TÉCNICO LISBOA

# Enterprise Architecture in an Agile world

**David Apolinário Garcia Pereira**

Thesis to obtain the Master of Science Degree in

## Information Systems and Computer Engineering

Supervisor: Prof. Pedro Manuel Moreira Vaz Antunes de Sousa

### Examination Committee

Chairperson: Prof. Mário Jorge Costa Gaspar da Silva
Supervisor: Prof. Pedro Manuel Moreira Vaz Antunes de Sousa
Member of the Committee: Prof. Alberto Manuel Rodrigues da Silva

**November 2022**

# Acknowledgments

I would like to express my deepest appreciation to Professor Pedro Sousa for providing me with an enriching experience and the guidance needed to achieve these results.

I would like to extend my sincere thanks to the whole team at Link Consulting who always helped me solve the challenges that arose while developing this project, in particular, João Zeferino, David Moreira, and Alexandre Marques.

I would also like to thank Instituto Superior Técnico for giving me the knowledge and tools I needed for this project and for preparing me for any future challenges.

I am also thankful to my family, who have shown nothing but unwavering support, allowing me to thrive as an upcoming adult and working engineer.

Finally, I would like to thank my loving girlfriend, Ana Sofia Fernandes, and my Wonder Team (Miguel Marcelino, Vasco Castro, Diogo Soares, Samuel Ferreira, and Andreia Batista), who provided me with encouragement and patience throughout my academic years.

# Abstract

Enterprise Architecture and Agile methodologies are two increasingly important notions in the enterprise world. The first is crucial to ensure business success through documentation of the current practices of the enterprise, and enable innovation through the safety of well-documented processes and functions. The latter helps reduce operation costs, and in turn, maximize profits, through different software development processes. This study utilizes the already defined workflows of an Agile development project to develop automated processes that help in creating and maintaining architectural representations of the project. The Enterprise Architecture follows the principles stated in TOGAF's ADM and it uses the Atlas Enterprise Cartography Tool to support the creation of dynamic architectural assets, providing a mapping between Agile development and Enterprise Architecture.

# Keywords

Enterprise Architecture; Agile methodologies; TOGAF ADM; Atlas Enterprise Cartography Tool.

# Resumo

Arquitectura Empresarial e metodologias ágeis são duas noções cada vez mais importantes no mundo empresarial. A primeira é crucial para assegurar o sucesso empresarial através da documentação das práticas atuais da empresa, e permitir a inovação através da segurança de processos e funções bem documentadas. A Arquitetura Empresarial ajuda a reduzir os custos de operação e, por sua vez, a maximizar os lucros, através de diferentes processos de desenvolvimento de software. Este estudo utiliza os fluxos de trabalho já definidos de um projecto de desenvolvimento ágil para desenvolver processos automatizados que facilitam a criação e manutenção de representações de Arquitectura Empresarial do projecto. A arquitetura criada segue os princípios do framework TOGAF ADM. O Atlas Enterprise Cartography Tool apoia a criação de bens arquitectónicos dinâmicos, permitindo um mapeamento entre o desenvolvimento ágil e a Arquitectura Empresarial.

# Palavras Chave

Arquitetura Empresarial; Desenvolvimento ágil; TOGAF ADM; Atlas Enterprise Cartography Tool;

# Contents

# List of Figures

# List of Tables

# Listings

# Acronyms

**ADM** Architecture Development Method

**API** Application Programming Interface

**CPU** Central Processing Unit

**IDE** Integrated Development Environment

**JSON** JavaScript Object Notation

**PAT** Personal Access Token

**PBI** Product Backlog Item

**RAM** Random Access Memory

**REST** Representational State Transfer

**TOGAF** The Open Group Architecture Framework

# 1

# Introduction

## Contents

This Chapter includes the motivation for the presented study, the objectives, an overview of the concepts and tools utilized throughout it, and the structure of the document.

The notion of Enterprise Architecture is ever-present within organizations. With the demanding requirements of the technological landscape, it is becoming harder to process information and control change. Enterprise Architecture is therefore gaining importance, as a method to control and document the organization's current practices and as an enabler of change and evolution [1].

On the other hand, software development methodologies are evolving to prioritize a fast product delivery, focused on the client's needs. This is achieved by bridging the gap between the business and the development aspects, and prioritizing in-person meetings and discussion. These are the principles of Agile development, as stated in the Agile Manifesto [2].

Given Agile's focus on delivering software and limit the time spent on the documentation aspects of the project, this can highly affect the capability to support the project in the long run or to replicate it altogether, limiting the value that could have been earned from it. However, Enterprise Architecture values this documentation. Its main focus is to generate value by creating representations of the organization's assets, to better assess strengths and weaknesses, change, and threats to the company's growth [1].

There are some incompatibilities between Enterprise Architecture and Agile development. The main incompatibility is the Agile mindset of focusing on delivering complete software as fast as possible and sacrificing the documentation to achieve that.

Joining these two domains could lead to several benefits. Documenting the Agile projects using Enterprise Architecture principles would generate value for the organization. Deploying mechanisms that simplify the creation of such material, would allow Agile development teams to focus their efforts on creating valuable software whilst maintaining a detailed architecture of the project.

Nevertheless, given the fast-paced nature of Agile projects, the documentation must be able to be quickly adapted to fit ever-changing requirements. Failing to do so will result in an outdated architecture that requires too much human effort to adapt, and will therefore provide little to no value to the project and to the enterprise.

To create dynamic documentation, the Atlas Cartography Tool [3] is used. This tool implements several Enterprise Architecture principles and easily allows for the creation of dynamic representations, that will automatically evolve as new information is added to its repository. More information regarding the Atlas Enterprise Cartography Tool can be found in Section 1.1.3. Furthermore, The Open Group Architecture Framework (TOGAF)'s Architecture Development Method (ADM) [4] is used alongside Atlas, guiding the development of the Enterprise Architecture assets. This framework is detailed in Section 1.1.2.

The result is an Enterprise Architecture that follows the Agile principles, generating as much documentation with the available information as possible, whilst allowing for manual creation of new informa-

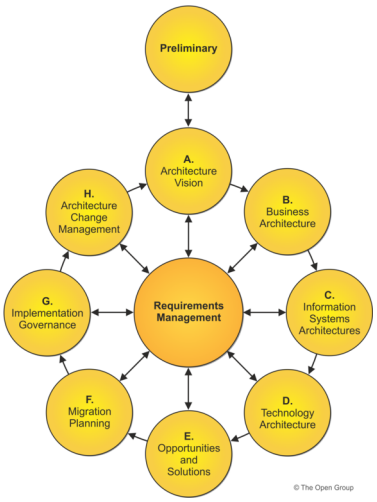


**Figure 1.1:** Phases of TOGAF's Architecture Development Method

tion when needed, to enrich the resulting architecture. This architecture includes representations that automatically evolve with the new information gathered during the development process.

## 1.1 Background Work

This Section will provide some insight into the concepts of Enterprise Architecture and the TOGAF ADM framework, as well as Agile Methodologies. Finally, some details of the Atlas Enterprise Cartography Tool are presented in Section 1.1.3.

### 1.1.1 Enterprise Architecture

Enterprise Architecture is a widely studied topic. Zachman defines Enterprise Architecture as "that set of descriptive representations (i.e. 'models') that are relevant for describing an Enterprise such that it can be produced to management's requirements (quality) and maintained over the period of its useful life (change)" [1]. These representations are created by joining domains that may have been unrelated in the past. This poses a challenge, as different domains are certain to differ in definitions and practices used [5].

With the gradual increase in the number of enterprises and the increasingly competitive market, innovation is now more than ever a requirement for success. Having good Enterprise Architecture practices is crucial. Providing insight on the business processes of the company, as well as keeping records of customers and partners are two advantages of Enterprise Architecture [6].

### 1.1.2 TOGAF ADM

To guide the creation of the project's Enterprise Architecture, TOGAF's Architecture Development Method [4] was the chosen framework. This framework supports the creation of the architecture through different phases that can be adapted to the needs of each Enterprise, with a focus on managing requirements during the whole cycle. Additionally, several documents and assets are recommended to support each phase. An overview of the Architecture Development Method can be seen in Figure 1.1. This Section contains a brief description of each phase of the ADM.

#### 1.1.2.A Phase 0: Preliminary Phase

Contains the groundwork of the cycle. In this phase, details such as the context and the scope of the architecture are detailed. One of the main goals of this process is to adapt the ADM to the needs of the enterprise, highlighting what practices should be streamlined, and what documentation is relevant.

#### 1.1.2.B Phase A: Architecture Vision

The main objectives of this phase are the definition of the value that should be provided by the architecture, as well as the compilation of information regarding vision, strategy, and goals. To achieve this, one needs to examine the capabilities that the enterprise has to achieve the desired outcome, the value streams or activities that offer value to a customer or stakeholder, and the relationships among the entities of the enterprise, partners, and stakeholders.

#### 1.1.2.C Phase B: Business Architecture

The main objective of this phase is the creation of rules and guidelines that detail how the architecture will achieve the required business goals. To do so, it is recommended to start from already existing models and resources. After analyzing them, they can be used to create the Baseline Business Architecture Description, as well as a Target Business Architecture Description. These documents include requirements and other important business-related elements, that are important to the current state of the architecture, and to the future state, respectively. An additional document that compiles the decision-making process, and information regarding the core business functions, is also an output from this phase. This document is then used as an input to the next architecture phases, namely, Phase C and Phase D.

#### 1.1.2.D Phase C: Information Systems Architecture

The Information Systems Architecture phase is divided into Data and Application Architecture. The main objective of this phase is to create a mapping between the Business Architecture artifacts and the Data and Application artifacts. During this phase, the architecture of the Information Systems is designed,

and the main components of those systems are specified. Furthermore, models and other resources relevant to both Data and Application scopes are analyzed and included in a Roadmap. Finally, an Architecture Definition Document is created. This document includes details regarding the decisions made throughout the definition of the Information Systems Architecture, supported through models and different matrices and graphics.

#### 1.1.2.E Phase D: Technology Architecture

The Technology Architecture phase creates documentation regarding the decisions behind technological choices. The details of technological components and services and the way that they impact the overall architecture are described during this phase. Similar to phases B and C, the first step is to analyze the current documentation and build models and documents that will then be used to support the architecture. The result is an Architecture Definition Document that includes the technological decisions, building blocks, and interfaces (i.e. hardware and software used).

#### 1.1.2.F Phase E: Opportunities & Solutions

After creating the different scopes of the architecture, the next objectives include creating a complete Architecture Roadmap, utilizing the information received from phases B, C, and D. Furthermore, another aspect that is analyzed during this phase is the Transition Architecture. If the project requires several iterations of the ADM, it is important to highlight the valuable assets going forward. It is also during this phase that the Target Architecture is finalized. This is done with the past documentation in mind and represents the final goal for the architectural work.

#### 1.1.2.G Phase F: Migration Planning

This phase is mainly focused on ensuring that the Architecture Roadmap is feasible, through estimations and risk assessment, and creating a Migration Plan that includes the strategy for implementation and migration, milestones, costs, and other important information.

#### 1.1.2.H Phase G: Implementation Planning

Monitoring the implementation, and whether the result is in line with what was planned, is the next step in the cycle. This includes reviewing the documentation produced so far, identifying methods that will aid in the development of the solution, and reviewing compliance.

#### 1.1.2.I Phase H: Architecture Change Management

The final phase of the Architecture Development Method is concerned with the lifecycle of the architecture, change, and compliance with requirements. Steps in this phase include monitoring risk, performing the verified Change Requirements, and creating a new Request for Architecture Work, that will be used as a baseline for the next iteration of the ADM cycle.

### 1.1.3 Atlas Enterprise Cartography Tool

The Atlas Enterprise Cartography Tool is an instrument that can be used to control transformations within an organization, through the use of Enterprise Architecture principles.

To achieve this, Atlas provides a repository where data can be gathered both manually and automatically and several representations of the imported data, that can be configured to suit the needs of the enterprise.

#### 1.1.3.A Atlas Data Model

Regarding data, Atlas follows the Object Relationship Model to internally store information. Users can define classes with different properties and relations between objects. Each class can then be instantiated by objects with certain properties and relations.

When creating a Country class, the user can specify what properties a Country object may have. For example, the object `Portugal` of class Country has the following properties: Name, Inhabitants, Language, and President. The President property is established as a relation to an object of class Person. Figure 1.2 includes a representation of the `Portugal` object in Atlas. Each of the aforementioned properties includes a value, notably, the President property is established through a relationship to an object of class Person with the Name property equal to `Marcelo Rebelo de Sousa`. The properties and relations of each object can then be used to generate representations and queries can be created to further analyze the data. Examples of this can be found in Section 1.1.3.C. Figure 1.3 contains an example of Atlas' Object-Relationship model for the `Portugal` object. The relationship with the Person class is achieved through the *managed by* relation type, which corresponds to the type of relationship that was chosen when configuring the class.

#### 1.1.3.B Atlas Structure

Atlas is divided into nine main categories. Each category offers different features to its users. A brief description of the more relevant categories in the context of this project can be found below.

**Explore Tab:** The Explorer View gives access to navigation shortcuts that can be configured according to the needed configuration.

| Name ↑ | Value |
|---|---|
| Name * | Portugal |
| Inhabitants | 10310000 |
| Language | Portuguese |
| President | Marcelo Rebelo de Sousa |

**Figure 1.2:** Properties of an Atlas Object - Example

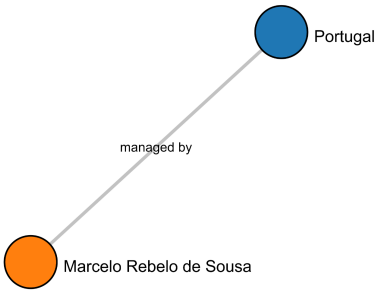


**Figure 1.3:** Atlas Object Relationship Model - Example

**View Explorer Tab:** The View Explorer View supports the creation of different representations of data. These representations are dynamic, and can evolve with new information added into the repository. The main types of representation that will be used throughout the project are the Blueprint Canvas, Matrix Canvas, and Tabular Views. More information regarding representations in Atlas, can be found in Section 1.1.3.C and their creation is detailed in Section 3.3.6.

**Data Explorer Tab:** The Data Explorer View includes a list of every object that has been loaded into the repository alongside its properties and relations.

**Query Explorer Tab:** The Query Explorer View allows for the creation of queries that interact with objects. These queries can be used to filter instances with certain properties or relations. Additionally, these queries can be combined with the representations, to create additional value.

**Batch Explorer Tab:** The Batch Explorer View supports the use of Java code to extend the functionalities of Atlas and create automation processes. This feature is particularly necessary when integrating external services into Atlas or when dealing with more complex requirements.

#### 1.1.3.C Atlas Representations

One of the crucial aspects of an Enterprise Architecture are the different representations and documents. Atlas supports this through representations. The objects and relations present within the Atlas repository can be represented in blueprints that can be configured to display exactly what the user needs.

Atlas supports several types of representations, most notably, tables that contain objects and their

properties, matrices that relate objects of two classes on a specific relation, and blueprints. Figure 1.4 contains an example of the latter. The Application Layered Blueprint representation utilizes the *Campaign Analytics Application* object (of the Application Component class) as an argument, and it shows the different application modules of the main application (*Presentation Module*, *Integration Module*, *Data Module*, and *Business Module*). It also represents the Business Process objects that use it as well as the System Software objects that are needed to ensure that this application is supported. Figure 1.5 contains an example of a matrix. In this particular example, the rows represent Applications and the columns represent Business Processes. The representation shows that the *Campaign Analytics Application* currently supports the *2-1 - Manage Organization* Business Process and that the *Campaign Management Application* currently supports the *2-2 - Administrate HR* Business Process. This information can be quickly edited directly in the representation, either by clicking a cell where a new relation will be created, or by removing an existing relation.

Every Atlas representation is automatically generated with the most recent data, which ensures that whenever the *Campaign Analytics Application* object is related to other Business Process or System Software objects, the respective objects will be shown in the blueprint. Additionally, Atlas supports object lifecycle. This means that every object can include information regarding when the object started being conceived, when the object started its alive state, and when the object has reached its end, and is no longer alive. To quickly check the evolution of the object's lifecycle, Atlas also supports a time bar with different milestones. Navigating through it will change the blueprint to reflect the state of the architecture on that date, either in the past or planned changes in the future. An example can be seen in Section 4.2.

### 1.1.4 Agile Development Methodologies

Agile Software Development methodologies are defined in the Agile Manifesto [2] through a list of principles. Agile values the continuous delivery of quality software to the customer, while embracing change in the requirements "even late in development". Another important aspect of Agile is having the business side and the developers working closely together on a daily basis, with the goal of reducing development time and improving agility and adaptability.

Figure 1.6 contains the Scrum Framework. Scrum development is based around a small team of people that have the capability to self-manage their work and focuses on being nimble. The team works in Sprints, usually no longer than three weeks and planned before their start. A Backlog with the requirements and remaining work is compiled and used to plan each Sprint. Before the Sprint starts, the team estimates what items from the Backlog will be developed, according to their estimated effort and total time available, and a Sprint Backlog is created. During the Sprint, the team is responsible for completing the items in the Sprint Backlog, and a Daily Scrum meeting is recommended to assess the progress. At the end of the Sprint, the results are analyzed, and the new Sprint is planned with new

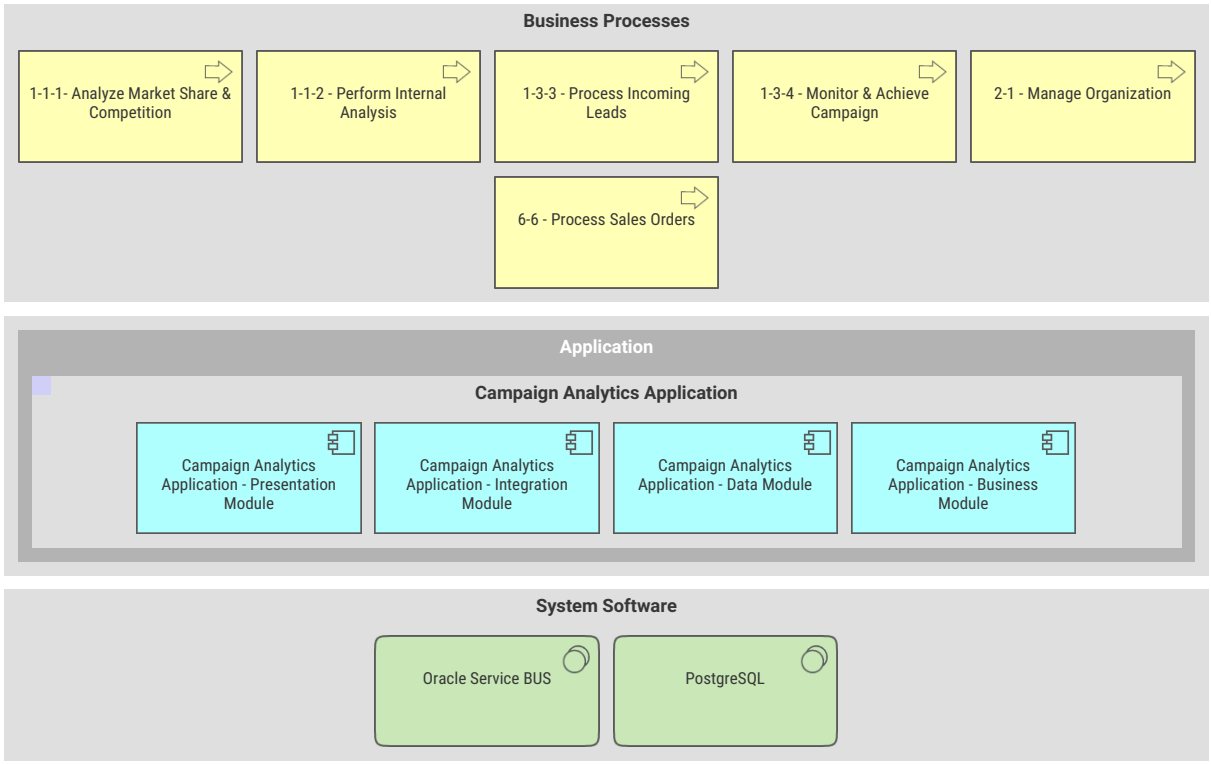


**Figure 1.4:** Atlas Application Layered Blueprint - Example

| | 1-1 - Analize Marketpl | 1-2 - Develop Marketin | 1-3 - Conduct Campaig | 2-1 - Manage Organiza | 2-2 - Administrate HR |
|---|---|---|---|---|---|
| Campaign Analytics Application | | | | ■ | |
| Campaign Management Application | | | | | ■ |

**Figure 1.5:** Atlas Application and Business Process Matrix - Example

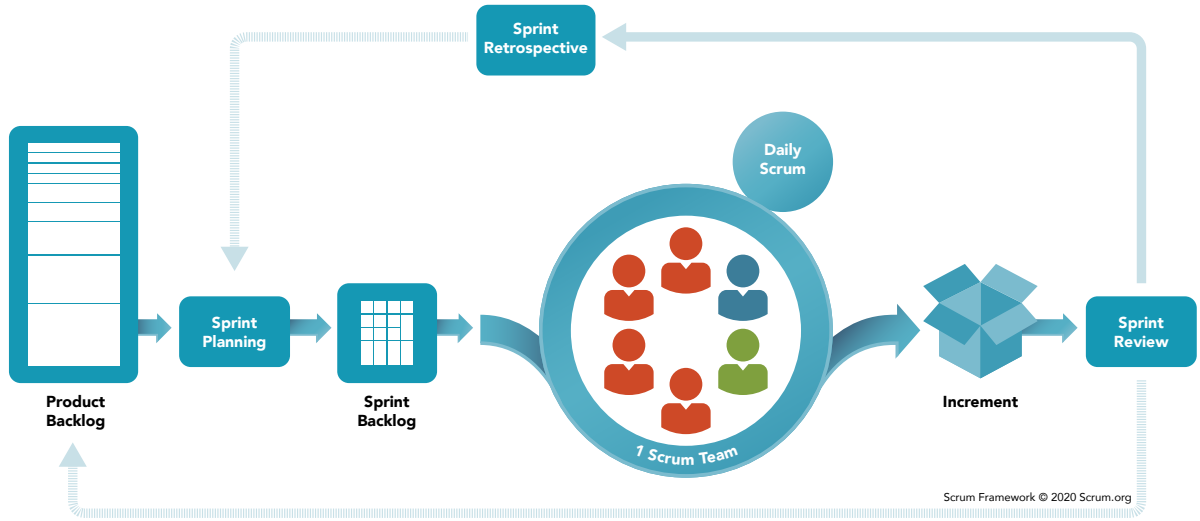


**Figure 1.6:** The Scrum Framework

items from the Backlog [7].

Agile Development is increasing in popularity among enterprises. Scrum [8] is currently the most widely used Agile development methodology, representing 66% of overall Agile use, according to Digital.ai's 15th State of Agile Report [9]. Additionally, 52% of respondents claim that their enterprise has adopted Agile as the go-to development methodology, and only 3% claim that Agile development is not used on any scenario. https://www.overleaf.com/project/628a70dae4afccd7d76c6cda

## 1.2 Objectives

The goal of this project is to develop a tool that can be used by Agile development teams to easily generate Enterprise Architecture documentation. Reducing the effort necessary to track the changes to the architecture of the project will lead to the production of more architectural assets.

To achieve this, Atlas was used as a supporting tool, to gather data from an Agile development project. An integration of Microsoft's Azure DevOps was created within an Atlas repository. More information regarding this solution and the supporting Agile development project can be found in Chapter 3.

Additionally, to quantify the success of the proposed solutions, the following statements will be taken into account.

**A - Quality of the generated Enterprise Architecture assets:** It is of prime importance that the generated documentation can contribute to the overall value of the project and to the organization as a whole. Hence, producing updated and relevant representations is one of the priorities of this solution.

| Objective | Description |
|---|---|
| Automatically gather data from different sources, without disrupting the team's workflow | It is crucial that the proposed solution is compatible with the current workflow of the development team, to ensure that it provides value to it, rather than posing as a challenge |
| Automatically generate architectural documentation supported by the gathered information | It is crucial that the generated assets are generated without requiring excess human intervention, which would reduce the usability of the tool |
| Update the documentation as needed, without introducing too much human effort. | It is crucial that the process of updating the assets is simple and that the updates are regular, such that the result is a valuable asset as development occurs |
| Assess the usability of the created documents | It is crucial that the result is a tool that is simple to use and that can be quickly learned and explored by the project team, clients, and other parties |

**Table 1.1:** Description of the Objectives

**B - Manual effort required to generate and maintain the assets:** With added effort being the number one deterrent to the creation of Agile projects' documentation, the process of generating such documents with the proposed solution must be as automated as possible. Furthermore, updating these documents also proves a challenge. Outdated documentation represents little to no value to both the project and the enterprise. It is, therefore, crucial that the generated representations are easily maintained and evolve to accommodate project changes and progress.

Having these considerations in mind, Table 1.1 contains the objectives and their description, for the final iteration of the solution. These emphasize the need of generating a Solution that does not negatively impact the already existing workflow, and that can easily evolve to meet the ever-changing needs and requirements of the development team and client.

This thesis is divided into five Chapters. Chapter 1 includes the motivation, objectives, background work and structure of the document. Chapter 2 includes an overview of the literature surrounding the topic of Agile development and Enterprise Architecture. Chapter 3 discusses the implementation details of the proposed solution. Chapter 4 highlights the results that have been found from the creation of the Enterprise Architecture and its assessment, focusing on the achieved goals and limitations of the solution. Finally, in Chapter 5, a conclusion is presented alongside future work suggestions.

# 2

# Related Work

## Contents

This Chapter covers the work that has been considered as relevant for this topic. It is divided into three main Sections.

1. Compatibility between Enterprise Architecture and Agile development
2. Models that aim to connect Enterprise Architecture and Agile development
3. Summary and discussion of the findings

Section 2.1 contains the findings related to the perceived compatibility of both ideas and some pointers as to how to bridge the gap. Section 2.2 focuses on the analysis of several models that propose mappings between Enterprise Architecture and Agile development. Finally, Section 2.3 includes a summary and discussion of the findings.

## 2.1 Compatibility

The differences between Enterprise Architecture and Agile development are well-established and have been discussed in Chapter 1. Therefore, the first category that will be explored is the viability of merging both of these notions, and the advantages and disadvantages that are associated with it. Furthermore, analysis of the main benefits, possible connecting points, and related difficulties are also provided.

The notion that Enterprise Architecture and Agile development can be coupled appears to be widely accepted in this area. When questioned about it, professionals believe that even with all the aforementioned differences, merging Enterprise Architecture and Agile development should be possible, at least in some levels [10].

There are, however, some difficulties. A frequently observed statement references the difficulty of manually maintaining relevant architectural representations in the context of a software development project ("Low-level models [...] are considered to be too expensive to maintain manually compared to the benefits they bring") [10], leading to poorly documented projects. Adding to this, most established Enterprise Architecture frameworks are based on Waterfall development methodologies [11], making it harder to adapt them to the changing requirements and overall focus of an Agile development project [12].

Deploying Enterprise Architecture practices in Agile projects, will result in more readiness when dealing with change and improve adaptability with emerging technologies [11]. One way to achieve this is by shortening the distance between the development of the projects and the creation of the architecture. Having developers and architects work too far apart will reduce the effectiveness of communication, leading to "a lack of understanding and seeing the value of having an enterprise or solution architect" [10] and reducing the possibility of early and frequent delivery, therefore hindering the value of the overall architecture [11].

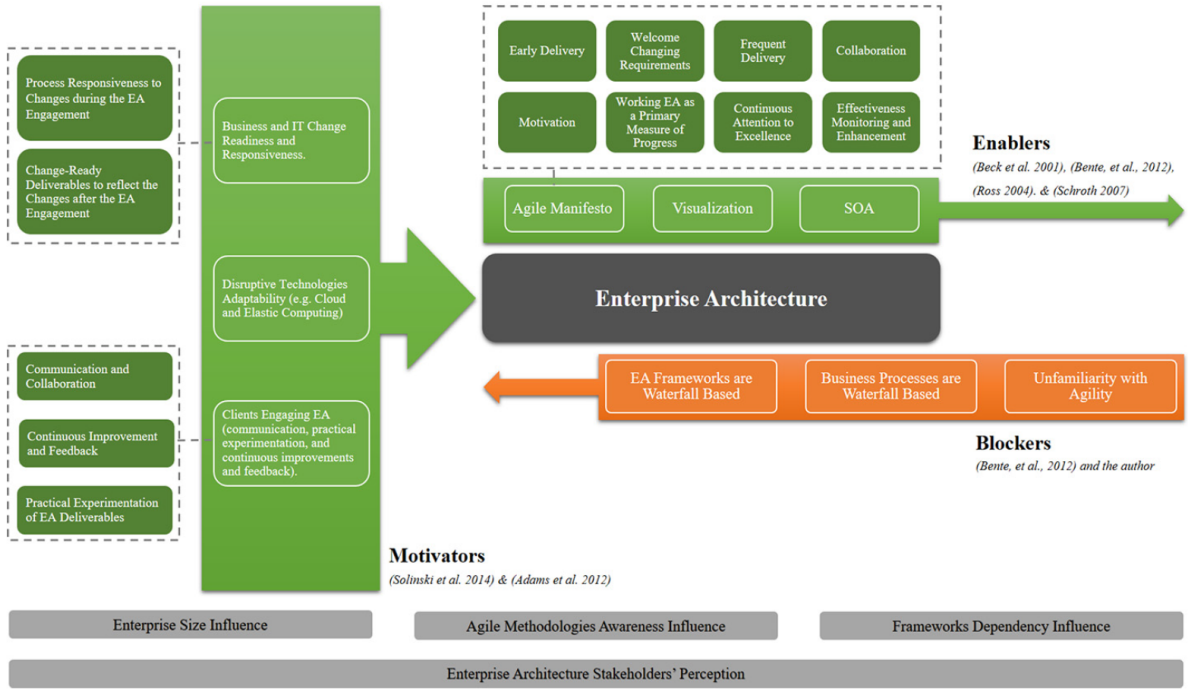


**Figure 2.1:** Watfa and Kaddoumi's Final Framework

Regarding the architectural assets themselves, there is a need for rich and updateable representations. A commonly identified flaw is the lack of detail and integration of these assets ("EA models should contain information about the software portfolio and dependencies between software and technology infrastructure" and "there should be more interaction between the high-level business models and lower-level solution models" [10]).

## 2.2 Proposed Frameworks and Models

After establishing that combining Enterprise Architecture and Agile development is possible and includes several possible benefits to both the development teams and the governance aspect of organizations, in this Section, a summary of different proposed frameworks and models is presented. These models utilize TOGAF's ADM [4] as an enabler of said connection, and propose different approaches to achieve it. The proposed models focus on different phases of the Architecture Development Method (Section 1.1.2 contains more information about the ADM).

### 2.2.1 Architecture Definition

Regarding the earlier phases, it is important to simplify the architecture definition process. Starting by gathering requirements for every stakeholder of the project will result in a more streamlined pro-

cess. This can be followed by the creation of representations, such as a matrix that relates the different requirements with the corresponding ADM phase. Afterward, the relevant ADM documents and deliverables can be selected, leaving out the ones that were deemed as irrelevant. This will result in a well documented starting point for the creation of the architecture which will, in turn, enable agility throughout the ADM cycle [13].

### 2.2.2 Mapping between Enterprise Architecture and Agile

A mapping between the ADM and the Scrum Agile development methodology [8] is also presented [14, 15]. The established mappings define roles and responsibilities that the Scrum team must oversee, to ensure that the architecture of the project is created and correctly maintained. A committee of owners must ensure governance of the architecture, business, and application sides. The architecture owner must ensure that every team is on the same page regarding the architecture of the solution. The business owner must oversee all the business processes' application and optimization. The application owner is the expert on the application landscape and keeps a backlog of features needed for the application. This structure separates the notions of Architecture, Business, and Application into different backlogs, creating a simple way to store and share information, that aims to support agile aspects such as handling unforeseen change and quickly adapting to it [14].

Another presented approach relates Scrum teams to different phases of the ADM [15]. Figure 2.2 illustrates the distribution of teams throughout the cycle. The Architecture Development Team is responsible for phases A (Architecture Vision), B (Business Architecture), C (Information Systems Architecture), and D (Technology Architecture), as well as the Requirements Management phase. This team is responsible for the creation of the Enterprise Architecture, and to do so, an Architecture Backlog and User Stories are used to ensure that the next sprint is aligned with the goals of the project. The Portfolio Management Team is the second team. They are responsible for phases E (Opportunities and Solutions) and F (Migration Planning), ensuring that the planned architecture changes are coherent, as well as correcting any problems during the merging phase. Finally, the Implementation Team uses pure Scrum to complete the Sprint proposed by the Architecture Development Team, during phase G (Implementation Governance). Using a structure similar to the one presented above can mitigate the high amount of overhead required to create an architecture whilst allowing the project teams to decide on implementation details, resulting in better delivery [15].

## 2.3 Discussion of the Literature

With the analysis provided above, some conclusions can be drawn. The first is that there is evidence that Enterprise Architecture can be deployed in the context of Agile development methodologies. Studies

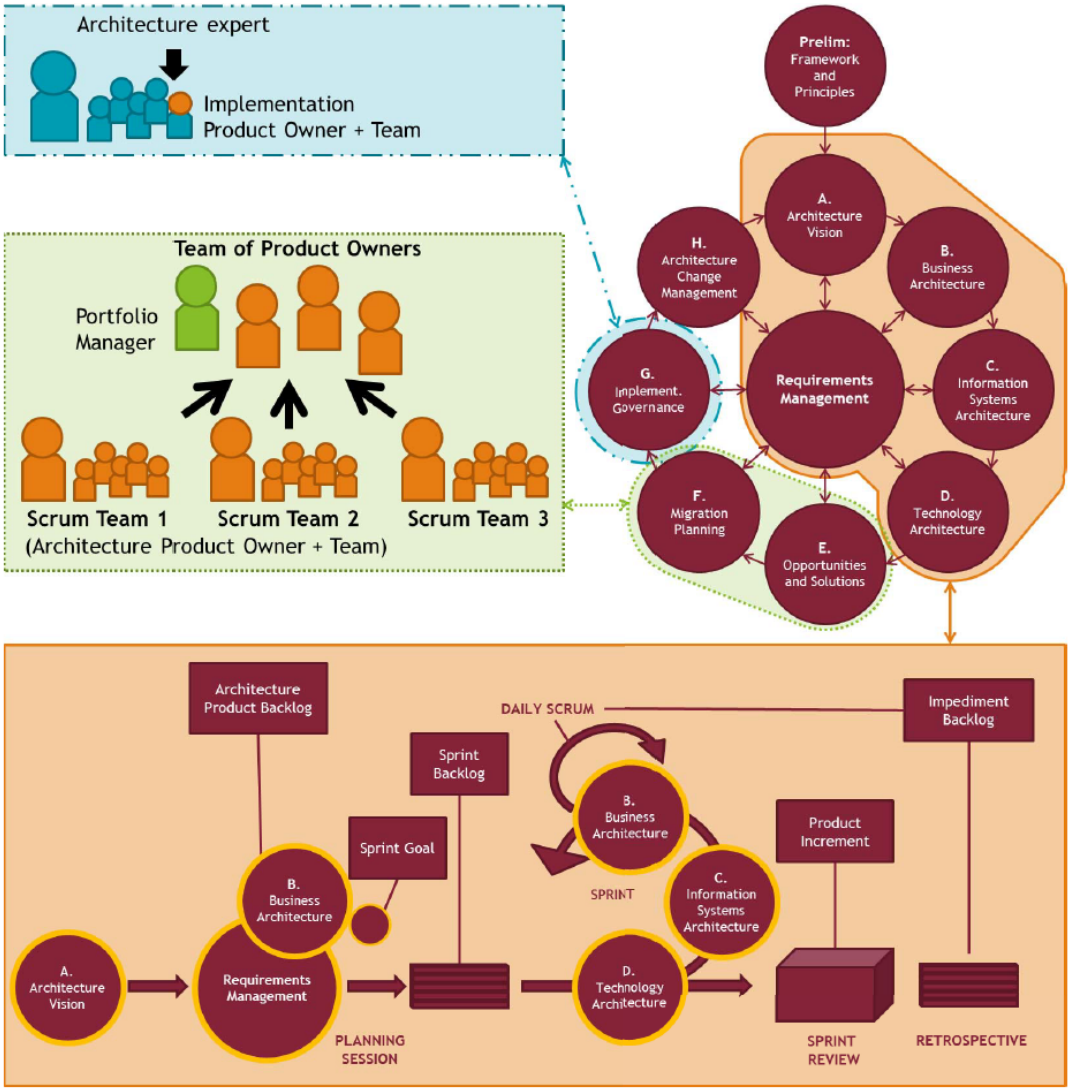


**Figure 2.2:** Hanschke's TOGAF ADM implementation with Scrum

show that doing so may lead to several benefits to both the Agile development team, and the governing body of the organization [10, 11]. This is recognized by the majority of the participants of those studies.

There are several models that propose concrete mappings between Enterprise Architecture and Agile. Regarding TOGAF's ADM, it is possible to utilize agile processes when developing each phase. For instance, during the early phases (Preliminary Phase and Architecture Vision) of the ADM, deploying a streamlined process of requirement definition and restricting the scope of the architecture, will ensure that the rest of the cycle can proceed without wasting time on unneeded objectives [13]. During the later stages of the cycle, it is possible to assign team members responsible for different checkpoints of each phase, such that the architecture that results from ADM is iterative and valuable [14, 15].

However, there are certainly some limitations and important considerations that can dictate the suc-

cess of this implementation. The fact that Enterprise Architecture is still broadly seen as a better fit for Waterfall-based projects alongside the disruptive nature of Agile methodologies (mainly sacrificing documentation and focusing on delivering quality software as fast as possible), poses a challenge for this combination [11]. Additionally, to achieve a satisfactory level of Enterprise Architecture integration, a strict connection between architects and developers is required [10], which will require additional planning and generate some overhead. Another negative aspect, is the fact that some projects may be limited in architectural resources, which can invalidate the possibility of delegating the process of creating and maintaining the Enterprise Architecture to different internal teams, rendering some of the explored models as unusable [14, 15].

Deploying automatic procedures for generating the Enterprise Architecture may mitigate these problems, and this is one of the main focuses of the proposed solution. Automatically generating and updating the architectural assets will reduce the strain placed on the development team, whilst offering a baseline and planned perspective needed by the governing bodies. Furthermore, there is evidence that Enterprise Architecture frameworks such as TOGAF can be utilized to support Agile methodologies. The recommended artifacts, documents, and ADM can be utilized to promote agility while creating an Enterprise Architecture and to focus on the AS-IS and TO-BE aspects of that architecture [13]. On the other hand, TOGAF naturally supports agility by offering a cycle that can be continuously repeated to create the architecture. Phases B (Business Architecture), C (Information Systems Architecture), and D (Technology Architecture) from the ADM can be used to define the scope and detail of the baseline and target architectures. Phases E (Opportunities and Solutions), F (Migration Planning), and G (Implementation Governance) can deploy the target architecture through proper collaboration and governance. Phase H (Architecture Change Management) can be used to ensure that changing requirements are being met and assess their impact [16].

# 3

# Solution Details

## Contents

This Chapter includes the details on how the development of the project took place, as well as its structure. Additionally, some important details of the particular Agile development project that was used as the base to implement this project are also presented.

As an attempt to bridge the gap between Agile development and Enterprise Architecture, a software development project was created. The main goals of this project coincide with the objectives detailed in Section 1.2.

The project was created as an extension to the Atlas Enterprise Cartography Tool [3]. The source code is written in the Java programming language. Additionally, the program utilizes Microsoft's Azure DevOps Representational State Transfer (REST) Application Programming Interface (API) [17] to gather information from DevOps repositories and the Swagger OpenAPI specification files attributed to each created application (more information can be found in Section 3.2). The collected data is then automatically imported into Atlas and architectural representations are created based on the latest information.

## 3.1 Solution Architecture

Figure 3.1 contains an overview of the architecture of the created solution. The architecture representation follows the ArchiMate Enterprise Architecture Modeling Language [18]. The created Atlas Agile Integration solution is integrated into the Atlas Enterprise Cartography Tool.

The Atlas Agile Integration application includes three components:

1. **Azure DevOps Batch Job**: This component is responsible for collecting information from the DevOps environment utilized by the Agile team, and importing it into Atlas. This program can be run periodically to ensure that the latest changes are reflected in the Atlas repository, and automatically, ensuring that the project team does not need to manually import the data.

2. **Swagger File Parser**: This component includes an integration with Swagger. Its main objective is to parse internal documents to gather additional information that can be utilized by the Atlas configuration to better represent the architecture of the Project. Mainly, the information that can be gathered from these files includes dependencies between project modules, and external service utilization.

3. **Atlas Agile Repository Configuration**: This component includes all of the classes, objects, representations, and documents that are generated by Atlas, following TOGAF ADM recommendations. Blueprints, such as the one shown in Figure 1.4, matrices, and charts allow for the visualization of the architecture during a specific timestamp. This component supports the architectural assets of the created Enterprise Architecture.

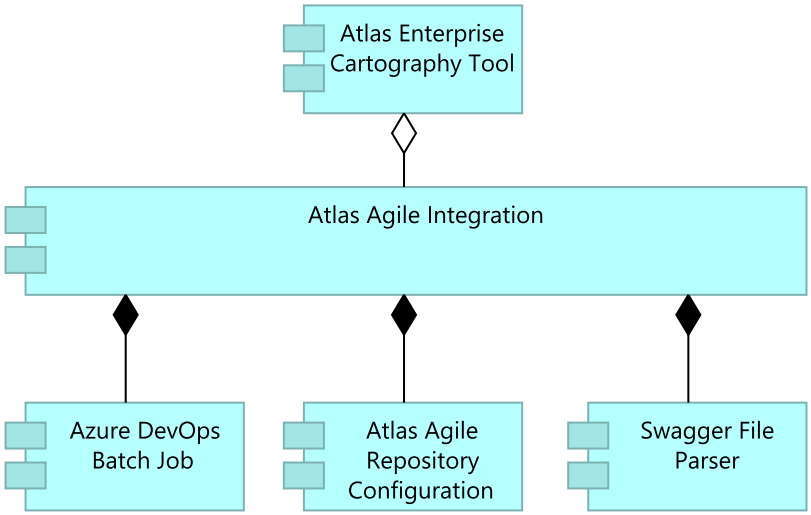


**Figure 3.1:** Solution Architecture

## 3.2 Supporting Agile Project

The goal of the solution presented above is to generate Enterprise Architecture documentation for an Agile development project. This project consists of a digital transformation of a Portuguese insurance provider. The result includes several application modules, featuring back and front-office features.

The project team chose to utilize Microsoft's Azure DevOps [17] environment to track the project's progress. The DevOps environment includes information regarding backlog management, sprint planning, and client acceptance. This information is used by Atlas to represent the architecture of the project.

When planning a sprint, the project team estimates and decides what Product Backlog Items (PBIs) must be completed before the deadline. These are added to the Sprint Backlog, and are then assigned to the project team's members, to be worked on during the development sprint. Each sprint has a duration of two weeks and it is concluded in a sprint closing meeting, where the progress is assessed and the next sprint is planned.

Azure DevOps' work items are used to represent the details of the project. Regarding the structure of the information, Figure 3.2 contains an overview of the different Work Items that are in use. Each of the main modules of the project is represented as an Epic. The Epics aggregate a set of Features. To realize the Features, each one includes a set of PBIs. Finally, each PBI is divided into a set of Tasks that are worked on by the members of the development team.

Each Work Item includes a list of states that are transitioned to and from, as the Work Item is managed and worked upon. For instance, the PBI Work Items start in the *New* state. When a team member starts working on it, the item will be set to the *In Progress* state. When the item is completed by the assignee, it is moved to the *Committed* state, where it awaits approval. If the PBI is approved, the state

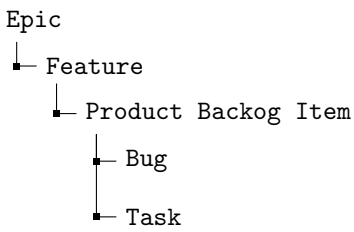


**Figure 3.2:** Azure DevOps Work Item Structure

will be moved accordingly to *In Progress*, otherwise a discussion around it is started and its state may return to *In Progress* to accommodate the new changes. The *Done* state is applied after the Sprint is concluded. The Epic and Feature Work Items may have the *New*, *In Progress*, and *Done* states, depending on the state of their child Work Items. An Epic is considered *Done* after all of its Feature Work Items have the *Done* state. By the same token, a Feature is considered *Done* when all of its PBI Work Items have the *Done* state.

A mapping between these Agile concepts and the Atlas architectural concepts is provided in Table 3.1. The development team establishes each Epic as a different Application required to complete the project. Under each Application, the team defines and aggregates the Goals and Requirements needed to successfully implement each one. The Requirements are created through discussion with the client and further divided into simpler Requirements that are assigned to a single member of the development team to be completed during the Development Sprints. An Application is considered to be complete after every Requirement has been addressed and accepted by the client. Additionally, new Requirements may arise at any point of development, if deemed necessary by the client, and the Requirements may be rejected by the client if their needs are not met. This leads to a new development of that Requirement, before it is evaluated by the client again.

The development of the software solution is done using several tools, such as the .NET development framework and the Swagger toolset, for configuration and dependency management. Each software module has two or more executables, generally a back-office and an Angular front-office. These applications communicate through REST services and the code is deployed into different environments, namely, production, development, testing, and quality assurance, depending on the state of development.

Regarding the difficulties of the development team, three main pain points were identified. The first is a lack of a simple way to verify the progress of the project, namely to check on what Requirements have been completed, and how close to completion a software module is at a certain point in time. The second is the lack of information regarding the dependencies between the software modules, as currently there are no mechanisms that show the connections between modules and internal and external services. The last identified issue is the high amount of manual effort required to create the documentation that

| Work Item | Architectural Mapping |
|---|---|
| Epic | Application |
| Feature | Goal |
| Product Backlog Item | Requirement |

**Table 3.1:** Mapping between Azure DevOps Work Items and Atlas Architectural Assets

the client requires to check the evolution of the project.

## 3.3 Solution Implementation

After establishing the details and constraints of the supporting Agile development project, the Atlas Agile Integration Solution was implemented. This Section contains the details of each module. Chapter 4 contains an overview of the results that were produced by this Solution, mainly focusing on the resulting architectural assets and their impact.

### 3.3.1 Preparation Phase

Section 3.2 introduced the details of the Agile project that was used as a guide to develop this Solution. This Section highlights the development and implementation of the Atlas Agile Integration solution, represented in Figure 3.1.

As a starting point, information was collected from the development team regarding the methods and technology already in use. This was an essential step that aided in the definition of the mechanisms that could be explored and implemented to support the creation of automated Enterprise Architecture assets. The information gathered from this stage is present in Section 3.2.

After gathering the information that was needed to establish the objectives, the next step was to get acquainted with the Atlas Enterprise Cartography Tool (analysed in Section 1.1.3). This analysis, in conjunction with the findings regarding the needs of the project team and the used tools and methodologies, shaped the strategy and implementation of the solution.

Concerning the tools that are used by the development team, the one that was highlighted as the most relevant, was Microsoft's Azure DevOps, as this was the main platform used by both the internal team and the client to track the ongoing and upcoming work and to accept or reject any changes made during development, acting as the major information repository of the project. Furthermore, this environment also contains the links to the deployment of each Application, where the Swagger OpenAPI configuration is stored. This file includes information regarding the REST calls made by each application.

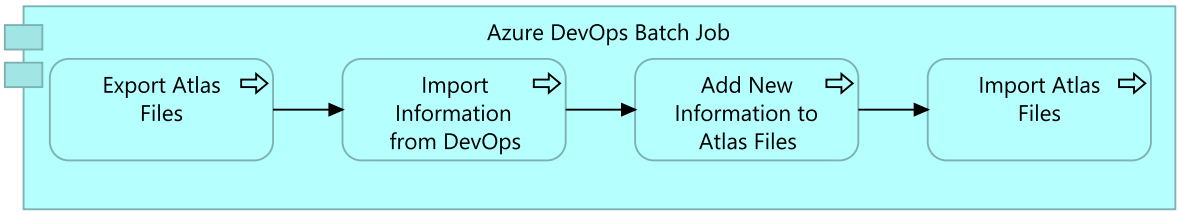


**Figure 3.3:** Atlas Batch Job Process

### 3.3.2 Azure DevOps Batch Job Implementation

With the previous constraints in mind, some important implementation decisions were made. To collect the information from the DevOps environment, an Atlas Execution Batch Job was created. This option allows for the use of Java code directly, which in turn enables the usage of REST APIs, such as Azure DevOps' API [17]. Additionally, a Batch Job can be scheduled to run in certain time intervals, automating the importation of the updated information. The process of the created Batch Job is displayed in Figure 3.3. The first step is to collect the Work Items that are already present within the Atlas repository. The Work Items are exported in the Atlas XML file format and these files are later edited with the new information that has been collected. Afterward, the DevOps Work Items are requested by the Java application, through the REST API. To minimize overhead, only new Work Items or Work Items that have been modified in the last three days are requested. Properties from these Work Items, such as the name, state, and completion dates, are collected and parsed into the Atlas XML file format. These new properties are appended to the exported Atlas XML files, and finally they are imported back into the Atlas repository. This ensures that any new change performed by a team member in the DevOps environment is propagated into Atlas. Furthermore, this Atlas Batch Job is configured to run on a daily basis, ensuring that the latest information is always available in the Atlas repository.

The development of the Batch Job was aided by Eclipse's Integrated Development Environment (IDE) software. The IDE was used alongside a private Git repository, created through GitHub, to enable a better version control throughout the development process. The Git repository was also used during the development of the Swagger File Parser module, paired with Microsoft's Visual Studio Code and the Python programming language.

To collect the Work Items from the DevOps repository, the first step was to generate a Personal Access Token (PAT) from the DevOps environment, such that the requests through the Azure REST API were authenticated, and the reading permissions were granted. Afterward, to query the environment for the needed Work Items, queries were created using Azure's Work Item Query Language. Three queries were created, one for the Epic Work Items, another for the Feature Work Items, and the final one for the PBI Work Items. The queries are shown in Listing 3.1. These queries return the Work Items that have been modified in the last three days, and different properties are collected from different Work Item

types. From the Epic Work Items, the name, status, identifier, creation date, and the team member that created it are collected. From the Feature Work Items, all of the previous properties are collected, as well as the Epic Work Item that aggregates it. From the PBI Work Items, the description and acceptance criteria of each one is also collected.

**Listing 3.1:** Azure Work Item Queries

```
1 epicQueryWiql = SELECT [System.Title],[System.Id],[System.State],[
      System.WorkItemType],[System.CreatedBy],[System.CreatedDate] FROM WorkItems
      WHERE [System.TeamProject] = @project AND [System.WorkItemType] = 'Epic' AND
      [System.ChangedDate] > @startOfDay('-3d') ORDER BY [System.State], [System.Id
      ]
2 featureQueryWiql = SELECT [System.Title],[System.Id],[System.State],[
      System.Parent],[System.WorkItemType],[System.CreatedBy],[System.CreatedDate]
      FROM WorkItems WHERE [System.TeamProject] = @project AND [System.WorkItemType
      ] = 'Feature' AND [System.ChangedDate] > @startOfDay('-3d') ORDER BY [
      System.State],[System.Id]
3 pbiQueryWiql = SELECT [System.Title],[System.Id],[System.Parent],[System.State],[
      System.WorkItemType],[System.Description],[
      Microsoft.VSTS.Common.AcceptanceCriteria],[System.CreatedBy],[
      System.CreatedDate] FROM WorkItems WHERE [System.TeamProject] = @project AND
      [System.WorkItemType] = 'Product Backlog Item' AND [System.ChangedDate] >
      @startOfDay('-3d') ORDER BY [System.State]
```

These queries are then used in conjunction with the PAT and the `azure-devops-java-sdk` module [19] to gather and store all of the information in XML Files that are then imported into Atlas. The XML files follow the Atlas XML standard (shown in Listing 3.2). The document represents the properties and objects of a single class. The `DocumentElement` tag contains `PropsAndValues` tags. Each `PropsAndValues` represents the Properties of a single Object. For instance, Listing 3.2 represents the Goal class, the *Claim Data Integration* and *Document Access Control* objects and their respective property values, as well as a configuration tag that shows all of the properties that are valid for the Goal class.

**Listing 3.2:** Atlas XML File Example - Goal Class

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <DocumentElement>
3     <PropsAndValues>
```

```
4           <Name># CONFIGURATION_NODE # DO NOT DELETE THIS ROW #</Name>
5           <Decommission_x0020_Date>#</Decommission_x0020_Date>
6           <Description>#</Description>
7           <Owner>#</Owner>
8           <Productive_x0020_Date>#</Productive_x0020_Date>
9           <Strategic_x0020_Domain>#</Strategic_x0020_Domain>
10      </PropsAndValues>
11      <PropsAndValues>
12          <Name>Claim Data Integration</Name>
13          <Decommission_x0020_Date>31-12-2022</Decommission_x0020_Date>
14          <Description>The Claims received from the current platform must be
                completely integrated.</Description>
15          <Owner>Alice Clark</Owner>
16          <Productive_x0020_Date>07-02-2022</Productive_x0020_Date>
17          <Strategic_x0020_Domain>Claim</Strategic_x0020_Domain>
18      </PropsAndValues>
19      <PropsAndValues>
20          <Name>Document Access Control</Name>
21          <Decommission_x0020_Date>28-02-2023</Decommission_x0020_Date>
22          <Description>The Documents received from the different platforms must be
                accessible.</Description>
23          <Owner>Bob Patel</Owner>
24          <Productive_x0020_Date>04-07-2022</Productive_x0020_Date>
25          <Strategic_x0020_Domain>Document</Strategic_x0020_Domain>
26      </PropsAndValues>
27  </DocumentElement>
```

### 3.3.3 Swagger File Parser Implementation

To further enrich the information gathered from the development of the project, a Swagger File Parser was created. The process is described in Figure 3.4. The project team utilizes the OpenAPI Specification to specify the API calls performed by each Application. Each Application includes a Swagger file, available in the `/swagger` path of their deployment URL. The Swagger files are JavaScript Object Notation (JSON) specifications that include the REST calls of each Application, listed under the `/api/api_name` path. This offers the possibility of retrieving information regarding the dependencies that exist between the created Applications. Listing 3.3 shows an example of a Swagger file. The `paths` list includes the API calls performed to other services, namely to the Extranet API and to the Subscription

API.

**Listing 3.3:** Swagger JSON File Example - Extranet

```
{
    "x-generator": "NSwag v13.16.1.0 (NJsonSchema v10.7.2.0 (Newtonsoft.Json v13.
        0.0.0))",
    "openapi": "3.0.0",
    "info": {
      "title": "Online Web",
      "description": "Online Web API",
      "version": "v1"
    },
    "servers": [
      {
        "url": "https://online_web_address.pt"
      }
    ],
    "paths": {
      "/api/extranet/authenticate": {
        "post": {...}
      },
      "/api/subscription/search-subscription": {
        "post": {...}
      }
    }
}
```

To automatically collect that information, a Python script was developed. The script parses the JSON files that contain the API calls, collecting the names of the APIs that are present within that file. These names are then translated into the corresponding Application names, through a name alias file. The final step of the process is to create the Atlas importation file that contains the dependencies of each Application. This is achieved with the creation of an Excel file that follows the Atlas standard, with the services that are requested by each Application, namely having a column with the name of the object, followed by a column with every property that must be imported. Finally, this Excel file can be imported into Atlas to update the information.

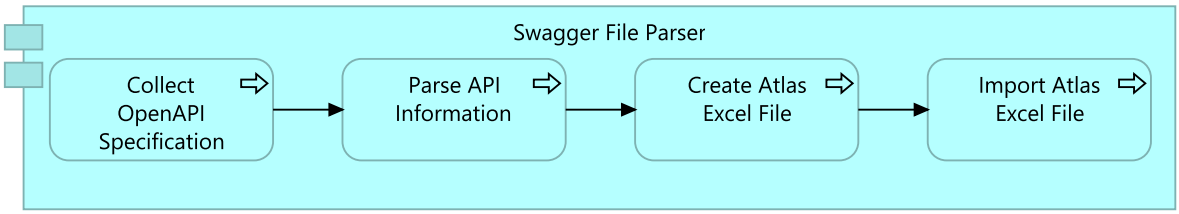


**Figure 3.4:** Swagger File Parser Process

### 3.3.4 Atlas Agile Repository Configuration

After establishing a process to automatically gather information from the DevOps environment (Figure 3.3) and to collect the dependencies between the applications (Figure 3.4), the next step was to prepare the configuration of the Atlas repository. The main focus of the configuration is to utilize the collected data to generate architectural representations that are useful and easily maintained. Representations (such as the one depicted in Figure 1.4) can be used to dynamically display the latest information gathered from the Azure DevOps Batch and Swagger File Parser. On top of that, the TOGAF ADM recommended outputs and artifacts, were used as a guide when developing the configuration for the Atlas repository.

The utilized approach was to first define a set of basic representations for each of the Work Items, representing their important properties and hierarchy. For instance, a blueprint representing every Feature Work Item, and consequently every Goal that is part of the project's target solution, was created. Using the status of the DevOps work item, Atlas can show a timeline, allowing for a quick overview on which Applications have been created and which of them are yet to be completed on a certain date. Additionally, blueprints representing every Goal and Requirement that are aggregated by a specific Application were also created, enabling a big picture view of the progress status of each Application. Afterward, one context blueprint was created for each of the Work Items, representing the status of each one, as well as other information that could be gathered, such as the team member in charge of developing each item.

The focus then shifted to the analysis of the ADM's suggested documents, to create similar representations within Atlas. Throughout the cycle, several Architectural Artifacts [20] are proposed as supporters of the architecture and Architectural Deliverables [21] evolve throughout the different phases of the ADM. Section 3.3.5 contains an analysis of the more relevant deliverables within the scope of this Solution and Section 3.3.6 contains information regarding the creation of the several representations.

### 3.3.5 TOGAF ADM Artifact Analysis

This section contains a list of important TOGAF ADM Deliverables [21]. These documents are sorted by the phase in which they are created or updated, and contribute to the creation and of the project's

Enterprise Architecture. Furthermore, the focus of this Section is to also highlight which documents can enrich the architecture of the project and to guide the creation of these assets within Atlas. Additional information regarding each TOGAF ADM Phase can be found in Section 1.1.2.

#### 3.3.5.A Preliminary Phase

The outputs of this phase are intended as guiding documents, that will aid in the development of the Enterprise Architecture of the project. The documents relevant to this phase are mostly created before the start of the project.

**Architecture Repository:** This output includes all the architecture-related assets that are relevant to the project. During the ADM cycle, this Repository must be updated with new documentation and models that are created in each phase. The Architecture Repository will be supported through the Atlas repository configuration. Using it, it is possible to store and update the created architectural representations, achieving a complete view of the architecture.

#### 3.3.5.B Phase A (Architecture Vision)

The outputs of Phase A include the initial planning of the cycle. Aspects such as scope, principles, goals, and drivers of the project are defined during this phase and changed as needed throughout the cycle.

**Statement of Architecture Work:** This document contains roles, responsibilities, and deliverables used throughout the ADM, coupled with the architectural scope and approach definitions. Furthermore, it must include a way to measure the successful execution of the architectural project. The success criteria and User Stories provided by the client are used to complement it.

**Architecture Principles:** This output defines a standard format to specify principles that concern the architecture of the project. Usually, a Principle is defined in a table that contains the name of the rule, the rationale behind the creation, and the implications that the requirement will have on the architecture. To maintain and share this information in a timely manner, tables that allow each member to create Goals and Requirements are supported. Additionally, these representations include all the Goals and Requirements extracted from the DevOps environment.

#### 3.3.5.C Phases B, C, and D (Business, Information Systems, and Technology Architecture)

Phases B, C, and D output similar documents, albeit with different scopes. Phase B focuses on the definition of Business-related requirements, goals, constraints, and scheduling. Similarly, Phases C and D focus on the definition of Information-related and Technological-related constraints, respectively.

**Refinements to the Architecture Vision:** This document includes a problem description, summary diagrams, and requirements for the Business, Data, Application, and Technology scopes. It is constantly evolving throughout the ADM, making it an important automation target when reducing the manual overhead required to maintain the architecture up-to-date. Utilizing the information from the DevOps environment and from the Swagger tool, models were created for the different scopes (Business, Information Systems, and Technology). This document is heavily tied to the Architecture Definition Document.

**Architecture Definition Document:** This document includes architectural artifacts and assets created during the project's development. They relate to the three scopes of phases B, C, and D, and include artifacts such as Principles, models, and a Transition Architecture. To automate the maintenance of these assets, Atlas representations are used. This ensures that every artifact is stored and that information evolves as the project is developed and changes are made.

**Architecture Roadmap:** This document includes scheduling of work packages, divided by scope, shown in a timeline with completion status and dates for each package. This is achieved through the use of lifecycles in Atlas, meaning that each object includes a set of dates which inform the past, present, and planned state changes.

#### 3.3.5.D Phases E, F, G, and H (Opportunities and Solutions, Migration Planning, Implementation Governance, and Architecture Change Management)

These phases are tied with several aspects that are not included in the primary scope of this study, namely, they mainly modify the already created documents, with information regarding risk and change assessment. Hence, they are not discussed further.

### 3.3.6 Architectural Representations Creation

The three main representations utilized to support the Enterprise Architecture are tabular views, matrices, and blueprints. This Section includes details on how to create those representations.

#### 3.3.6.A Tabular View Representation

The tabular view representations are available for every class in the Atlas repository. They show every object of that class, alongside chosen relevant attributes. To create a tabular view, one needs to edit a class through the Data Explorer tab in Atlas. The Tabular View Configuration category allows users to specify the properties to be displayed in the table through a drag-and-drop interface. Figure 3.5 shows an example of a Business Actor tabular view showing the actors, their e-mails, and their business phone numbers. Several tabular views are shown in Chapter 4.

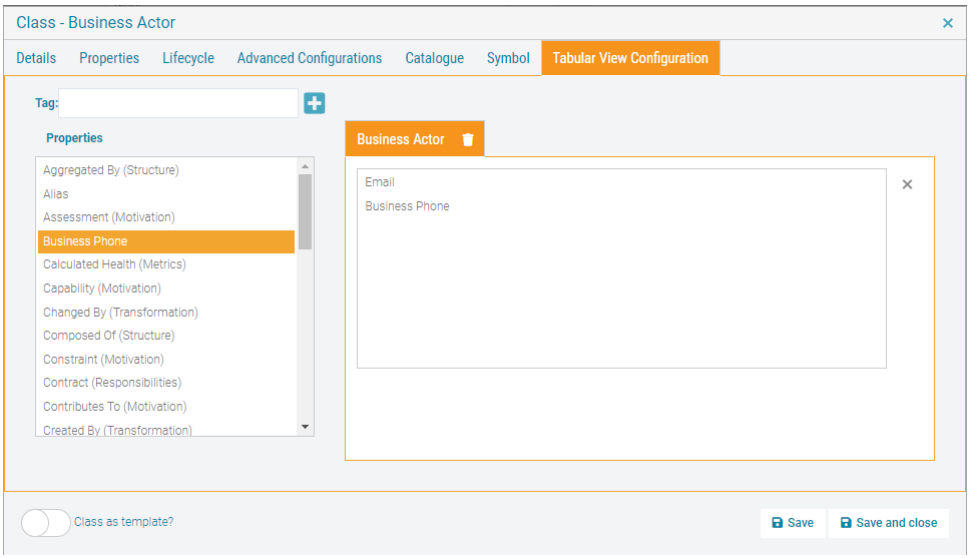


**Figure 3.5:** Creation of a Business Actor Tabular View

#### 3.3.6.B Matrix Representation

The matrix representations are defined through Atlas' View Explorer Tab. To create a matrix representation, one needs to create a view of type MatrixCanvas. On the Configuration window, the classes to be displayed on the rows and columns can be picked, as well as the property that will determine whether or not the matrix intersection is filled. Figure 3.6 shows the configuration for a matrix that shows the dependencies between Applications. If an Application represented in the rows of the matrix interacts with an Application represented in the columns through the Consolidation property, the intersection cell will be painted blue. It is possible to add and remove relations directly through the matrix. Examples of matrices are shown in Chapter 4.

#### 3.3.6.C Blueprint Representation

The blueprint representations are also defined in Atlas' View Explorer Tab. They can be created through a visual designer that supports containers, variables, and queries. An example of the configuration of a blueprint can be seen in Figure 1.4. The designer for this particular representation includes three containers (Node, Operating System, and Application Components). The Node container shows the argument Node that is used to access this representation. The other two containers show the result of the `operating_system` and `applications` variables. The variables are the result of queries that are defined in the right-side of the Figure, by querying the argument Node's relations. For instance, the `applications` variable results from querying the *deploys* relation of the argument Node and then filtering the results to show only Application Component objects. The same approach was used to define the `operating_system` query with the differences being the used relation (*behaves as*) and the filtered

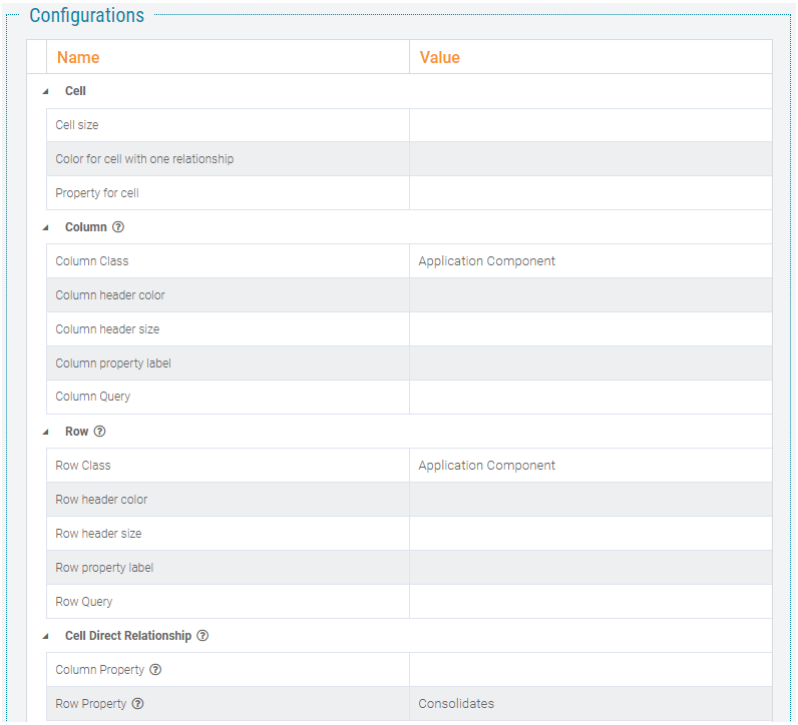


**Figure 3.6:** Application Matrix View Configuration

objects' class (System Software). The resulting blueprint is shown in Figure 4.6 and other examples of blueprints can be found in Chapter 4.

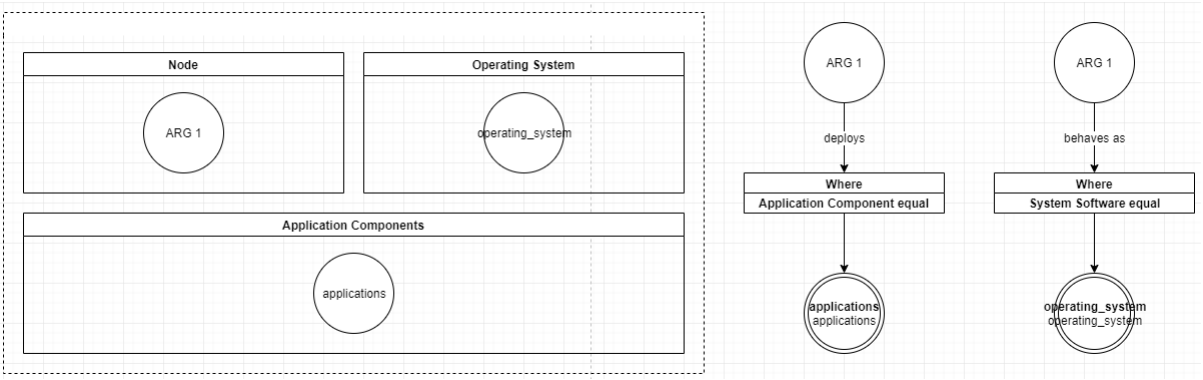


**Figure 3.7:** Node Context Blueprint Configuration

# 4

# Results and Analysis

## Contents

This Chapter includes an overview of the created representations and architecture, the mapping with the TOGAF ADM, and a summary of the observed results related to the deployment of the solution in the context of the Agile Development project. It also contains information regarding the evolution of the representations, to maintain an updated view of the project throughout its development, and assessment, focusing on how the solution meets the objectives identified in Section 1.2.

## 4.1 Components of the Architecture

This Section contains an overview of the created Enterprise Architecture. Figure 4.1 shows the layers and components utilized to represent the architecture of the project. The representations are divided into several categories following the TOGAF standard, namely Architecture Vision, Business Architecture, Application Architecture, Data Architecture, and Technology Architecture. Regarding the layers of the presented architecture, the Architecture Vision layer includes the Goal and Requirement classes, which are mainly connected to the needs of the client. These Goals and Requirements are owned by Business Actors in the Business Architecture layer that oversee their completion and approval. The Architecture Vision artifacts also directly influence the Application Architecture layer, given that each Goal and Requirement is established with a certain Application Component in mind. Additionally, each Application Component may provide and consume Application Services. Finally, the Technology Application layer includes Nodes and System Software that supports the components of the Application Architecture layer.

After establishing the architecture layers, the representations were created. The types of representations used are detailed in Table 4.1. Catalogs are mainly used as a listing of the architectural objects. Each Catalog is configured to display the most relevant properties of each object, and this can be easily edited at any time by a member of the project team, to complement the information. Maps are used to relate two classes based on a relevant property, and to also allow for quick updating of information (Figure 1.5). Finally, Blueprints are used to overview important objects, and relevant properties and relations, in specific contexts (Figure 1.4). The next Sections will highlight the representations that are used to represent the Enterprise Architecture of the project, sorted by the architecture layer that categorizes each one.

### 4.1.1 Architecture Vision Layer

The first layer of the proposed architecture is the Architecture Vision. Similarly to the TOGAF standard, this layer includes the motivation for the architecture, mainly focusing on the goals and requirements that are established for the project. This layer includes the Goal and Requirement classes that are extracted from the project team's DevOps environment. The dates and the actor responsible for their creation are

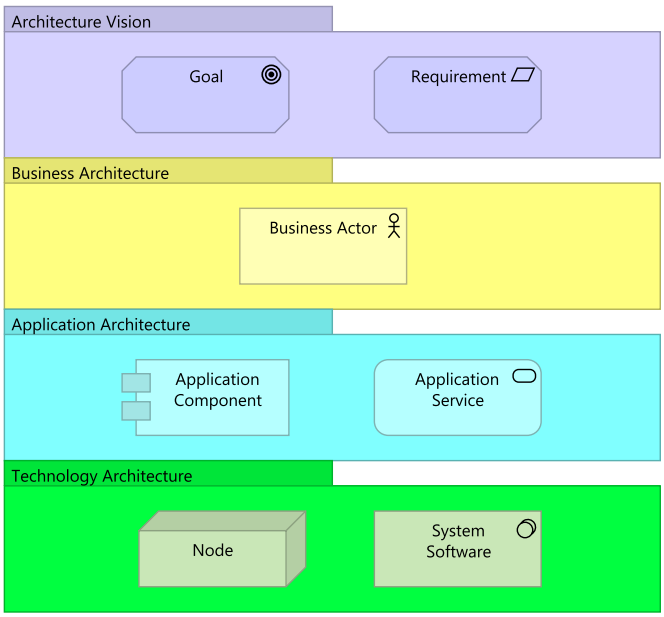


**Figure 4.1:** Layers and components of the created Enterprise Architecture

| Representation Type | Description |
|---|---|
| Catalog | Tabular view showing every object of a certain class and the most relevant properties |
| Map | Matrix view showing a relation between the column class and the row class |
| Blueprint | Visual representation showing objects and relevant relations, according to the scope |

**Table 4.1:** Description of the main architectural Representation Types

used to complement the information.

The main representations of this layer are the Goal and Requirement Catalogs, and the Goal Organic Blueprint, shown below.

#### 4.1.1.A Goal Catalog

The Goal Catalog is a tabular representation of every Goal that is currently loaded into the Atlas repository. These can be either collected from the Azure DevOps integration (Section 3.3.2) or manually created as needed. This Catalog is shown in Table 4.2, where the Goals are shown next to their Domain, Owner, Description, Productive Date, and planned Decommission Date. These properties can be updated as needed, both manually and automatically.

New Goals can be manually added to this table, and every new Goal collected from the Azure DevOps Batch will appear here.

| Name | Strategic Domain | Owner | Description | Productive Date | Decommission Date |
|---|---|---|---|---|---|
| Claim Data Integration | Claim | Alice Clark | The Claims received from the current platform must be completely integrated. | 07-02-2022 | 31-12-2022 |
| Client Area | Client | Mark Evans | A new Client Area that shows the past and current Claims must be created. | 27-06-2022 | 31-12-2022 |
| Customer Support | Client | Mark Evans | A Customer Support feature must be integrated into the new solution, with a Chat Bot. | 18-04-2022 | 01-06-2023 |
| Document Access Control | Document | Bob Patel | The Documents received from the different platforms must be accessible. | 04-07-2022 | 28-02-2023 |
| Secure Sign-In | Security | John Irwin | Sign-in must be performed through secure protocols. | 21-02-2022 | 19-05-2023 |

**Table 4.2:** The Goal Catalog Representation

| Name | Strategic Domain | Owner | Contributes To | Influences | Productive Date | Decommission Date |
|---|---|---|---|---|---|---|
| Configure Claim Data | Claim | Alice Clark | Claim Data Integration | Claim Management Application | 07-02-2022 | 31-12-2022 |
| Create Document Templates | Document | Bob Patel | Document Access Control | Product Management Application<br>Intranet Application | 04-07-2022 | 28-02-2023 |
| Design Client Area | Client | Mark Evans | Client Area | Extranet Client Application<br>Extranet Application | 15-07-2022 | 01-06-2023 |
| Implement Secure Protocols | Security | John Irwin | Secure Sign-In | Intranet Application | 21-02-2022 | 19-05-2023 |
| Review Data Structure | Document | Bob Patel | Claim Data Integration | Product Management Application<br>User Management Application<br>Claim Management Application | 14-07-2022 | 31-12-2022 |

**Table 4.3:** The Requirement Catalog Representation

#### 4.1.1.B Requirement Catalog

Similarly to the Goal Catalog, a Requirement Catalog representation is also provided (Table 4.3). This representation shows the Requirements that are currently loaded into the Atlas repository. The Domain, Owner, and Productive and Decommission Dates properties are shown, as well as the Goal that aggregates this Requirement (under the *Contributes* To column) and the Applications that are influenced by this Requirement (under the *Influences* column).

New Requirements can be manually added to this table, and every new Requirement collected from the Azure DevOps Batch will appear here.

#### 4.1.1.C Goal Organic Blueprint

The Goal Organic Blueprint lists every Goal under its Domain, offering a global view. Figure 4.2 includes this Blueprint. Here, the different Domains of Claim, Client, Document, and Security are shown, alongside the Goals that have been established for each of them.

Every time a new Domain is created or a new Goal is added to any of the already existing Domains, this representation will be automatically updated.

### 4.1.2 Business Architecture Layer

To support the Business level of the architecture, the Requirements and Goals utilized in the Architecture Vision phase (Section 4.1.1) were combined with the information collected from the DevOps environment regarding the project team members that are assigned to each PBI, and consequently, assigned to each Goal, Requirement, and Application.

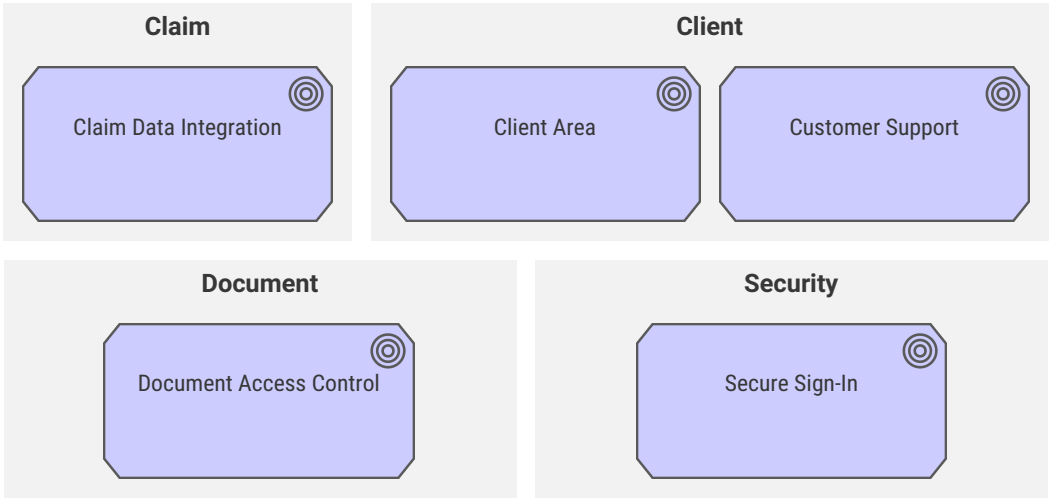


**Figure 4.2:** The Goal Organic Blueprint

The main representation of this layer is the Business Actor Context Blueprint, shown below.

#### 4.1.2.A Business Actor Context Blueprint

Figure 4.3 represents the Business Actor Context Blueprint. This representation shows the architectural assets that are owned by a Business Actor, i.e. the assets that are impacted by a specific actor in some way. The Figure shows the example of Business Actor `Mark Evans`. The assets that are impacted by this actor, are shown below. Namely, `Mark Evans` directly impacts the *Client Area* and *Customer Support* Goals, the *Design Client Area* Requirement, and the *Rule Engine Application* Application.

### 4.1.3 Application Architecture Layer

Regarding the Application Architecture Layer, the main components of this layer are the Application Component and the Application Service classes. They represent the required software to be developed and the communication interfaces established by them, respectively. The information for this layer is mainly obtained from the DevOps environment Epics, as these map to the Application Components to be developed, and from the Swagger Files representing the communication performed among the different Application Components. This information can also be manually enhanced as needed.

The main representations of this layer are the Application Component and Application Service Catalogs, the Application Integration Blueprint, and the Application Interaction Map, shown below.

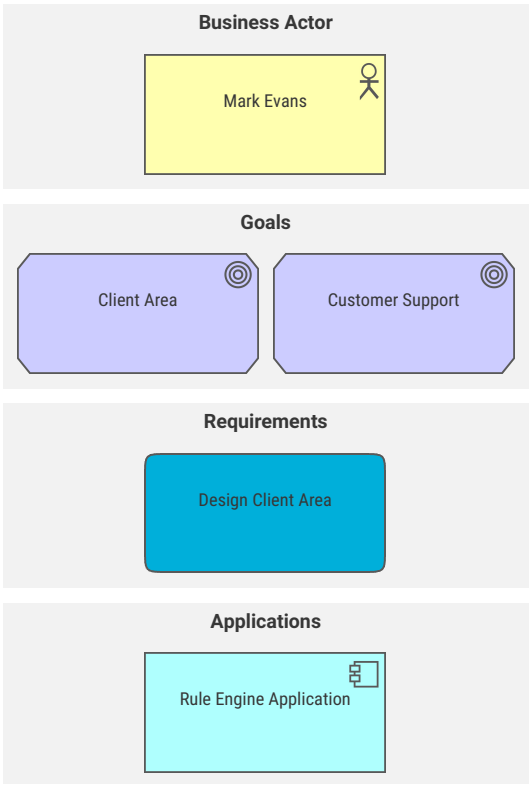


**Figure 4.3:** The Business Actor Context Blueprint

#### 4.1.3.A Application Component Catalog

The Application Component Catalog representation is shown in Table 4.4. It includes the Applications that are planned to be developed during the project. It shows the planned Under Implementation dates of each Application, as well as the Domain, Owner, and planned Productive and Decommission dates.

New Application Components can be manually added to this table, and every new Application Component collected from the Azure DevOps Batch will appear here.

| Name | Organic Domain | Owner | Under Implementation Date | Productive Date | Decommission Date |
|---|---|---|---|---|---|
| Claim Management Application | Management | Alice Clark | 07-02-2022 | 07-03-2022 | 31-12-2023 |
| Commercial Application | Online | Bob Patel | 04-07-2022 | 15-09-2022 | 31-12-2023 |
| Extranet Application | External | Sophia Miller | 25-07-2022 | 31-10-2022 | 31-12-2023 |
| Extranet Client Application | External | Sophia Miller | 25-07-2022 | 31-10-2022 | 31-12-2023 |
| Extranet Mediator Application | External | Sophia Miller | 25-08-2022 | 31-10-2022 | 31-12-2023 |
| Extranet Partner Application | External | Sophia Miller | 25-08-2022 | 31-10-2022 | 31-12-2023 |
| Intranet Application | Internal | John Irwin | 19-05-2022 | 26-09-2022 | 31-12-2023 |
| KeyCloak Application | Security | John Irwin | 19-05-2022 | 26-09-2022 | 31-12-2023 |
| Management Web Application | Management | Alice Clark | 16-05-2022 | 16-06-2022 | 31-12-2023 |
| Online Web Application | Online | Bob Patel | 08-08-2022 | 30-09-2022 | 31-12-2023 |
| Product Management Application | Management | Alice Clark | 14-11-2022 | 08-02-2023 | 31-12-2023 |
| Rule Engine Application | Management | Mark Evans | 27-06-2022 | 10-11-2022 | 31-12-2023 |
| User Management Application | Management | Alice Clark | 12-09-2022 | 16-12-2022 | 31-12-2023 |

**Table 4.4:** The Application Component Catalog Representation

| Name | Application Usage | Productive Date | Decommission Date |
|---|---|---|---|
| Commercial Application Service | KeyCloak Application | 15-09-2022 | 31-12-2023 |
| Extranet Application Service | KeyCloak Application<br>Commercial Application | 31-10-2022 | 31-12-2023 |
| Extranet Application Subscription Service | Product Management Application<br>User Management Application | 21-11-2022 | 31-12-2023 |
| Keycoak Application Service | | 26-09-2022 | 31-12-2023 |
| Management Web Application Service | Intranet Application | 16-06-2022 | 31-12-2023 |
| Online Web Application Service | Online Web Application | 30-09-2022 | 31-12-2023 |
| Product Management Application Service | Rule Engine Application | 08-02-2023 | 31-12-2023 |
| Rule Engine Application Service | Intranet Application | 10-11-2022 | 31-12-2023 |
| User Management Application Service | Extranet Client Application | 16-12-2022 | 31-12-2023 |

**Table 4.5:** The Application Service Catalog Representation

#### 4.1.3.B Application Service Catalog

The Application Service Catalog representation is shown in Table 4.5. It represents all of the Application Services that are provided by the Application Components to allow for different Applications to communicate and exchange information. Aside from the planned Productive and Decommission Dates of each Service, the Applications that use the respective Services are specified (Application Usage column).

New Application Services can be manually added to this table, and every new Application Service collected from the Swagger File Parser will appear here.

#### 4.1.3.C Application Integration Blueprint

The Application Integration Blueprint is represented in Figure 4.4. It shows the connections between different Applications, both requested and provided, as well as the Application Services utilized to do so, for each Application. This representation will be updated automatically as new Applications and Application Services are created and new relations are established between them.

In Figure 4.4, the *Extranet Application* requests the services that are on its left (*Commercial Application Service*, *Online Web Application Service*, etc.). These services are provided by the Applications represented on the left-most column (*Commercial Application* realizes *Commercial Application Service*, *Online Web Application* realizes *Online Web Application Service*, etc.). The *Extranet Application* also provides the services on the right (*Extranet Application Service* and *Extranet Application Subscription Service*) to the *Online Web Application*.

#### 4.1.3.D Application Interaction Map

The Application Interaction Map is a matrix-style representation, that shows an overview of every Application that interacts with other Applications. Figure 4.5 represents all of the Applications that are loaded into the Atlas repository. A blue square is shown on the intersection if an Application represented in the rows of the matrix requests information from an Application represented in the columns of the matrix.

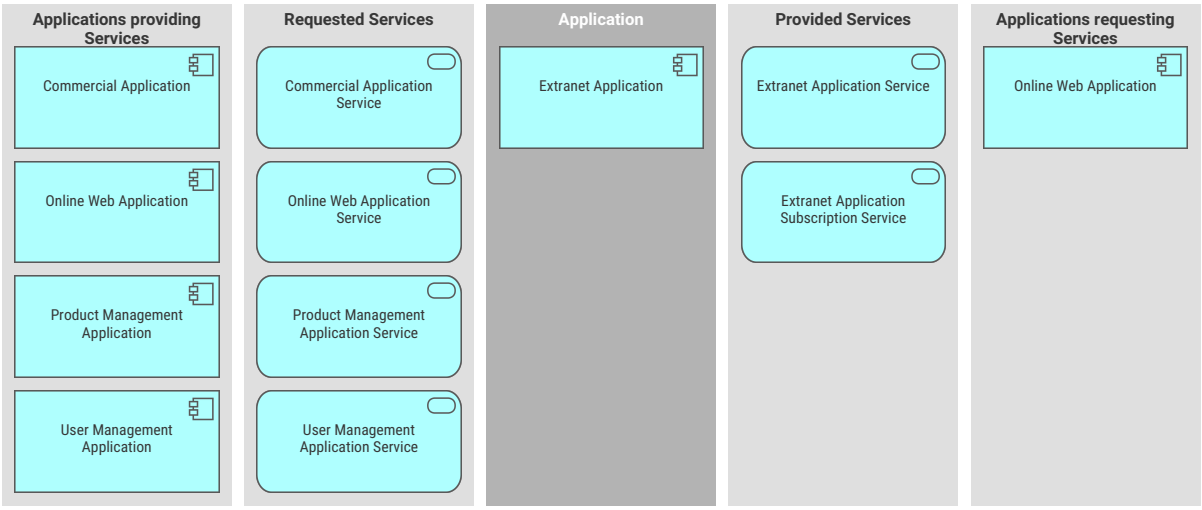


**Figure 4.4:** The Application Integration Blueprint

For instance, the *Claim Management Application*, the *Management Web Application*, and the *Product Management Application* request information from the *Online Web Application*.

### 4.1.4 Technology Architecture Layer

The Technology Architecture Layer features the infrastructure that supports the Application, Business, and Vision landscapes. The main components of this layer include the Nodes where the Applications are executed. The System Software represents software that supports the use of the Applications and that belong to a third-party vendor, supplied through licensing. The Nodes represent the physical machines that host the Applications and all of their functionality.

The main representations of this layer are the Node and System Software Catalogs, and the Node and System Software Context representations, shown below.

#### 4.1.4.A Node Catalog Representation

The Node Catalog represented in Table 4.6, shows every Node that is currently loaded into the Atlas repository, alongside its Central Processing Unit (CPU) core count, Random Access Memory (RAM) size in gigabytes, as well as its Operating System. The Nodes specified here are utilized to deploy the Application Components that are developed throughout the project.

New Nodes must be manually added to this table, as this information could not be collected from the identified data sources.

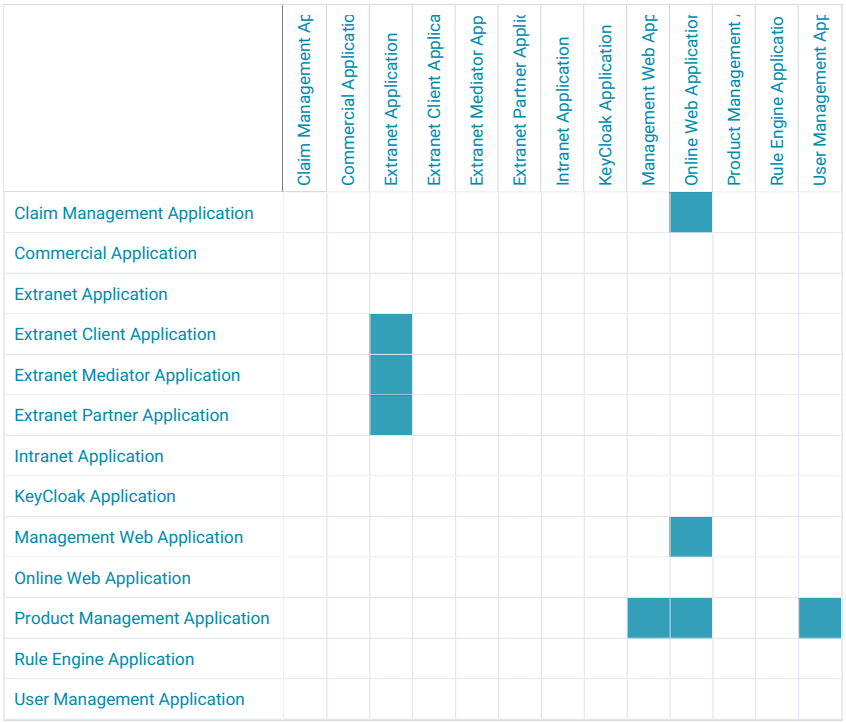


**Figure 4.5:** The Application Interaction Map Representation

| Name | CPU Core Count | RAM (GB) | Operating System |
|---|---|---|---|
| Production Application Server 1 | 8 | 16 | Windows 10 |
| Production Application Server 2 | 4 | 8 | Windows 10 |
| Production Application Server 3 | 4 | 8 | Windows 10 |
| Production Back-End Server 1 | 4 | 16 | Windows 10 |
| Production Back-End Server 2 | 4 | 8 | Windows 10 |
| Production Back-End Server 3 | 4 | 8 | Windows 10 |

**Table 4.6:** The Node Catalog Representation

| Name | Supplier | Productive Date | Decommission Date |
|---|---|---|---|
| .NET Framework 4.7 | Microsoft | 07-02-2022 | 31-12-2023 |
| Angular 13.0 | Angular Team | 08-03-2022 | 31-12-2023 |
| Swagger UI 4.10 | SmartBear Software | 24-03-2022 | 31-12-2023 |
| Windows 10 | Microsoft | 02-02-2022 | 31-12-2023 |

**Table 4.7:** The System Software Catalog Representation

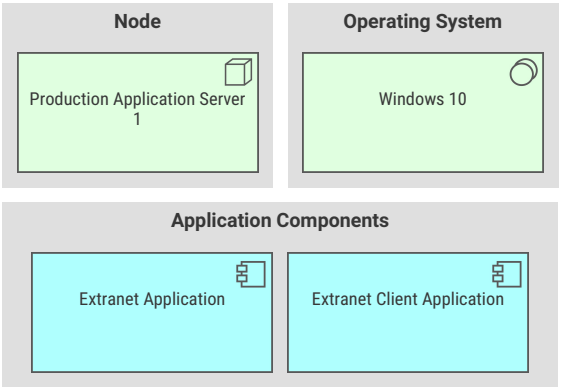


**Figure 4.6:** The Node Context Blueprint

#### 4.1.4.B System Software Catalog Representation

The System Software Catalog shown in Table 4.7 shows the System Software that is used by the Applications. A System Software includes its supplier, and the dates that correspond to the use of the Software.

New System Software must be manually added to this table, as this information could not be collected from the identified data sources.

#### 4.1.4.C Node Context Blueprint

Figure 4.6 includes the Node Context Blueprint. This representation shows the Applications that are deployed at a certain Node, as well as the Operating System that is used by that Node. Whenever information regarding the Operating System changes, this representation will display the new information. Additionally, as new Application Components are implemented and deployed on a certain Node, the Application Components will appear in the corresponding container.

#### 4.1.4.D System Software Context Blueprint

Figure 4.7 contains the System Software Context Blueprint. This representation shows the Applications that use a certain System Software asset, as well as its supplier. As the team inputs new information regarding the use of System Software by Application Components, new objects will be added to the

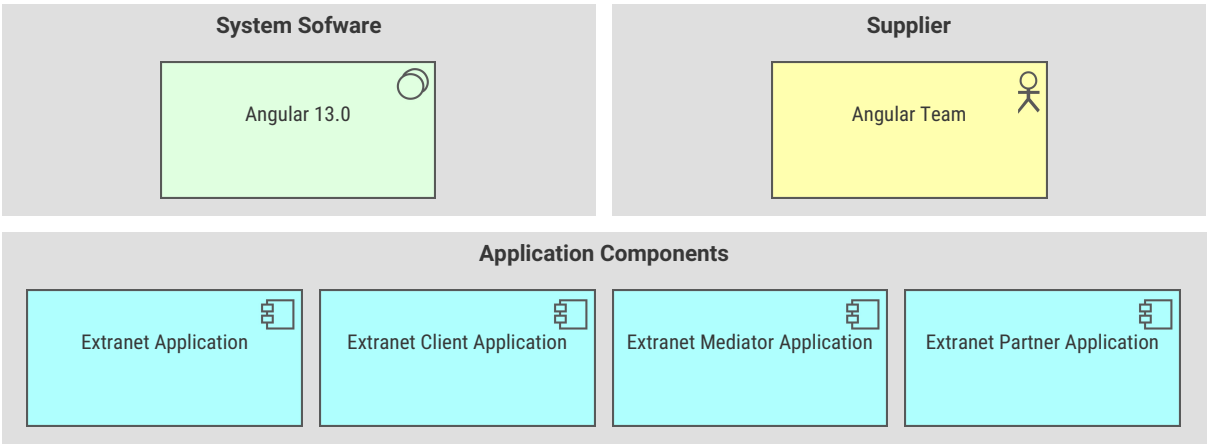


**Figure 4.7:** The System Software Context Blueprint

Application Components container. On the other hand, if a certain Application is changed, and no longer uses a certain System Software, that Application Component will no longer appear in the System Software Context Blueprint of that System Software.

## 4.2 Evolution of the Architecture

The components highlighted in Section 4.1 support a layered Enterprise Architecture. However, the Architecture of an Agile development project is constantly changing and evolving, with new requirements from the client and unforeseen changes during development. Therefore, relying on representations that are too difficult to update is detrimental to the overall value of the Architecture.

To combat that, the representations shown in the previous Section support Atlas' time bar. This feature, paired with the collected information regarding planned implementation, productive, and decommission dates, allows for a quick understanding of the evolution of the Enterprise Architecture.

This is supported through the use of lifecycles and the time bar feature in Atlas. Lifecycles can be created for any class within Atlas and they specify the sequence of states from the conception of the object, until the object is no longer utilized and removed. Each state can be assigned an order, and a date such that after a certain timestamp, the object will change its state, according to the information available in the repository.

A time bar is available in every Atlas representation, which allows for quick browsing through dates where state changes occur. This highlights the evolution of the Architecture in a visual manner, as colors are assigned to each state. For instance, Figures 4.8 and 4.9 show an example of how the time bar affects the representations in Atlas. Figure 4.8 shows the Goal Organic Blueprint representation with the time bar set to the 7th of February 2022. The Goals filled with a grey color correspond to Goals that are on the *Conceiving* state on this timestamp, which means that they are not yet realized (*Client Area*,

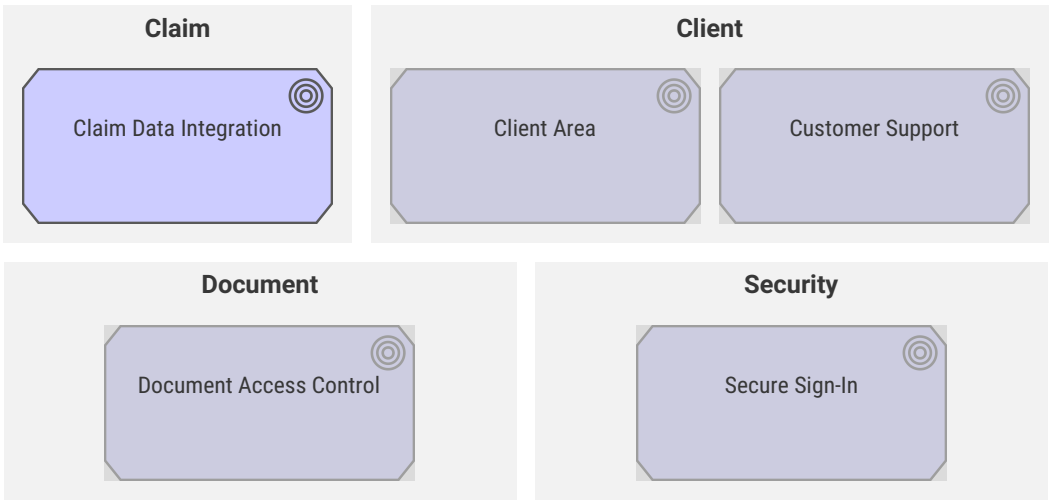


**Figure 4.8:** The Goal Organic Blueprint on 07/02/2022

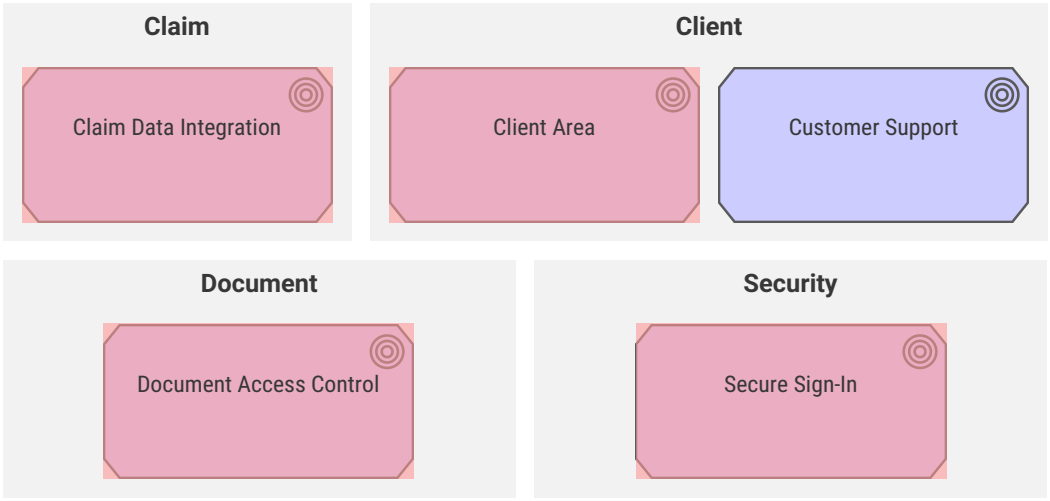


**Figure 4.9:** The Goal Organic Blueprint on 01/06/2023

*Customer Support*, *Document Access Control*, and *Secure Sign-In*). The *Claim Data Integration* Goal is in the *Productive* state, as it is not filled with any colors. Figure 4.9 shows the Goal Organic Blueprint representation with the time bar set to the 1st of June 2023. The Goals that are filled with a red color correspond to Goals that are on the *Decommissioned* state on this timestamp, which means that they have been in a *Productive* state in the past, and have been marked as no longer relevant (*Claim Data Integration*, *Client Area*, *Document Access Control*, and *Secure Sign-In*). On this date, the *Customer Support* Goal is still in the *Productive* state.

This information can change according to the imported data. For instance, if a Goal's decommission date property were to be changed in some manner, the result would be immediately visible in the representations that showed that Goal.

## 4.3 Assessment of the Proposed Solution

This Section provides an assessment of the proposed solution with the objectives established in Section 1.2 in mind. The main established concerns relate to the value offered by the Enterprise Architecture as well as the need to avoid high manual effort to be maintained.

Regarding the objective of gathering information whilst avoiding the disruption of the team's workflow, this is fully achieved with the use of the automated Atlas Batch Job (Section 3.3.2) and Swagger File Parser (Section 3.3.3), the data can be automatically collected from the respective sources and transformed into information supported by Atlas. Additionally, the impact of these automated tools is very limited as, on one hand, the Atlas Batch Job only requests textual information from Work Items that have been edited in the last three days, and on the other hand, the Swagger Parser is limited to the number of Applications that have been developed, which will limit its execution time growth.

Regarding the objective of automatically generating architectural documentation without introducing manual overheads, this is achieved by the creation of the dynamic representations in Atlas shown above. Atlas supports the creation of maps that are automatically updated as new information is added to the repository. For example, as new Goals are added, either through the collection of data from the DevOps environment or through manual input, the Goal Organic Blueprint (Figure 4.2) will be populated with new Objects and new containers, should a new Domain be created to categorize the new Goals. By the same token, as Goals reach their Decommission Date, or are removed from the repository altogether, they will no longer appear in the Goal Organic Blueprint. This ensures that there is no need to change the representation's specification, rather, the representations evolve as data evolves.

When it comes to the ability to update the data utilized to represent the architecture, there are some mixed results. Regarding the information that can be directly obtained from the automated processes, the effort required to update it is little to none, as the processes are deployed automatically on a daily basis. This is the case for the mapped information from the DevOps environment and Swagger files, such as the Goals, Requirements, Application Components, and Application Services that are created and interact. In contrast, information that can not be found through the deployed automated processes must be added manually, if needed. This is the case of the aforementioned Technology Architecture Layer and several dates that are not presented in the identified platforms. Therefore, the proposed solution offers an easily expandable starting point for the architecture that may evolve as necessary. It may, however, require some manual effort to update information that can not be mapped from external tools.

To summarize, a mapping between Agile development and the TOGAF framework is proposed and an Enterprise Architecture that follows the TOGAF ADM recommended artifacts (discussed in Section 3.3.5) is presented. The Atlas repository acts as the Architecture Repository, containing the updated assets and representations. The Architecture Vision layer of the Architecture supports the Architecture

Principles, through the Goal and Requirement tables and representations. The Business, Application, and Technology layers of the Architecture include the assets that are required by the Architecture Definition Document. All of the representations include the lifecycle of the objects, allowing each user to look into the future of the project and look at the planned changes, supporting an Architecture Roadmap of the project.

## 4.4 Limitations of the Proposed Solution

After analyzing the quality of the solution, some limitations are now highlighted. These limitations mainly concern the usability of the Atlas Agile Repository Configuration and the specificity of the Azure DevOps Batch Job and Swagger File Parser.

Given the fact that the Azure DevOps Batch Job was developed with the team's particular workflow in mind, it makes it difficult to replicate the automated processes in different settings. The data that is gathered from the DevOps environment is specific to the data structure in place. For instance, the mapping shown in Table 3.1 is limited to the work structure utilized by the development team and various information sources are limited to the use of the specific technologies (Azure DevOps environment and Swagger). This makes it a necessity to adapt the Atlas Batch Job to the working environment of each project, when trying to replicate these results.

Another limitation that is tied with the previous is the need for well-established data sources, that can be automatically queried through some kind of service. Without a structured development repository, automatically gathering data to build the architectural assets in Atlas is not possible. Looking at the proposed solution, some of the information that composes the representations can be automatically collected, however, there are several details that could not be gathered from the identified sources, and as such, have to be added manually. Therefore, to build the final architecture, some manual effort is needed to input the information that can not be automatically collected.

# 5

# Conclusion

## Contents

After assessing the solution and resulting architecture, two main conclusions are reached. The first is that Agile development and Enterprise Architecture are not mutually exclusive. This study shows an example of how an Enterprise Architecture framework (TOGAF ADM) can be deployed in the context of an Agile development project, without disrupting the workflow, and utilizing the available resources to facilitate its implementation.

The second discovery is that by analyzing the different information sources of an Agile project, it was possible to deploy automation processes that translated that information into architectural data. This proved useful when creating the underlying architecture, as it significantly lowered the manual effort required to maintain the architectural assets, allowing the project members to focus on delivering software instead of constantly updating the architecture. This is, however, highly dependant on the available data sources. The automated generation of representations requires at least some updated information to ensure the value of the architecture as a whole. Failing to identify and create these automated features will lead to an increase in required manual effort.

## 5.1 Resulting Architecture

The Architecture detailed in Chapter 4 shows a collection of representations that can be used during Agile software development to represent the current and planned state of the assets. The solution makes use of the identified data suppliers and current workflows to generate documentation that evolves as new requirements are created and as new modules are developed.

This creates a base that can be extended upon as the needs of the enterprise and the development team shift. For instance, if the team's workflow changes, the updated methodology can be used to develop new automated data gathering processes, to ensure that the information can be swiftly incorporated into the architecture. For example, if the DevOps environment were to be replaced with another Agile development pipeline, the Atlas Repository Configuration that was created could still be used, provided that a new Atlas Batch Job that requests relevant information from the new application's APIs was created. Therefore, the Atlas Repository Configuration that was created with TOGAF ADM as a guide, can be utilized as a basis for the creation of architectural assets for any Agile development methodology workflow that utilizes similar concepts.

On the other hand, depending on the priorities of the project team, data can be manually inserted by any team member when there are no viable ways to automate its collection, without the need for a separate architectural team. This is the case of some information displayed in Section 4.1, such as the Technology Architecture Layer (Section 4.1.4). As no viable mechanism was found to automatically collect the Nodes that are hosting the Applications and the Software that support each Application, this information was added manually by the team members.

Therefore, the result is a template that supports the creation of a layered architecture, focusing on the Architecture Vision (Section 4.1.1), Business Architecture Layer (Section 4.1.2), Application Architecture Layer (Section 4.1.3), and Technology Architecture Layer (Section 4.1.4), corresponding to phases A, B, C, and D of the TOGAF ADM, respectively. This shows that there is, in fact, a possible mapping between Agile development methodologies and Enterprise Architecture frameworks.

## 5.2 Future Work

After looking at the results and discussing some limitations, this Section contains pointers regarding possible future additions to the proposed solution. These additions are proposed at two levels, the first being a more complete mapping of TOGAF ADM structure within Atlas, and the second being the creation of more generic automatic data gathering processes.

Regarding the mapping between the TOGAF ADM and Atlas, the proposed solution includes artifacts that utilize the data that could be extracted from the team's workflow tools. This excludes several notions, such as Capabilities (representing the ability of an enterprise to reach a specific purpose) for the Architecture Vision Phase, Business Functions (representing the Capabilities aligned with the organization) and Business Services (representing interfaces that support the defined Capabilities) for the Business Architecture Phase. Supporting these concepts in the layered architecture and creating their respective artifacts would lead to a more complete architecture that could be adapted into more distinct use-cases.

Regarding the creation of generic data gathering processes, the proposed solution deploys specific processes that are tailored to the team's established workflow, namely the usage of Azure DevOps and OpenAPI specification files. To create a more generic solution, easily adaptable to more methodologies, extensions to these automation processes could be made. For instance, creating data collection processes that interact with other popular DevOps repositories (such as Jira) that collect data and generate Atlas XML Files, would create new use-cases for the solution.

# Bibliography


[1] J. A. Zachman, "Enterprise architecture: The issue of the century," *Database programming and design*, vol. 10, no. 3, pp. 44–53, 1997.

[2] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for agile software development," 2001, https://agilemanifesto.org/. Accessed 10/09/2022. [Online]. Available: https://www.agilemanifesto.org/

[3] P. Sousa, R. Leal, and A. Sampaio, "Atlas: the enterprise cartography tool," in *Proceedings of 8th the Enterprise Engineering Working Conference Forum*, vol. 2229, 2018.

[4] The Open Group, "The TOGAF Standard, Version 9.2 - Introduction to Part II," 2018, https://pubs.opengroup.org/architecture/togaf9-doc/arch/chap04.html. Accessed 14/10/2022. [Online]. Available: https://pubs.opengroup.org/architecture/togaf9-doc/arch/chap04.html

[5] M. Lankhorst *et al.*, *Enterprise architecture at work*. Springer, 2009, vol. 352.

[6] H. Jonkers, M. M. Lankhorst, H. W. ter Doest, F. Arbab, H. Bosma, and R. J. Wieringa, "Enterprise architecture: Management tool and blueprint for the organisation," *Information systems frontiers*, vol. 8, no. 2, pp. 63–66, 2006.

[7] K. Schwaber and J. Sutherland, "The scrum guide," *Scrum Alliance*, vol. 21, no. 19, p. 1, 2011.

[8] K. Schwaber, "Scrum development process," in *Business object design and implementation*. Springer, 1997, pp. 117–134.

[9] Digital.ai, "15th Annual State of Agile Report," 2021, accessed 03/10/2022. [Online]. Available: https://info.digital.ai/rs/981-LQX-968/images/SOA15.pdf

[10] M. Canat, N. P. Català, A. Jourkovski, S. Petrov, M. Wellme, and R. Lagerström, "Enterprise architecture and agile development: Friends or foes?" in *2018 IEEE 22nd International Enterprise Distributed Object Computing Workshop (EDOCW)*. IEEE, 2018, pp. 176–183.

[11] M. Watfa and T. Kaddoumi, “A foundational framework for agile enterprise architecture,” *International Journal of Lean Six Sigma*, 2021.

[12] M. McCormick, “Waterfall vs. agile methodology,” *MPCS, N/A*, vol. 3, 2012.

[13] F. Sandoval, V. Galvez, and O. Moscoso, “Development of Enterprise Architecture using a Framework with Agile Approach,” *ENFOQUE UTE*, vol. 8, no. 1, pp. 135–147, FEB 2017.

[14] W. Daoudi, K. Doumi, and L. Kjiri, “An approach for adaptive enterprise architecture.” in *ICEIS (2)*, 2020, pp. 738–745.

[15] S. Hanschke, J. Ernsting, and H. Kuchen, “Integrating agile software development and enterprise architecture management,” in *2015 48th Hawaii International Conference on System Sciences*. IEEE, 2015, pp. 4099–4108.

[16] M. Lambert, “How the TOGAF® Standard Enables Agility,” Jun 2018, Accessed 14/10/2022. [Online]. Available: https://blog.opengroup.org/2018/06/19/how-the-togaf-standard-enables-agility/

[17] E. Batkoski, “Azure DevOps Services REST API Reference,” Jul. 2022, publisher: Microsoft, Accessed 18/10/2022. [Online]. Available: https://learn.microsoft.com/en-us/rest/api/azure/devops/

[18] The Open Group, “ArchiMate® 3.1 Specification,” 2019, https://pubs.opengroup.org/architecture/archimate3-doc/. Accessed 18/09/2022. [Online]. Available: https://pubs.opengroup.org/architecture/archimate3-doc/

[19] H. Karthic, “Java SDK for managing Azure Devops Services,” Accessed 20/10/2022. [Online]. Available: https://github.com/hkarthik7/azure-devops-java-sdk

[20] The Open Group, “The TOGAF Standard, Version 9.2 - Architectural Artifacts,” 2018, https://pubs.opengroup.org/architecture/togaf9-doc/arch/chap31.html. Accessed 14/10/2022. [Online]. Available: https://pubs.opengroup.org/architecture/togaf9-doc/arch/chap31.html

[21] ——, “The TOGAF Standard, Version 9.2 - Architecture Deliverables,” 2018, https://pubs.opengroup.org/architecture/togaf9-doc/arch/chap32.html. Accessed 14/10/2022. [Online]. Available: https://pubs.opengroup.org/architecture/togaf9-doc/arch/chap32.html