Towards a Systematic Refactoring Method to Support Deployability

Daniel Salvador Salas Torres, Humberto Cervantes, Ricardo Marcelín Jiménez
Depto Ciencias Basicas e Ingenieria.
Universidad Autónoma Metropolitana - Iztapalapa
CDMX, México
salvador.st09@gmail.com, hcm@xanum.uam.mx, rmarcelin@izt.uam.mx

***Abstract*—This article presents a systematic refactoring methodology to support deployability, applicable to any type of application, with the goal of improving or incorporating the quality attribute of deployability. We focus on applications that did not consider this attribute during the early stages of design. The methodology consists of four stages: Evaluation, Analysis and Prioritization of activities, Refactoring, and Re-evaluation, which together allow for continuous improvement of the deployment process each time they are applied. To test this methodology, we worked with three different development teams where the initial stages were applied. Additionally, a prototype was built in which all four stages were implemented, successfully proving the methodology's effectiveness and achieving optimal results that improved the deployment process of the prototype.**

Keywords-component; deployability; pipeline; containers; cluster kubernetes;

I. Introduction

The efficient deployment of applications has become a crucial activity for today's organizations. This involves the capacity of implementing and moving the application across various development environments such as the local environment, pre-production testing environments, and finally the production environment, where each of these may have different operating conditions. This process ensures that the application is tested and works correctly before reaching the end users. Efficient deployment requires the removal of manual activities in the deployment process.

To achieve automated deployment, it is essential to incorporate a quality attribute known as deployability.[1] This term refers to the ability to put a system into operation in a specific environment with predictable effort and time [2]. Achieving deployability requires design decisions to be made both in the application and its development and deployment infrastructure. These design decisions should, ideally, be made early on.

However, it is important to highlight that most organizations tend to focus on adding functionality to the applications and not on how to effectively deliver it to end users. The consequence is that, once developed, the organization experiences delays when the system needs to be moved to production. The motivation of this research is to investigate means to improve the support to deployability in applications that were not originally designed to support this quality attribute.

In this research, we propose a systematic methodology for refactoring applications with the main objective of improving deployability by making changes to both the development and deployment infrastructure and the functionality of the application. We will refer to the development and deployment infrastructure simply as infrastructure in the rest of this paper. This methodology is divided into four stages. The first stage is the evaluation, where we examine the current state of the application and its infrastructure in terms of deployability. Next, we move to the analysis stage, which is divided into three sub-stages. The first one involves studying the data obtained in the evaluation process. The second one involves deciding, based on the results obtained, if the application or its infrastructure needs any intervention to improve deployability. Finally, we enter the last phase, which is prioritization and proposing actions to be taken if an intervention is necessary. Subsequently, we proceed with a stage called refactoring, where changes will be implemented in the application or its infrastructure according to the proposed activities. Finally, we enter a re-evaluation stage, where we subject the application and its infrastructure to another analysis to compare the before and after of the intervention and to evaluate if it is necessary to continue refactoring or if deployability is now successfully supported.

This article is divided into seven sections; introduction. theoretical framework, related work, proposal, results and analysis, conclusions and future work.

II. Theoretical Framework

In this section, we explore the key concepts and essential tools that are needed to support deployability. When properly integrated, these tools and practices not only streamline deployment but also improve the stability and quality of the final product.

II.a Deployment Environments

Deployment is defined as the process of moving an application from a development environment to a production environment, or to an environment where end users can effectively access and use the application[2]. This process goes through three different types of environments: the local environment, the pre-production environment(s), and the production environment, as shown in Fig 1.

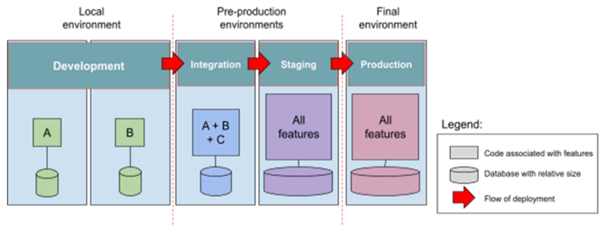


Fig 1. Illustrates the transition flow through the deployment environments in the software development process [2]

In the local environment, developers write the code and perform unit tests on each component of the application locally on their machines. From the beginning, the code is managed through a version control system, and tools like Git or Subversion [3][4] are often used, allowing for effective change control and better management of the application's components.

In pre-production environments, several essential activities are performed. In the integration environment, all the components developed locally are brought together to create an executable version of the application. Continuous integration practices are implemented here, constantly merging and evaluating code updates, resulting in a deployable system. The integration environment is the first line of validation and testing, essential for later stages. Another important environment is the testing environment, where the application undergoes exhaustive evaluations in an environment similar to production, also using continuous integration. Although these are the main environments, there may be others depending on the specific needs of the application in development.

In the production environment, after successfully passing pre-production tests, the system is deployed for end users. This transition is efficiently carried out through continuous deployment practices, facilitating a quick and smooth implementation of updates. Additionally, rollback mechanisms are in place to revert the application to a previous state in case of update failures.

II.b Continuous Integration and Deployment (CI/CD) Practices.

Continuous Integration (CI) automates the integration and testing of application components regularly to identify and address issues early in development, using tools like Jenkins, Travis CI, CircleCI, or GitLab to automate building, testing, and delivering software. Continuous Deployment (CD) extends this automation to the delivery of software to production environments, enabling rapid delivery of changes to end users. CI and CD can be implemented in the same pipeline or in separate ones, depending on preferences and project requirements [5]. A pipeline is an automated sequence of processes that spans from continuous integration to continuous deployment (CI/CD), improving the efficiency and quality of software development by automating code building, testing, and deployment. GitLab [10] is one of the most used tools for implementing and managing these pipelines.

II.c Microservices Architecture.

The microservices architecture involves dividing an application into smaller elements that can be developed and deployed independently, with each performing a specific task. These elements, known as microservices, usually have an associated database, and communicate with each other through simple and clearly defined interfaces (APIs) or events. Since microservices are naturally deployment units, deployability is an inherent quality attribute in this type of architecture.

II.d Containers.

Containers are execution environments, related to virtual machines, that include everything needed to run an application, such as code, dependencies, and configurations, without depending on the underlying infrastructure [7]. The fact that a container packages all of its runtime dependencies makes it an ideal mechanism to implement microservices, however, they can also be used to deploy monolithic applications. Compared to virtual machines, they do not need to contain the whole OS and as a result they are more efficient and lightweight, allowing for consistent and fast deployment in different environments. Docker [8] is the most common container technology in use today.

II.e Deployment Infrastructure.

By deployment infrastructure, we refer to the set of components necessary to manage and deploy an application, such as servers, networks, storage, automation tools, orchestration platforms, among others, which together ensure that the application can be implemented, updated, and maintained efficiently and reliably. In the context of container-based applications, the deployment infrastructure relies on technologies such as container repositories and orchestration technologies such as Kubernetes [9], which is needed for creating, managing, and orchestrating containers.

III. RELATED WORK

This section presents various research studies related to the automation of application deployment, addressing key aspects that are related to the approach proposed in this work. Below, the most relevant articles and their contribution to our project are summarized.

The reference [13] details the implementation of a Docker container management platform to execute and monitor Hadoop and Spark. Docker's performance is compared to virtual machines using KVM. The tools and techniques presented helped demonstrate some ideas proposed in our application refactoring methodology.

The work in [14] presents the Adaptive Container Deployment (ACD) model for automatic deployment of applications in geo-distributed environments. ACD optimizes the horizontal and vertical elasticity of containers to meet resource requirements. The importance lies in its practical and detailed approach to automatic deployment, providing a solid framework adaptable to our methodology.

The research described in [15] presents a continuous integration and deployment model using Jenkins for microservices in MercanetXpress. Integration and deployment tasks are automated, reducing time and errors. The implementation of workflows is relevant to our methodology, allowing for continuous and automated delivery in microservices environments.

The work described in [16] proposes a method to automate the continuous deployment of microservices in the cloud, using DevOps practices and Bitbucket Pipeline for continuous integration and deployment on AWS EC2. The practical approach of the article is crucial for improving efficiency and quality in the development and deployment of microservices-based applications.

[17] talks about a method for implementing a big data platform in the cloud using MiCADO is presented, which automates deployment and scaling of resources. The relevance of the article lies in demonstrating how MiCADO automates deployment and scaling, crucial for our goal of facilitating application deployment in cloud environments.

The work in [18] addresses the autonomous orchestration of containers, identifying research challenges such as resource management and performance monitoring. It highlights the need to automate orchestration in dynamic environments, providing guidance to improve our methodology.

The research described in [19] provides a comprehensive overview of container-based deployment of object storage applications using Docker and Kubernetes. It highlights the importance of Kubernetes for automating deployment and management, and its alignment with DevOps practices to simplify cloud operations.

In summary, these articles have provided a deeper understanding of the concepts and tools related to deployment automation, helping us identify key principles and practices to develop our deployability methodology. Although there is abundant literature in these areas, no specific studies were found on deployability and its integration into applications not originally designed to support this quality attribute, which constitutes the core of this proposal.

IV. Proposal

The proposal focuses on improving deployability by identifying key points to optimize the application deployment process, regardless of its nature or architecture. It is important to note that deployability not only refers to the application but is also directly related to its development and deployment infrastructure. This underscores the need to optimize both the application and the infrastructure, as well as deployment practices, to achieve a more efficient and effective process in development and maintenance.

A Refactoring Methodology to Support Deployability

The relationship between the application and its infrastructure presents a challenge that must be addressed comprehensively. Our methodology, illustrated in Fig 2, aims to incorporate or improve the quality attribute of deployability. To achieve this, the methodology is divided into four distinct stages:

Evaluation Stage: This stage allows us to gather information about the current state of deployment.
Analysis and activity prioritization stage: Provides an overview of the current deployment process and helps identify areas for improvement.
Refactoring stage: Involves making changes to both the application and its infrastructure to optimize the deployment process.
Reevaluation stage: Verifies the implemented changes and checks if the deployability objectives have been met, adjusting the methodology as needed.

Most tasks in Fig 2 are separated into two parallel tracks: tasks shown in blue correspond to the application, while tasks shown in yellow correspond to the infrastructure. Common tasks are shown in purple. It should be noted that this is an iterative methodology. This approach was developed based on practical experience with the evaluated teams and incorporates ideas from agile methodologies, such as continuous integration and DevOps practices.

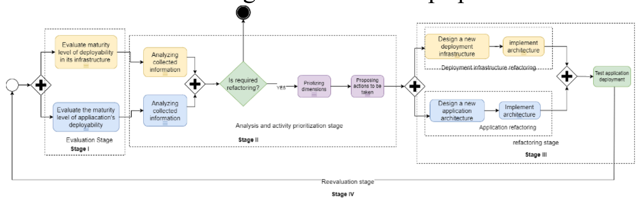

Evaluate maturity level of deployability in its infrastructure
Evaluate the maturity level of appliacation's deployability
Evaluation Stage
Stage I
Analyzing collected information
Analyzing collected information
Is required refactoring?
YES
Priotizing dimensions
Proposing actions to be taken
Analysis and activity prioritization stage
Stage II
Design a new deployment infrastructure
implement architecture
Deployment infrastructure refactoring
Design a new application architecture
Implement architecture
Application refactoring
refactoring stage
Stage III
Test application deployment
Reevaluation stage
Stage IV

Fig 2: Application Refactoring Methodology Diagram to Support Deployability.

As seen in Fig 2, this methodology consists of 4 major stages involving both the application and its deployment infrastructure, which will be described below.

IV.a Stage 1: Evaluation Stage

In this initial stage, a comprehensive assessment of the application's deployability is conducted. Both the application and its infrastructure are analyzed, identifying challenges and obstacles in the deployment process. This evaluation provides a detailed understanding of the current situation and serves as the basis for the following stages, represented by the block shown in Fig 3.

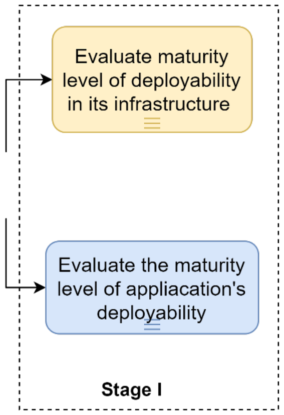

Evaluate maturity level of deployability in its infrastructure
Evaluate the maturity level of appliacation's deployability
Stage I

Fig 3: Evaluation Stage Block

To carry out this evaluation, we have developed two questionnaires: one to assess the current state of application deployment and another to assess its infrastructure. The dimensions evaluated in these questionnaires will be discussed in detail in a later section. The goal is to analyze the support for deployability in a holistic way.

Regarding the application, we have identified the following dimensions to be evaluated:

- AM: Is a microservices architecture used? The characteristics of this architecture facilitate the deployment process due to the independence of each component developed under this scheme.
- CE: Are configurations externalized? By separating configuration from the application's source code, parameters and features can be adjusted in each deployment environment without the need to recompile the source code.
- PA: Is testing automated? Automated tests are essential to support moving the application from the preproduction to the production environments in a rapid way.
- DPD: Are deployment procedures documented? Detailed documentation of deployment procedures provides a clear guide for the operations team, ensuring a reproducible process and minimizing human errors. The existence of this documentation facilitates its automation.
- PO: Is portability considered? Portability ensures not being tied to specific infrastructure and allows the application to adapt to changes in its execution environment.

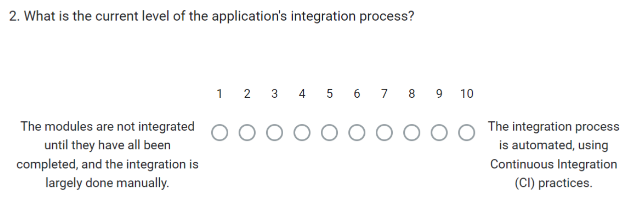
To assess the maturity level of the deployment infrastructure, we consider the following dimensions:

- SDA: Is automated deployment supported? Automation allows for the rapid deployment of new application versions without errors, reducing human intervention and minimizing downtime. Facilitates continuous delivery and ensures consistent deployments.
- AAE: Is provisioning automated? Automated provisioning includes tasks such as resource allocation, software installation, network management, or even the creation of new resources such as virtual machines, ensuring a deployment-ready environment.
- ACE: Is configuration Automated ? ACE Enables efficient management of system configurations and resources created by provisioning, ensuring that the application behaves as expected in different environments.
- **SCV:** Is a version control system used? A version control system facilitates management and tracking of changes in source code and other resources related to the application.
- RA: Is an artifact repository used? This type of repository centrally stores components and resources necessary for deployment, such as libraries, dependencies, and binary files.

IV.a.a Evaluation through Questionnaires

The use of questionnaires in research is essential for gathering data in a structured and efficient manner, ensuring uniformity in responses and facilitating their analysis and interpretation. Additionally, they allow for obtaining personal and sensitive responses, as well as scaling and replicating studies. Proper drafting and structuring of questionnaires are crucial to avoid biases and ensure the quality of research [11].As we had mentioned before, we decided to conduct the evaluation through questionnaires to obtain a deep understanding of the level of maturity in deployability.

These questionnaires include detailed questions about the application, scales from 1 to 10, deployment times between environments, and the number of microservices, as well as spaces for additional comments. Designed to address all the previously discussed dimensions, the questions are tailored to the respective roles within the organization. An example of these questions is shown in Fig 4.



2. What is the current level of the application's integration process?

	1	2	3	4	5	6	7	8	9	10	
The modules are not integrated until they have all been completed, and the integration is largely done manually.	○	○	○	○	○	○	○	○	○	○	The integration process is automated, using Continuous Integration (CI) practices.

Fig 4. Example of a question asked in the questionnaires.

IV.b Stage II: Analysis and Prioritization Stage

Once the data from the initial evaluation has been collected, the results are analyzed to identify the most critical areas for improvement. Activities necessary to address these areas are prioritized, involving careful planning and the establishment of clear objectives to effectively guide the improvement strategy. The block of this stage of the methodology is shown in Fig 5.

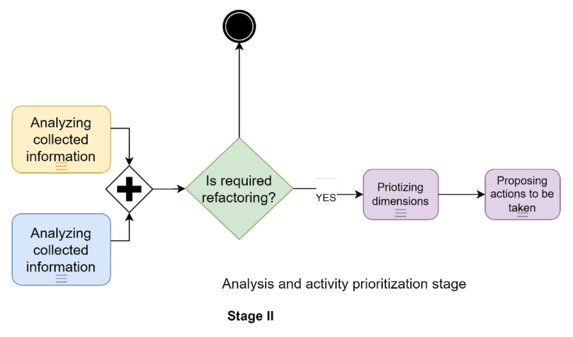
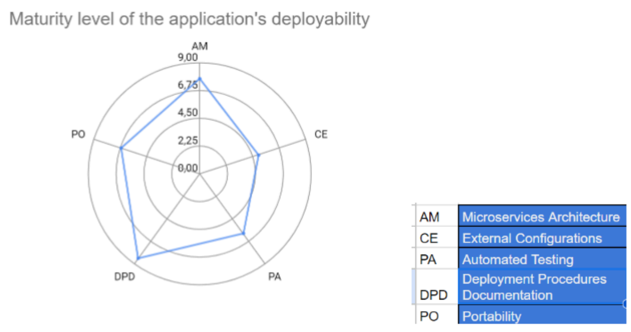

Analyzing collected information
Analyzing collected information
Is required refactoring?
YES
Priotizing dimensions
Proposing actions to be taken
Analysis and activity prioritization stage
Stage II

Fig 5. Block of Stage II of the methodology

In this block shown in Fig 5,The analysis of the collected data involves calculating the average scores for all evaluated dimensions. This average is computed based on the values provided by the developers, with each dimension typically evaluated through an average of five questions, although some dimensions were assessed with just one additional question. Open-ended questions were only included in the average if they resulted in a binary response (yes or no); in these cases, a score of 0 or 10 was assigned based on the response. Open-ended questions that did not provide binary answers were not included in the average calculation but offered supplementary information about the deployment process.

The results are summarized in an evaluation report presented to the development team. This report provides a clear view of the current state of the deployment process, featuring radial charts like the one shown in Fig 6 and application statistics, which allow the team to make informed decisions on how to improve the application's deployability. Based on these results, the team is consulted to determine whether a refactoring activity is necessary or if the system already meets all its needs. This consultation is crucial for reviewing the findings and assessing the importance of the identified areas for improvement. If the team concludes that the system adequately supports deployability, refactoring may be deemed unnecessary. If improvements are needed, activities are prioritized to focus efforts on the most critical areas, thereby maximizing the benefits of refactoring.



AM	Microservices Architecture
CE	External Configurations
PA	Automated Testing
DPD	Deployment Procedures Documentation
PO	Portability

Fig 6: Example of the type of chart presented in the reports.

IV.b.a Activity prioritization

If the results of the previous phase demonstrate that improvements are needed, we carry out an activity prioritization process. To determine which dimensions need to be addressed first, an additional consultation is conducted with the development team, where members collectively decide on the priority of the aspects to be improved. To achieve this, a procedure based on the Quality Attribute

Workshop (QAW) prioritization approach was followed [12].

Below is the process we propose to determine deployment priority:

IV.b.b Procedure for activity prioritization

Initial point allocation: Each team member will receive 4 points to allocate among the 10 evaluated dimensions, covering both the application and the environment. In this initial stage, half of the available points will be used.

Collection of point assignments: After each team member makes his/her point assignments, they will be collected to calculate the accumulated total.

Review and analysis: A detailed review of the accumulated point assignments will be conducted to identify patterns and trends that will be presented to the participants.

Allocation of remaining points: In the second stage, the allocation of the remaining points will be completed. The accumulated total will be recalculated.

Identification of critical points: The method will allow for the accurate identification of critical areas, those that received a higher accumulation of points during both stages.

This procedure provides a systematic and transparent approach to determine the priority of the evaluated dimensions, enabling the development team to focus their efforts on areas that have a greater impact on the improvement of deployability. Based on this priority, we investigate and propose actions to optimize the deployment process. Once approved by the development team, we move on to the next stage of the methodology: the refactoring stage.

IV.c Stage III: Refactoring Stage

Application refactoring involves restructuring the internal architecture of an application without altering its external functionality, with the aim of improving quality attributes such as efficiency, maintainability, and readability. This process aims to elevate the quality of the application over time without making visible changes to users. A key aspect of refactoring is its ability to reduce technical debt, which are technical commitments made during the development of the application. By addressing problematic areas and eliminating obsolete or inefficient practices, refactoring improves the structure and quality of the code, facilitating more efficient continuous development. Additionally, by reducing technical debt, future problems are prevented, and the application is ensured to be easier to maintain and evolve, resulting in more robust and high-quality software [6].

In this stage, the actions identified during the previous analysis are implemented. Modifications are made to the application and/or infrastructure with the aim of improving deployability. This may involve code optimization, configuration updates, or any other measures considered necessary to achieve the established goals. The section of the flow diagram corresponding to this stage is shown in Fig 7.

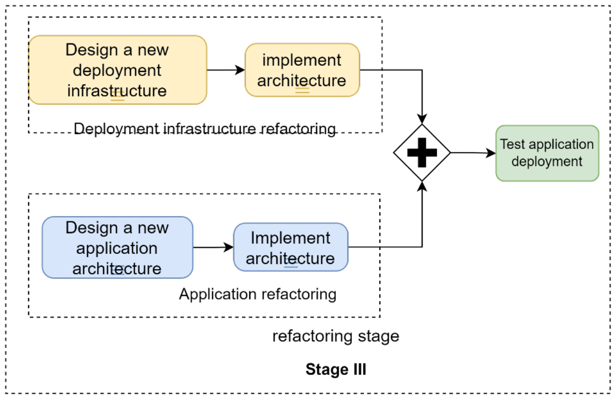

Design a new deployment infrastructure
implement architecture
Deployment infrastructure refactoring
Test application deployment
Design a new application architecture
Implement architecture
Application refactoring
refactoring stage
Stage III

Fig 7. Block of Stage III of the methodology

According to the block shown in Fig 7, the activities to be carried out in this stage are as follows:

Design of the new deployment infrastructure: A new deployment infrastructure is developed to address the areas identified for improvement. This may include changes to pipelines, environment configuration, artifact repositories, among other aspects related to infrastructure.

Design of the new application architecture: Here, design decisions to improve deployability at the application level are identified.

Implementation of improvements: Necessary changes are made to the application and infrastructure through refactoring to incorporate the changes identified previously. This may involve modifying code, configuring automation tools, updating infrastructure, among other aspects according to the chosen priorities.

IV.d Stage IV: Reevaluation Stage

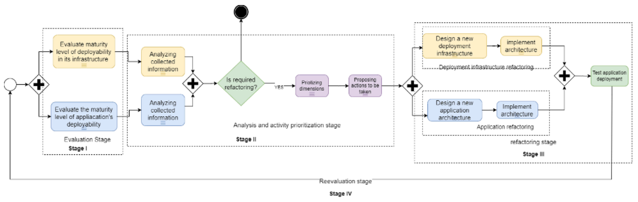

Evaluate maturity level of deployability in its infrastructure
Evaluate the maturity level of appliacation's deployability
Evaluation Stage
Stage I
Analyzing collected information
Analyzing collected information
Is required refactoring?
YES
Priotizing dimensions
Proposing actions to be taken
Analysis and activity prioritization stage
Stage II
Design a new deployment infrastructure
implement architecture
Deployment infrastructure refactoring
Design a new application architecture
Implement architecture
Application refactoring
Test application deployment
refactoring stage
Stage III
Reevaluation stage
Stage IV

Fig 8: Methodology Reevaluation

After implementing the proposed changes during the refactoring process, all steps of this deployment methodology shown in Fig 8 are reapplied. This step is crucial to conclusively verify whether the introduced changes have significantly improved deployability. The following reevaluation steps are presented:

Questionnaire Reevaluation: A new evaluation will be conducted using previously designed questionnaires, allowing for updated data on deployment ease after the implemented changes.

Results Analysis: After collecting and analyzing the new data, areas for improvement and points already offering satisfactory results will be identified, guiding improvement decisions.

Generation of New Reports: Updated reports reflecting the evaluation results will be generated, providing a detailed view of the implemented changes and their impact on the deployment process.

Comparison with Previous State: New results will be compared with previously obtained ones to assess the effectiveness of the changes and verify if the established objectives have been achieved.

Decision Making: The development team will make decisions based on the results, deciding whether to continue improving the system or if the desired objectives have already been achieved.

The proposed methodology can improve the implementation process by reducing costs and time through automation and optimization, increasing efficiency, and decreasing errors. Although it may face challenges such as resistance to change and bureaucratic procedures, careful management can overcome these difficulties and enhance efficiency. Additionally, with a continuous improvement approach, the process is constantly updated to meet the needs of both users and the development team.

V. Results and Analysis

To validate the approach, we applied it in 3 different real projects and a prototype. Because of time constraints, the methodology could not be completely applied in the projects, but it was completely applied in the prototype.

V.a Results of the First and Second Stages

As previously mentioned, during this stage, we evaluated the questionnaires on real applications. For this purpose, we contacted three development teams who allowed us to assess the current state of their deployment process.

It is important to highlight that these applications have different architectures and approaches, providing us with the opportunity to test these initial ideas of the methodology across a variety of applications. The development teams that agreed to collaborate with us to test these ideas were: the ALEPH team (developed at UAM-Iztapalapa), the Xelhua team (developed at CINVESTAV de Tamaulipas), and the DocsyMatOPL development team, developed at the INE (National Electoral Institute).

ALEPH is a distributed storage system based on a microservices architecture, which performs information processing operations including encoding and redundancy generation. Its purpose is to process a file that needs to be stored and later retrieved to ensure its integrity, security, and high availability.

The next application we analyzed was Xelhua. This big data system facilitates the creation of high-availability data science services to support data-driven decision-making. It consists of a design framework for selecting analytical tools, a processing model that converts designs into software, an orchestration model for managing data delivery, and a decentralized model to mitigate service unavailability. Xelhua is applied in projects such as a national cancer observatory and data analysis on suicide, mental health, drug consumption, and macroeconomic data to identify spatio-temporal patterns.

The last development team we analyzed was the one behind the DocsyMatOPL application, a system used by the INE (National Electoral Institute in Mexico) to manage resources during election day. Its objective is to comprehensively track documents and materials from their initial approval by local bodies, such as the IECM, to their final approval by the INE. The application monitors the production of documents by designated companies, validates critical materials like indelible ink, and certifies and generates seals to ensure the integrity and legitimacy of the electoral process.

Finally, as previously mentioned, we developed a small microservices-based application which we refer to as "Prototype," where we applied the methodology in its entirety to obtain an initial view of its effectiveness. Table I shows the results obtained from these applications in this initial stage of the methodology.

Results of Stage I	Current deployment status			
	ALEPH	Xelhua	DocsyMat OPL	Prototy pe
Average deployability (application)	6.783/10	6.7/10	5.93/10	1.93/10
Average deployability (infrastructure)	3.2/10	4.7/10	5.725/10	0.75/10
Deployment time	8 days	30 min	6 hr	5 min

Table I: Results of the evaluated applications.

As seen in table 1, our initial assumption that many applications do not support deployability adequately is validated. There are opportunities for improvement in all the evaluated applications to improve deployability. Additionally, this step allowed us to verify the effectiveness of our questionnaires, as they were well understood by the development teams. We improved each aspect of the questionnaire as we evaluated them with the teams, thus achieving a robust questionnaire that allows obtaining vital information for deployability. In conclusion, the results of stage II consist of analyzing the results and defining whether a refactoring is required or not. In this case, we will only work with the ALEPH system and the prototype.

Regarding ALEPH, after presenting the results to the development team, the decision was made to carry out the refactoring process. For this purpose, the prioritization workshop was applied to determine the most critical dimensions to address, which allowed us to research and propose actions to improve its deployment process to the development team.

On the other hand, activities were established to refactor the prototype with the aim of obtaining foundational knowledge to automate other more complex applications, such as ALEPH.

V.b Third stage results

As mentioned earlier, in this third stage, we carried out the refactoring process in the prototype, for which the following actions were taken:

External Configurations:

The routes of the prototype microservices were externalized for greater flexibility in future changes.

Automated Testing:

A set of unit tests for the microservices were defined and executed using JUnit, ensuring integrity and functionality before deployment.

Automated Deployment Support:

A project was created in GitLab, and a CI/CD pipeline was set up, divided into two stages:

Build: Compiles, tests, and builds the JAR files of the microservices, and stores them in GitLab's artifact repository.

Dockerize: Creates Docker images for the microservices and stores them in a local image repository.

Environment Provisioning: A local Kubernetes cluster was set up for production environments, and a local Docker repository was created for storing the images, facilitating the deployment of the prototype.

Portability: Creation and storage of Dockerfiles for the microservices, ensuring consistency and reproducibility in the development and deployment infrastructure.

These actions enabled the automation of compilation, testing, JAR generation, Docker image creation, and continuous deployment of the prototype in a Kubernetes environment.

V.c Fourth stage results

For this final stage, we re-evaluated the methodology to verify if we improved the deployment process of the prototype, to what extent, and if there are still dimensions to work on, or if deployability is sufficiently satisfied. Below is a comparative table of before and after the prototype's deployment process.Table II Comparison Before and After Prototype Deployment

Prototype	State before refactoring	State after refactoring
Average deployability (application)	1.93/10	8.8/10
Average deployability (infrastructure)	0.75/10	8.2/10
Deployment time	5 min	30 sec

Table II presents the data before and after the refactoring, where we concluded the following:

Deployability (Application):

Before: The prototype's deployability was very low, with an average score of 1.93 out of 10.

After: Following the refactoring, this average significantly improved to 8.8 out of 10, indicating that the implemented measures have significantly eased the deployment process.

Deployability (Infrastructure):

Before: The deployment infrastructure had a very low score, averaging 0.75 out of 10.

After: Post-refactoring, the average increased to 8.2 out of 10, reflecting substantial improvements in the configuration and management of the deployment infrastructure.

Deployment Time

Before: The time required to deploy the prototype was 5 minutes.

After: This time was drastically reduced to 30 seconds, thanks to the automation and optimization of the deployment process.

The application of this methodology has not only improved the deployability at the application and infrastructure levels of the prototype but has also substantially impacted the deployment time. These results provide an initial validation of the effectiveness of the proposed methodology and highlight the importance of addressing not only the application but also its deployment infrastructure.

VI. Conclusions

The application of our methodology provided a comprehensive analysis of the current state of deployability in the evaluated applications, revealing very low initial scores due to a lack of consideration for this quality attribute in their design. Surveys highlighted developers' limited knowledge about deployability, prompting educational interventions. Collaborating with development teams led to improvements in our evaluation tool, creating a robust questionnaire that better presented analysis results and identified improvement opportunities. Furthermore, the absence of good deployment infrastructure across all applications necessitated further research into its creation and automation.

In the second stage, detailed analysis and improvement proposals were delivered to the development teams, focusing on refactoring the deployment process. Working closely with the ALEPH team, a workshop helped prioritize dimensions and propose a refactoring action plan, significantly benefiting the team's process. The third stage involved implementing the proposed refactoring activities, although this could only be performed for the prototype. Through refactoring, we achieved a 90% reduction in deployment time and significantly improved deployability scores without altering application functionality. This refactoring also established a solid deployment infrastructure applicable to more robust applications.

Overall, we believe our methodology is a useful tool that allows deployability to be enhanced at both the application and infrastructure levels.

VII. Future work

As we can see, this methodology has many implications, and each of its stages could lead to independent research.

Although it lays the groundwork for improving deployability, it is still in its early stages. As future work, we hope to analyze more applications to find or refine the concepts presented, and we expect to be able to perform the complete method on a number of read applications.

We also need to delve deeper into the automation of deployment infrastructure, a crucial part of this process. There are still many opportunities to leverage this initial version of the methodology. With more research, we hope to develop an improved version that helps development teams incorporate these ideas into their projects, thus improving the support for deployability which is essential in today's systems.

Acknowledgment

I extend my deepest gratitude to Dr. Humberto Cervantes and Dr. Ricardo Marcelin for their invaluable guidance and support throughout the development of the project. Their expertise and advice were fundamental in achieving our objectives.

Additionally, I want to thank the development teams of Aleph, Xelhua, and DocsMatOPL for their collaboration and the time they dedicated to testing this methodology.

Lastly, I appreciate the support and contributions of all involved in this project. From start to finish, their efforts were indispensable, and I am sincerely grateful for their commitment.

References

[1] Humble, J., & Farley, D. (2010). Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation (Edición Illustrated ed.). Addison-Wesley Professional.

[2] H. Cervantes, R. Kazman. (2024). Designing Software Architectures: A Practical Approach (2d. Edition ed.). Addison-Wesley Professional.

[3] Documentation. (n.d.). Git. Retrieved May 14, 2024, from https://git-scm.com/doc

[4] Apache Subversion Documentation. (n.d.). Apache Subversion. Retrieved May 14, 2024, from https://subversion.apache.org/docs/

[5] Lukša, M., & Conner, K. (2024). Kubernetes in Action, Second Edition. Manning.

[6] Fowler, M. (2019). Refactoring: Improving the Design of Existing Code. Addison-Wesley.

[7] Sayfan, G. (2023). Mastering Kubernetes: Dive Into Kubernetes and Learn How to Create and Operate World-Class Cloud-native Systems. Packt Publishing, Limited.

[8] Kane, S. P., & Matthias, K. (2023). Docker Up & Running: Shipping Reliable Containers in Production. O'Reilly.

[9] Docker. (n.d.). Docker Docs. Retrieved April 11, 2024, from https://docs.docker.com/

[10] CI/CD pipelines | GitLab. (n.d.). GitLab Documentation. Retrieved March 30, 2024, from https://docs.gitlab.com/ee/ci/pipelines/

[11] Linaker, J., Sulaman, S. M., Höst, M., & de Mello, R. M. (2015). Guidelines for conducting surveys in software engineering v. 1.1. Lund University,, (50).

[12] De Gooijer, T. (2017). Discover Quality Requirements with the Mini-QAW. IEEE International Conference on Software Architecture Workshops (ICSAW), Software Architecture Workshops (ICSAW), 2017 IEEE International Conference on, ICSAW, 196–198. https://doi.uam.elogim.com/10.1109/ICSAW.2017.52

[13] Shih, W.-C., Yang, C.-T., Ranjan, R., & Chiang, C.-I. (2021). Implementation and evaluation of a container management platform on Docker: Hadoop deployment as an example. Cluster Computing: The Journal of Networks, Software Tools and Applications, 24(4), 3421–3430. https://doi.uam.elogim.com/10.1007/s10586-021-03337-w

[14] Nardelli, M., Cardellini, V., & Casalicchio, E. (2018). Multi-Level Elastic Deployment of Containerized Applications in Geo-Distributed Environments. IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud), Future Internet of Things and Cloud (FiCloud),(IEEE 6th International Conference on, FICLOUD, 1–8.). https://doi.uam.elogim.com/10.1109/FiCloud.2018.00009

[15] Quecha Claribel, B., Altamirano C. M., López G. O. E., Fuentes G. J. E., Toral E. F., & Méndez L. M. D. (2021). Diseño y Montaje de Un Modelo de Integración y Despliegue Continuos Para Una Arquitectura de Microservicios Orientada a Computación En La Nube. Congreso Internacional de Investigacion Academia Journals, 13(10),248–253. https://search.ebscohost.com/login.aspx?direct=true&db=edb&AN=161293566&lang=es&site=eds-live&scope=site.

[16] Vera-Rivera, F. H. (2018). Método de automatización del despliegue continuo en la nube para la implementación de microservicios. In Proc. 21st Conf. Iberoamericana Ingeniería Softw.(CIBSE), 597-604.

[17] Mosa, A., Kiss, T., Pierantoni, G., DesLauriers, J., Kagialis, D., & Terstyanszky, G. (2020). Towards a Cloud Native Big Data Platform using MiCADO. 19th International Symposium on Parallel and Distributed Computing (ISPDC), Parallel and Distributed Computing (ISPDC), 2020 19th International Symposium On,, 118–125. https://doi.uam.elogim.com/10.1109/ISPDC51135.2020.00025

[18] Casalicchio, E. (2016, October). Autonomic Orchestration of Containers: Problem Definition and Research Challenges. In VALUETOOLS.

[19] Jadhav, S. P. (2020). Object Storage and Kubernetes Containerization of Object storage Application. International Journal of Engineering Research & Technology (IJERT), 9(9).