Effort Estimation of Large-Scale Enterprise Application Mainframe Migration Projects: A Case Study

Sascha Roth
Lucerne University of Applied Sciences and Arts, School of Computer Science and Information Technology, Switzerland

Keywords: Enterprise Applications, Mainframe, Migration, Legacy Systems, Cost Estimation, Cloud Computing.

Abstract: How do you migrate an enterprise application which has decades old legacy code running on an IBM Z-series mainframe? What options do you have and how do you estimate the efforts best? In this paper, we present a model developed during a real-world case study of a migration endeavour of the worldwide warranty system at a major premium automotive. We present a pragmatic approach taken to ballpark migration efforts which allows for similar endeavours to estimate migration efforts in a similar fashion.

1 INTRODUCTION

The mainframe did indeed not go anywhere in the last decade as forecasted by (Barnett, 2005). Yet, it is one of those systems that you better do not touch: Changes are delivered (too) late, developers are either hard to find or hard to onboard, and business never is happy with what they get for the buck.

Enterprise applications running on mainframes usually carry decades of history with them (Bhatnagar et al., 2016). So, it does not surprise that they come with heavy luggage: technical debts. Even the best designs eventually become outdated and become hard(er) to maintain. Due to financial pressure, technical debt often is not addressed adequately with refactoring or rearchitecting measures. Despite software does not age, technical debts hit organizations regularly (Tom et al., 2013). In short: mainframe applications are often legacy than an organization needs to get rid of. A first step toward it is to plan for a migration to another system. In our case, the target was set to the AWS cloud. Effort estimation of software development is a non-trivial issue (Carbonera et al., 2020).

The remainder of the article is structured as follows: we briefly give an outline on mainframes and depict common strategies how to modernize mainframe applications. We then proceed by detailing our case study, set the frame conditions for the case and then outline the solution taken to estimate the efforts and derive a roadmap. In the discussion section, we reflect on the approach and also outline the limitations and risks we see. The paper concludes with an outlook of the migration and further initiatives we can think off to help organizations that are in a similar situation.

2 A (VERY) BRIEF HISTORY OF MAINFRAME MODERNIZATION

Mainframe application modernization has been around for decades. Albeit unwanted, mainframe applications are not that easy to modernize. While (Sun & Li, 2013) proposed an approach to estimate efforts for cloud migrations, they do not account for the mainframe specific challenges, which mostly is outdated design, an inadequate data models and other technical debt accumulated over the decades. However, several methods and approaches are known how to modernize the mainframe, with or without its full replacement.

2.1 Transpilation

Transpilation is one possibility to move from A to B (Schnappinger & Streit, 2021), or let us say: to convert the code base from A to B. That is, this technique takes the sources of one tech stack and transpiles it into the language of the target tech stack. In the authors` example, they performed a conversion at source level from Natural to Java. Our case has a

Roth, S.
Effort Estimation of Large-Scale Enterprise Application Mainframe Migration Projects: A Case Study.
DOI: 10.5220/0013289500003929
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 27th International Conference on Enterprise Information Systems (ICEIS 2025) - Volume 2*, pages 845-850
ISBN: 978-989-758-749-8; ISSN: 2184-4992
Proceedings Copyright © 2025 by SCITEPRESS – Science and Technology Publications, Lda.

different source technology and a 1:1 conversion would not address all issues and challenges the development teams and users of this application face. While this case study is a good example for an application which might not be changed anymore and its roadmap planned for a sundown scenario in the near future, our application is heavily used and, after migration, should strive again like on the first day.

2.2 Screen Scraping

A common technique also discussed during mainframe application modernizations is so-called screen scraping (Dörsam et al., 2009). The main idea is to literally scrape the screen of typically terminal-based mainframe applications and make them accessible via newer technologies, e.g. REST APIs. This technique can be a low-cost alternative if you want to stay on a mainframe. As we will state in the next section, this is not the case in our case study. Further, as the application already has gone through a partial modernization, its logic is no longer implemented in COBOL and the likes. Hence, screen scraping is neither applicable, nor would it help to retire the mainframe.

2.3 The 7-Rs

The 7Rs were initially 5Rs introduced by Gartner (*Migrating Applications to the Cloud*, 2009) and have been extended by Amazon and now include all possibilities the authors can think of (*About the Migration Strategies - AWS Prescriptive Guidance*, 2024). We briefly summarize the 7Rs as they were considered and evaluated in our case study as well.

Rehost: Also known as "lift and shift," this strategy involves moving applications to the cloud without making any changes. It's quick but may not take full advantage of cloud benefits.

Replatform: This involves making a few cloud optimizations to achieve some tangible benefits without changing the core architecture of the applications.

Repurchase: This strategy involves moving to a different product, typically a SaaS platform, which means abandoning the existing application.

Refactor/Re-architect: This involves re-imagining how the application is architected and developed, typically using cloud-native features. It's the most resource-intensive but can offer the most benefits.

Retire: Identify assets that are no longer useful and can be turned off. This can help reduce costs and complexity.

Retain: Keep applications that are critical to the business but are not ready to be migrated. This might be due to complexity, cost, or other factors.

Relocate: Move applications to the cloud without purchasing new hardware, often using virtualization technologies.

3 CASE STUDY

We begin our case study with providing an overview of the context of the application. Our client is in the automotive industry and a major player of luxury cars. The system is one of the aftersales systems to handle warranty and goodwill cases. It features many interfaces and as it is historically grown, many stakeholder consume data from the system, either directly (via interfaces or DB access) or indirectly via reports and file-based exports. [*Permission to disclose further information about the system and its context requested, permission may be provided after review process. Potential artefacts to be shared include: a capability map, process coverage, interfaces to other systems and integrations into the system landscape and datawarehouse and datalake environment*]

3.1 Guardrails

Business applications run in organizations and serve a certain purpose. They are also embedded in IT strategies. This builds guardrails for our decision on why and how to migrate. We briefly outline some key guardrails.

The organization implemented the scaled-agile framework (SAFe) and accordingly the teams work in product increments and sprints.

The mainframe will be retired on December 2026 and prolongation is not possible – we have a fixed deadline.

A predecessor project already investigated that the business capabilities and tech stack are future prove, so entirely replacing the system is out of scope.

3.2 The Tech Stack

The tech stack of our mainframe application is a mixture of mainframe tech and modern frameworks that one would still choose nowadays to design new enterprise applications.

As programming framework, Java Spring is used in combination with a DB2 database. What sounds almost plain vanilla turns out to be a leftover of refactoring measures that took place almost a decade ago. The mainframe parts, namely COBOL were largely removed during that initiative and a state-of-the-art framework was introduced. As for the UI, Angular is used.

An earlier analysis revealed that the application as such is highly customized and fit to the business needs. Given a) the used tech stack is state-of-the-art and b) the coverage of the business use cases is quite specific, our client followed our recommendation to keep the system and did not evaluate if it can be replaced by other solutions (software as a service in particular).

During analysis, one spark that was of our interest has been the DB2. Why would anyone combine a Spring application with DB2? It turns out there are a couple of reasons for that:

High Throughput and Performance: DB2 is designed to handle large volumes of transactions efficiently, making it ideal for high-demand environments. It leverages the robust processing power of mainframes to deliver exceptional performance and throughput.

Scalability: DB2 can scale to meet the needs of growing businesses. It supports large databases and can handle significant increases in data volume without compromising performance.

Reliability and Availability: Mainframes are known for their reliability, and DB2 benefits from this. It offers high availability and disaster recovery capabilities, ensuring that critical applications remain operational even in the event of hardware failures.

Security: DB2 provides advanced security features to protect sensitive data. This includes encryption, access controls, and auditing capabilities, which are essential for industries that handle confidential information.

Integration and Flexibility: DB2 integrates seamlessly with various data sources and applications, providing a flexible environment for data management. It supports both traditional and modern workloads, including cloud and mobile applications.

Advanced Data Management: DB2 includes tools for database management and optimization, helping administrators manage and monitor workloads effectively. This improves productivity and reduces administrative efforts.

3.3 Key Indicators to Consider

For the expert estimation, we used several indicators that ground their number and helped experts to derive their estimate from known facts.

- Code Lines: the total lines of code of all modules
- JPA Queries: total number of queries that conform to the JPA specification and hence can be reused with almost any database out of the box, provided that database is supported by spring boot data
- Native Queries: Number of SQL queries that use native SQL and may include proprietary DB2 functions
- Modules: The number of modules that make up the application, in our case the spring boot applications.
- Interfaces (Consuming, Pull, Polling, or Subscribe): All inbound interfaces that consume data from an endpoint. Most of the time this is an endpoint within the corporate network; but also outside the corporate network.
- Interfaces (Providing, Push, Publish): Interfaces that provide data to other systems.
- Interfaces that definitely need significant refactoring due doe technical debt, e.g. specific flags for use cases have been baked into the interface.
- Tivoli job nets: number of job networks in Tivoli
- Average steps of a job network in Tivoli
- Developers: FTE of developer resources currently dedicated to the application
- Business Analysts: FTE of subject matter experts wrt. the application’s scope
- Number of Tables of all modules
- Average size of a table in million rows
- Largest table in million rows
- Developers with SQL experience in FTE
- Blended rate per team member per day

- Number of database triggers
- Number of views in the database
- Number of history tables in the database
- Number of materialized queries in the database
- Total size of database size in (without LOBs)
- Total history size in GB
- Total size of LOBs
- Outgoing traffic from database per week in GB
- Incoming traffic to database per week in GB
- Total rows in database
- Total send rows per week
- Average row size (without LOBs)
- Number of data objects on application layer

4 EFFORT ESTIMATION MODEL FOR (MAINFRAME) APPLICATION MIGRATIONS

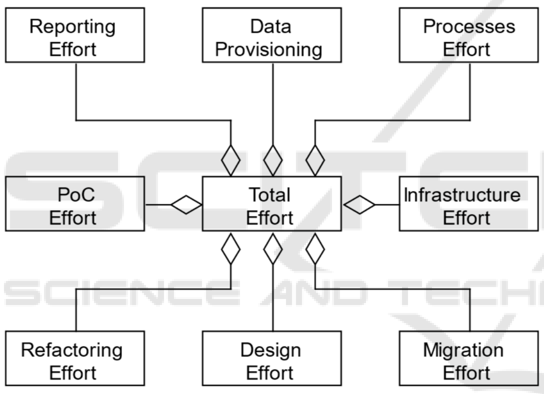

Figure 1: High-Level overview of an effort Estimation Model for Cloud Mainframe Migrations.

In Figure 1 we illustrate the applied estimation model as UML diagram. The total effort is modelled as an aggregation of different efforts. What is depicted as "Total Effort" is our bottom-up effort estimate that is derived from expert estimates. Thereby, each estimate is sufficiently broken down into bits that experts feel comfortable to provide an estimate for.

Figure 2 illustrates how this has been done for the class of "Migration Effort" whereas Figure 3 depicts the break-down for "Refactoring Effort".

In addition to the break-down into smaller bits and pieces and to account for learnings and scale

effects during the migration project, an additional factor for synergies was provided.

The key indicators provided above served to calculate the effort of each group. Various expert interviews helped to sharpen the understanding, whereas the key indicators helped to challenge and ground the experts' opinions.

Our goal was to estimate necessary and sheer unavoidable tasks that need to be performed during the migration.

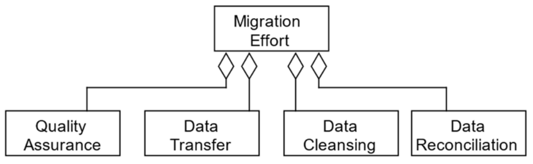

Figure 2. Breakdown of migration efforts.

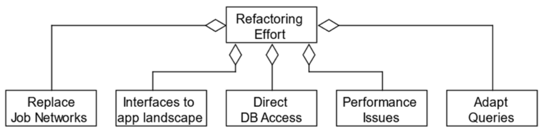

Figure 3: Breakdown of refactoring efforts.

4.1 Agile vs non-Agile Migration

During the project, we also challenged the application of agile methods for this migration project. Although our client already implemented SAFe and performs product increment (PI) plannings, for the migration of a mainframe, this setup seemed inadequate: Agile methods are particularly well suited if outcomes are unclear and/or customer requirements frequently change. This is not the case in migration projects.

4.2 The Target Architecture

Mainly guided by the guardrails and the current architecture of the application, the potential target tech stack reduces itself to not too many options. However, these need to be evaluated during PoCs and one needs to see if timeouts of application logic etc. will still hold in the cloud with potentially higher latencies.

Direct database access will no longer be possible as this (always) was a concerning issue to give business direct access to a database with even the query power to perform an unconditional and largely unrestricted "select *" of many GB- or TB-sized tables.

4.3 Organizational Overhead

Most startups are faster in developing applications. This is mainly due to two reasons: 1) They build green-field 2) They have less stakeholders which results in less communication overhead.

While experts can estimate their own work very well, we argue that they did not consider for organizational overhead, such as alignment meetings or conflicts that arise when discussing data ownership or alternative designs and redesign of the modules (cf. also (Henry & Ridene, 2020)). We chose a three-step approach bringing together the bottom-up perspective and challenging it with a top-down estimate.

Step 1: We validated the bottom-up number with experts as previously outlined.

Step 2: We added 5 dimensions of organizational overhead and estimated percentages of the initial estimate.

Step 3: We reflected the resulting number with top-down estimates of a migration, i.e. is it between a factor 2 to 3 of the annual operations cost.

As a rule of thumb, a top-down estimate for migration efforts can be derived from the annual operations cost. A factor 2 can be assumed as migration cost if the migration is more or less straightforward. On the other hand, a factor 3 can be assumed for more complicated setups like a mainframe migration to the cloud.

In our case study, the bottom-up estimates where indeed in the range of 2-3 times the annual operations cost for the mainframe system.

4 CONCLUSIONS & FURTHER RESEARCH

Mainframe migrations are a true challenge in practice. Grown over decades, organizations need to take them into account in enterprise application landscape planning. Little guidance is given to organizations how to plan and execute their mainframe migrations.

In this paper, we presented a practice proven model for mainframe effort estimation, offering a structured approach to ballpark the resources required for mainframe migration initiatives. While the model is still in its evaluation phase and has not yet sufficiently been empirically evaluated, it lays a solid foundation for future research and practical application.

The proposed model integrates historical system data and project-specific variables estimated by experts, aiming to provide a comprehensive framework for effort estimation. By addressing the complexities and unique characteristics of mainframe projects, this model has the potential to enhance project planning and resource allocation significantly.

Future work will focus on validating the model through additional empirical studies and real-world applications. This will involve collecting and analyzing data from various mainframe projects to assess the model's accuracy and reliability. Additionally, exploring the integration of advanced statistical techniques and machine learning algorithms could further refine the model and improve its predictive capabilities.

As the model undergoes further development and validation, it holds the potential to become a valuable tool for project managers and stakeholders, enabling them to make informed decisions and optimize resource utilization during mainframe migration projects.

Further research could focus on implementing a system that provides various model configurations and shares best practices with the community. Maturity of organizations as well as their industry could play crucial factors in estimating efforts for mainframe migrations. In some cases, one might choose to rebuild a new system after all. Models and systems could help preventing sunk costs and frustrating journeys for organizations.

REFERENCES

About the migration strategies—AWS Prescriptive Guidance. (2024). https://docs.aws.amazon.com/prescriptive-guidance/latest/large-migration-guide/migration-strategies.html

Barnett, G. (2005). *The future of the mainframe*.

Bhatnagar, M., Shekhar, J., & Kumar, S. (2016). One Architecture Fits All – IBM Mainframe. *GRD Journals-Global Research and Development Journal for Engineering, ISSN: 2455-5703*, *1*, 85–91.

Carbonera, C. E., Farias, K., & Bischoff, V. (2020). Software development effort estimation: A systematic mapping study. In *IET Software* (Vol. 14, Issue 4, pp. 328–344).

Dörsam, M., Gründling, S., Langholz, T., Roth, S., & Steinbrecht, A. (2009). Integrating a Legacy Terminal Application into an SOA. *Informatiktage 2009 - Gesellschaft Für Informatik (GI)*, 95.

Henry, A., & Ridene, Y. (2020). Migrating to Microservices. In A. Bucchiarone, N. Dragoni, S. Dustdar, P. Lago, M. Mazzara, V. Rivera, & A. Sadovykh (Eds.), *Microservices: Science and Engineering* (pp. 45–72). Springer International Publishing. https://doi.org/10.1007/978-3-030-31646-4_3

Migrating Applications to the Cloud: Rehost, Refactor, Revise, Rebuild, or Replace? (2009). Gartner. https://www.gartner.com/en/documents/1485116

Schnappinger, M., & Streit, J. (2021). Efficient Platform Migration of a Mainframe Legacy System Using Custom Transpilation. *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 545–554. https://doi.org/10.1109/ICSME52107.2021.00055

Sun, K., & Li, Y. (2013). Effort Estimation in Cloud Migration Process. *2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*, 84–91. https://doi.org/10.1109/SOSE.2013.29

Tom, E., Aurum, A., & Vidgen, R. (2013). An exploration of technical debt. *Journal of Systems and Software*, *86*(6), 1498–1516. https://doi.org/10.1016/j.jss.2012.12.052