replaced by one, adoption of microservice architectural principles to bot creation is essential. Important characteristics of this architectural design style are: capsuled, deployable sequence of application logic, sharing as little as possible, independence from other services,

and lightweight communication mechanisms [16] [17] [24]. Key microservice design principles compare with RPA as follows:

- Microservice architecture is based on a *share-nothing philosophy* [18]. This is a challenge for RPA since bots access an application such as a human user via the presentation layer. Therefore, the bot always requires a predetermined state of an application as a start and end. For example, each bot first must log on to an application before it can execute tasks within the application. Due to the share-nothing principle, the log-on routine cannot be shared between different bots.
- Microservices should *contain only limited application logic* to allow for independent deployment. It should be small enough that a team of developers can build and maintain it. A single person should be able to understand the full context of the microservice. Ideally, it should have less than a few hundred lines of code [35].
- The *principle of independence* calls for a system of services which consists of independent microservices (slices) that are mostly independent to each other [22]. This should be the case if services are individually deployable, run as self-contained units, and encompass an operating system along with the necessary runtimes, frameworks, libraries, and code [10].
- Microservices need to *tolerate the unavailability of the services* they access [10] and minimize fallout from unexpected constellations [16].

These design principles are addressed in the bot development phase as follows:

- Processes are split into small, distinct tasks. Some tasks are performed multiple times within one process. These standard tasks are assigned to microbots, small reusable bots designed to perform a single dedicated task only. Microbots have a defined starting and ending state, as well as defined input and output parameters. Since bots invoke microbots, the underlying business logic is implemented only once.
- For bots to be independently deployable and resilient against unavailability of services, they need to have a predetermined starting and ending point with clearly defined properties. In a Windows environment, this is the login window, because this is where the operating system will return after the computer reboots in the case of unforeseen events.
- Bots should execute a transaction one case at a time and only start the next case if the last run is successful. Batch processing is incompatible with microservice architectural principles. For example, a purchase order confirmation issued by a bot must be completed by one business process instance before the bot is restarted and the next purchase order is manipulated [24].
- Bots perform tasks in a sequential fashion. If an application is not responding, this creates a roadblock for process execution. Rather than aborting the sequence, the bot should pause and restart automatically after a pre-set amount of time has lapsed. This results in higher rate of successfully completed processes.

After successful development, bots are **deployed**. Microservices communicate through lightweight mechanisms, often a RESTful API [17]. The same mechanism should control the bot. Many RPA vendors offer this feature in their product. It is important to define input and output parameters according to RESTful API standards, since both bot and microservices use this interface. According to the open-close principle [23], the bots and subsequent microservices should be open to extension, but closed to modification. This implies that adding new functionality to software should not affect existing code. Stability on the interface level safeguards the eventual migration from bot to microservice.

In classical RPA, the **decommissioning** of bots is rarely discussed, as they are used for as long as they function properly and serve a purpose. The value of a bot predominantly lies in its encapsulated process knowhow, which may become obsolete if underlying business models or processes change. Therefore, decision-making on the decommissioning of bots is mostly discussed in the context of creating superior automation solutions, which then render the bots superfluous. In this context, bots are decommissioned once the microservice fully functions and parallel tests are completed.

5.2. RPA in the Migration Approach

We established that bots can mimic microservices if certain design principles are observed. Next, we will demonstrate how to integrate RPA technology into the migration process. For simplicity reasons, the flow chart does not depict process loops necessary to address the great number of different processes in a migration project:

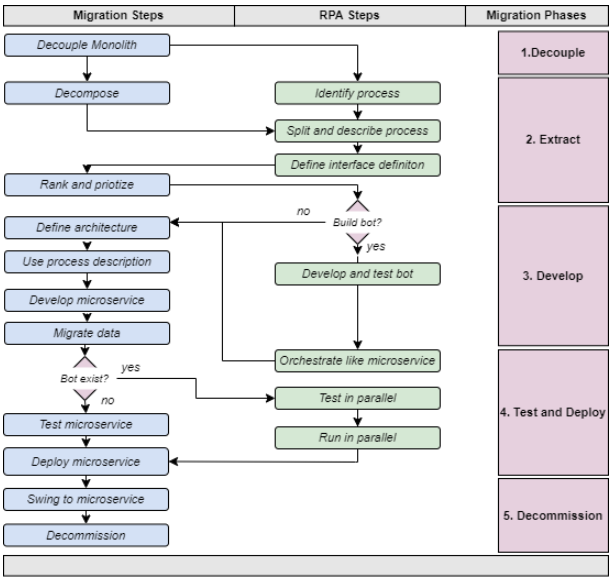

Fig. 1. RPA integrated into the Migration Process (own illustration)

First, the monolith is decoupled by introducing a façade layer between the user interface and the business logic layer [35]. A service mediation layer provides run-time control. The refactoring towards microservices should be done in small parts [28].
Next, monoliths are decomposed by identifying microservice candidates. If the code base is accessible, developers can use standard extraction methods as outlined in Section 3. Alternatively, RPA technology can extract business logic via Process Mining, Task Mining, and Process Discovery, described and documented via RPA vendor tools, and complemented by additional information on application programming interfaces. As a result, developers create a ranking of microservice candidates based on technical (e.g.., outdated code ratio) and commercial criteria (e.g., cost benefit ratio).

Starting with top-ranked microservice candidates, a decision as to whether or not a bot should be created is made. When a service does not perform updates on the database or call other services [19], or when it performs complex calculations, it is recommended to develop microservices without creating bots first. Some examples are rewards services from an online shop which exclusively uses customer information or authentication services [19]. For all other service candidates, a business case determines whether to build the bot.

If yes, the existing process documentation and interface definitions are the basis for the bot. Since the bot shall mimic a microservice, and eventually replace it, adoption of microservice architectural principles as outlined above is essential. Running the bot generates transaction data, which helps identify process variants that were potentially overlooked in the description phase. RPA developers can develop and test bots in parallel to software architects focusing on reengineering microservices. Therefore, having two teams with different skill sets working towards the same goal should expedite the migration process.

If no, the microservice architecture mirrors information derived by extraction methods as discussed earlier. In addition to business logic extracted via decomposing existing code, the process documentation represents valuable information to software architects and is helpful for reengineering and developing microservices.

Good test coverage is required due to the risk of new bugs ending up in the existing features [34]; therefore, the required data of the monolith must be migrated into the new data structure of the microservice, unit tested, and deployed for each service.

If there is no bot to test, developers need to write unit tests. After successful tests, the microservice runs parallel to the monolith until no errors occur. If there is a bot, it is orchestrated via microservice-compatible mechanisms. Both the bot and microservices run in parallel and perform the same request with identical input data. If the output is identical, the test was successful.

The monolith is decommissioned if test results confirm the full set of features of the monolith is covered by microservices [35].

5.3. Evaluation and risk mitigation

Leveraging RPA in a migration context is helpful because it provides an alternative solution in case there is no access to the code base and most extraction methods do not work. In addition, in many cases, it is easier and quicker for software architects to draw on business resources to extract business logic, via process identification and description, to obtain detailed guidance for developing microservices. Especially in large and complex monoliths it is tedious and difficult for software architects to extract business logic from code [28].

Software architects often are in short supply; therefore, leveraging business resources for providing input to the developer saves time by easing resource bottlenecks. It is quick and comparatively simple to document processes and create bots via RPA software even for non-IT staff lacking coding skills. Furthermore, not being bound to a potentially outdated structure of the legacy application provides developers with a higher degree of freedom.

In the testing phase, bots are also beneficial, since automatic testing by running bots and microservices in parallel is a safe and fast way to validate that the software does what it needs to do [28].

After completion of the testing, bots are decommissioned. This alleviates potential maintenance problems associated with RPA. Since bots are bound to the presentation layer of legacy applications, any change to the presentation layer (for instance, due to updating to a new release) requires bot modification. This is more likely to happen the longer a bot is in use. In the integrated migration approach, we propose to use bots only temporarily until microservices take over. Therefore, bot maintenance should not turn out to represent technical debt.

On the other hand, RPA technology is non-invasive and mandates no changes to existing application systems [6]. Therefore, bots build on existing, often outdated applications, which may turn out to be a limiting factor, potentially reducing flexibility.

In summary, based on the above evaluation, applying RPA technology in migration projects is advantageous and conducive to mitigating migration risks, especially in the extraction and testing phase.

6. Conclusion

Almost 90% of business leaders in the U.S. and U.K. expect IT and digital technologies to make an increasing strategic contribution to operations in the coming decade [20]. This technology-induced change involves improvements to existing processes, exploration of digital innovation, and potentially a transformation of the business model [9]. Managing change successfully provides organizations with opportunities to create value. IT contributes to value creation through automation effects (productivity improvements, labor savings and cost reductions), informational effects (collected, stored, processed, and disseminated information improves decision quality) and transformational effects (facilitate and support process

innovation and transformation) [27]. Therefore, IT strategies need to ensure efficient management of IT infrastructure and application systems. In particular, keeping IT systems aligned with business models and processes underscores the need for IT to become more agile and business-centric [20].

It Is not surprising that leading tech companies like Amazon or Netflix have transitioned from creating and running monolithic applications to applying microservice design principles to their software architecture because of flexibility, scalability, complexity, agility, and maintainability considerations. Still, many organizations operate and maintain legacy monolith systems despite the compelling advantages of microservices, and migrations to microservices are still rare [35]. Helping organizations to keep IT systems up to date and aligned with changing business models by facilitating a smooth migration to microservices offers substantial potential to create value. We, therefore, researched state-of-the-art migration approaches and discovered weaknesses, especially in the extraction phase.

In search of alternative solutions, we realized that instead of extracting business logic from code, RPA draws on business resources to understand business processes and requirements. This represents a novel use case for RPA technology in a migration context. We proposed to integrate RPA into the migration process and determined design principles for bots to mimic microservices. Once microservices are parallel tested and operational, bots are decommissioned, and microservices replace the monolith. Based on logical reasoning, we showed that our integrated approach is conducive to reducing migration risk. This should provide IT management and software engineers with guidance on how to manage migration projects.

The key limitation of this paper is a lack of empirical evidence, as the feasibility of the integrated migration approach is yet to be tested in practice. With this paper, we promote the idea to leverage RPA technology in the migration to microservices. However, empirical findings from real-life applications of this approach are as yet outstanding.

AUTHORS

Burkhard Bamberger* – ISM International School of Management GmbH, 44227 Dortmund, Germany, email: Burkhard.Bamberger@ism.de.

Bastian Körber – ISM International School of Management GmbH, 44227 Dortmund, Germany, email:bastian.koerber@outlook.de.

REFERENCES

[1] M. Abdullah, W. Iqbal, A. Erradi, "Unsupervised learning approach for web application auto-decomposition into microservices", *Journal of Systems and Software,* 151, 2019, pp. 243–257. DOI: 10.1016/j.jss.2019.02.031

[2] M. Aiello, E.B. Johnsen, S. Dustdar, I. Georgievski, (Eds.), Service-Oriented and Cloud Computing. Cham: Springer International Publishing (Lecture Notes in Computer Science), 2016.

[3] A. A. C. de Alwis, A. Barros, C. Fidge, A. Polyvyanyy, "Availability and Scalability Optimized Microservice Discovery from Enterprise Systems", in H. Panetto, C. Debruyne, M. Hepp, D. Lewis, C. A. Ardagna, R. Meersman (Eds.), On the Move to Meaningful Internet Systems: OTM 2019 Conferences, vol. 11877, Cham: Springer International Publishing (Lecture Notes in Computer Science), 2019, pp. 496–514.

[4] A. A. C. de Alwis, A. Barros, C. Fidge, A. Polyvyanyy, "Business Object Centric Microservices Patterns", in H. Panetto, C. Debruyne, M. Hepp, D. Lewis, C. A. Ardagna, R. Meersman (Eds.), On the Move to Meaningful Internet Systems: OTM 2019 Conferences, vol. 11877. Cham: Springer International Publishing (Lecture Notes in Computer Science), 2019, pp. 476–495.

[5] Another Monday Intelligent Process Automation GmbH, AM Muse, Version 1.35: Another Monday Intelligent Process Automation GmbH, 2020. https://www.anothermonday.com/products/am-muse

[6] G. Auth, C. Czarnecki, F. Bensberg, "Impact of Robotic Process Automation on Enterprise Architectures", in *Lecture Notes in Informatics (LNI)*, 2019. DOI: 10.18420/INF2019_WS05

[7] L. Baresi, M. Garriga, A. de Renzis, "Microservices Identification Through Interface Analysis", in F. de Paoli, S. Schulte, E. B. Johnsen (Eds.): Service-Oriented and Cloud Computing, vol. 10465. Cham: Springer International Publishing (Lecture Notes in Computer Science), 2017, pp. 19–33.

[8] L. Bass, I. M. Weber, L. Zhu, DevOps, A software architect's perspective, New York: Addison-Wesley Professional (The SEI series in software engineering), 2015.

[9] S. Berghaus, A. Back, "Stages in Digital Business Transformation: Results of an Empirical Maturity Study", *MCIS 2016 Proceedings, 22,* 2016. https://aisel.aisnet.org/mcis2016/22, checked on 2/19/2020

[10] F. Boyer, X. Etchevers, N. de Palma, X. Tao, "Architecture-Based Automated Updates of Distributed Microservices", in C. Pahl, M. Vukovic, J. Yin, Q. Yu (Eds.): Service-Oriented Computing, vol. 11236, Cham: Springer International Publishing (Lecture Notes in Computer Science), 2018, pp. 21–36.

[11] A. Bucchiarone, K. Soysal, C. Guidi, "A Model--Driven Approach Towards Automatic Migration

to Microservices", in Jean-Michel Bruel, Manuel Mazzara, Bertrand Meyer (Eds.): Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment, vol. 12055, Cham: Springer International Publishing (Lecture Notes in Computer Science), 2020, pp. 15–36.

[12] R. Chen, S. Li, Z. Li, "From Monolith to Microservices: A Dataflow-Driven Approach", in 2017 24th Asia-Pacific Software Engineering Conference (APSEC), 2017 24th Asia-Pacific Software Engineering Conference (APSEC), Nanjing, 04/12/2017 - 08/12/2017: IEEE, 2017, pp. 466–475.

[13] C. Czarnecki, P. Fettke, "Robotic Process Automation: De Gruyter", 2021. https://doi.org/10.1515/9783110676693

[14] T. H. Davenport, The new industrial engineering: information technology and business process redesign" in *Sloan management review,* 1954 (1990).

[15] D. Escobar, D. Cardenas, R. Amarillo, E. Castro, K. Garces, C. Parra, R. Casallas, "Towards the understanding and evolution of monolithic applications as microservices" in 2016 XLII Latin American Computing Conference (CLEI), 2016 XLII Latin American Computing Conference (CLEI), Valparaíso, Chile, 10/10/2016 – 14/10/2016: IEEE, 2016, pp. 1–11.

[16] M. Fowler, J. Lewis, „Microservices: Nur ein weiteres Konzept in der Softwarearchitektur oder mehr", in *Journal of Systems and Software*, 2015, pp. 14–20.

[17] M. Garriga, "Towards a Taxonomy of Microservices Architectures", in Software Engineering and Formal Methods, 2018, pp. 203–218.

[18] J. Ghofrani, D. Lübke, "Challenges of Microservices Architecture: A Survey on the State of the Practice", 2018.

[19] A. Henry, Y. Ridene, "Migrating to Microservices", in A. Bucchiarone, N. Dragoni, S. Dustdar (Eds.), Microservices. Science and engineering, Cham: Springer International Publishing, 2020, pp. 45–72.

[20] T. Hess, C. Matt, A. Benlian, F. Wiesboeck, "Options for formulating a digital transformation strategy", in *MIS Quarterly Executive* (15), 2016, pp. 123–139.

[21] A. R. Hevner, "A Three Cycle View of Design Science Research", Scandinavian Journal of Information Systems, Volume 19, Issue 2, 2007, pp. 87-92.

[22] M. Hilbrich, "In Microservices We Trust – Do Microservices Solve Resilience Challenges?", Tagungsband des FB-SYS Herbsttreffens 2019. Bonn: Gesellschaft für Informatik e.V., 2019. DOI: 10.18420

[23] A. Hochrein, "Anatomy of a Microservice", in Akos Hochrein (Ed.): Designing microservices with Django, An overview of tools and practices, 1st edition, [S.l.]: Apress, 2019, pp. 49–68.

[24] A. Hochrein, "From Monolith to Microservice", in Akos Hochrein (Ed.): Designing microservices with Django, An overview of tools and practices, 1st edition, [S.l.]: Apress, 2019, pp. 111–137.

[25] J. Holmström, M. Ketokivi Mikko, A.-P. Hameri, "Bridging Practice and Theory: A Design Science Approach", in Decision Sciences Volume 40, Issue 1, 2009, pp. 65-87.

[26] P. Jamshidi, C. Pahl, N. C. Mendonça, "Pattern--based multi-cloud architecture migration", in *Software Practice and Experience* 47 (9), 2017, pp. 1159–1184. DOI: 10.1002/spe.2442

[27] J. Mooney, V. Gurbaxani, K. L. Kraemer, "A Process Oriented Framework for Assessing the Business Value of Information Technology", *Forthcoming in the Proceedings of the Sixteenth Annual International Conference on Information Systems*, 2001.

[28] M. Kalske, N. Mäkitalo, T. Mikkonen, "Challenges When Moving from Monolith to Microservice Architecture", in I. Garrigós, M. Wimmer (Eds.): Current Trends in Web Engineering, Cham: Springer International Publishing, 2018, pp. 32–47.

[29] R. Khadka, A. Saeidi, S. Jansen, J. Hage, G. P. Haas, "Migrating a large scale legacy application to SOA: Challenges and lessons learned", in 2013 20th Working Conference on Reverse Engineering (WCRE), 2013 20th Working Conference on Reverse Engineering (WCRE), Koblenz, Germany, 14/10/2013 - 17/10/2013: IEEE, 2013, pp. 425–432.

[30] M. C. Lacity, L. P. Willcocks, "What Knowledge Workers Stand to Gain from Automation", Harvard Business Review, 2015. https://hbr.org/2015/06/what-knowledge-workers-stand-to-gain-from-automation, last checked 23.08.2021

[31] S. Li, H. Zhang, Z. Jia, Z. Li, C. Zhang, J. Li et al., "A dataflow-driven approach to identifying microservices from monolithic applications", *Journal of Systems and Software* 157, 2019, p. 110380. DOI: 10.1016/j.jss.2019.07.008

[32] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly, S. Pallickara, "Serverless Computing: An Investigation of Factors Influencing Microservice Performan-

ce", in 2018 IEEE International Conference on Cloud Engineering (IC2E), 2018 IEEE International Conference on Cloud Engineering (IC2E), Orlando, FL, 17/04/2018 - 20/04/2018: IEEE, 2018, pp. 159–169.

[33] S.-P. Ma, C.-Y. Fan, Y. Chuang, I.-H. Liu, C.-W. Lan, "Graph-based and scenario-driven microservice analysis, retrieval, and testing", *Future Generation Computer Systems,* 100, 2019, pp. 724–735. DOI: 10.1016/j.future.2019.05.048

[34] S. A. Maisto, B. Di Martino, S. Nacchia, "From Monolith to Cloud Architecture Using Semi-automated Microservices Modernization", in L. Barolli, P. Hellinckx, J. Natwichai (Eds.): Advances on P2P, Parallel, Grid, Cloud and Internet Computing, vol. 96, Cham: Springer International Publishing (Lecture Notes in Networks and Systems), 2020, pp. 638–647.

[35] A. Megargel, V. Shankararaman, D. K. Walker, "Migrating from Monoliths to Cloud-Based Microservices: A Banking Industry Example", in M. Ramachandran, Z. Mahmood (Eds.), Software Engineering in the Era of Cloud Computing, Cham: Springer International Publishing (Computer Communications and Networks), 2020, pp. 85–108.

[36] S. Newman, "Building microservices. Designing fine-grained systems", Sebastopol, CA: O'Reilly Media, 2015.

[37] L. Nunes, N. Santos, A. R. Silva, "From a Monolith to a Microservices Architecture: An Approach Based on Transactional Contexts" in T. Bures, L. Duchien, P. Inverardi (Eds.): Software Architecture, vol. 11681, Cham: Springer International Publishing (Lecture Notes in Computer Science), 2019, pp. 37–52.

[38] A. M. O'Brien, C. Mc Guckin, The Systematic Literature Review Method: Trials and Tribulations of Electronic Database Searching at Doctoral Level, 2016.

[39] A. J. Onwuegbuzie, N. L. Leech, K. M. T. Collins, "Qualitative Analysis Techniques for the Review of the Literature", The Qualitative Report, 17, 2012, pp. 1–28.

[40] I. Pigazzini, F. A. Fontana, A. Maggioni, "Tool Support for the Migration to Microservice Architecture: An Industrial Case Study", in T. Bures, L. Duchien, P. Inverardi (Eds.): Software Architecture, vol. 11681, Cham: Springer International Publishing (Lecture Notes in Computer Science), 2019, pp. 247–263.

[41] H. C. da Silva Filho, G. de Figueiredo Carneiro, "Strategies Reported in the Literature to Migrate to Microservices Based Architecture", in Shahram Latifi (Ed.): 16th International Conference on Information Technology-New Generations (ITNG 2019), vol. 800, Cham: Springer International Publishing (Advances in Intelligent Systems and Computing), 2019, pp. 575–580.

[42] H. A. Simon, "The Sciences of the Artificial, Third Edition", 2019.

[43] D. Taibi, K. Systä, "A Decomposition and Metric-Based Evaluation Framework for Microservices", in D. Ferguson, V. Méndez Muñoz, C. Pahl, M. Helfert (Eds.): Cloud Computing and Services Science, vol. 1218, Cham: Springer International Publishing (Communications in Computer and Information Science), 2020, pp. 133–149.

[44] J. Thones, "Microservices", In *IEEE Softw.,* 32(1), 2015, p. 116. DOI: 10.1109/MS.2015.11

[45] UiPath (2020): UI Task Capture: UiPath. Available online at https://www.uipath.com/product/task-capture, checked on 5/25/2020.

[46] W. M. P. van der Aalst, M. Bichler, A. Heinzl, "Robotic Process Automation, Bus Inf Syst Eng, 60(4), 2018, S. 269–272. DOI: 10.1007/s12599-018-0542-4