FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# Towards Live Requirements Visualization, Management and Traceability

**Margarida Alves Pinho**


Mestrado em Engenharia Informática e Computação

Supervisor: Professor António Lucas Soares [FEUP]

Second Supervisor: Professor Ademar Aguiar [FEUP]

Second Supervisor: Luís Moreira [Vodafone]

November 5, 2022

# Towards Live Requirements Visualization, Management and Traceability

**Margarida Alves Pinho**

Mestrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Prof. João Pascoal Faria

Arguente: Prof. Ângelo Martins

Vogal: Prof. Ademar Aguiar

November 5, 2022

# Abstract

Live programming is an idea espoused by programming environments from the earliest days of computing, and its main goal is making feedback nearly instantaneous and evaluation always accessible. Software Development Life Cycle phases such as implementation, testing, and deployment have long benefited from tools that help engineers better identify and fix problems. Unfortunately, these are not totally contextually aware of important events from other life cycle phases such as failing tests or unsuccessful deployments, which might dissatisfy existing requirements. Moreover, these tools often require manual intervention to scrutinize problems and to get the requirements back to being met. Requirements engineering, in particular, has also benefited from similar tools for managing and tracing requirements. This work researches new ways of bringing the ultimate liveness to the visualization, management, and traceability of requirements in the programmer's integrated development environment (IDE). With this work, we have: (1) a state of the art on the topics of development environments, live software development, requirements engineering (namely requirements management and traceability), and respective tools; (2) an analysis of the key activities of software understanding and software development that benefit from live requirements and tools; and (3) a toolset to support live requirements visualization, management, and traceability of software systems, based on the listing of the project's requirements directly in a tool window of the IDE; fetching the requirements from an external sources, Gitlab; link the requirements written in Gherkin to the respective unit tests automatically generated through the usage of Cucumber; and displaying all relevant requirements information. The results were tested by using what was described above in a Vodafone project, and, by then judging how they have changed the development cycle, the advantages, and the overall impact. In general, the the tool was well received and considered useful.

**Keywords**: Live Requirements, Software Development Life Cycle, Requirements Engineering, Agile

# Resumo

"Live programming" é uma ideia defendida desde os primórdios da computação,com o principal objetivo de tornar o feedback do ambiente essencialmente instantâneo e a avaliação do programa é sempre acessível. Várias fases do ciclo de desenvolvimento de software beneficiam de ferramentas que permitem a identificação e resolução de problemas. Infelizmente, estes mecanismos não estão totalmente cientes de outros eventos relevantes nas outras fases do ciclo de desenvolvimento de software, como testes ou deployments sem sucesso, o que pode pôr em causa requisitos pré-existentes. Além disso, mecanismos como estes requerem, com frequência, intervenção manual, necessitando testes e implementações de software, de forma a restaurar o alinhamento com os requisitos. A Engenharia de Requisitos, em particular, beneficia de mecanismos semelhantes para gerir e rastrear requisitos. O presente trabalho investiga novas formas de trazer a melhor liveness possível para a visualização, gestão e rastreio de requisitos, colocando-as no ambiente de desenvolvimento integrado do programador. Para este trabalho Foi realizada (1) uma revisão de estado da arte acerca dos tópicos de ambientes de desenvolvimento integrado, live software development, engenharia de requisitos (nomeadamente gestão e rastreio de requisitos), e ferramentas respetivas; (2) uma análise das atividades chave na compreensão e desenvolvimento de software que beneficiariam de requisitos e ferramentas live e (3) um conjunto de ferramentas para a visualização, gestão e rastreio de sistemas de software, com base na listagem de requisitos do projeto diretamente numa janela de ferramentas do ambiente de desenvolvimento integrado do programador; em ir buscar os requisitos a fontes externas como o Gitlab; em ligar os requisitos escritos em Gherkin aos respetivos testes de unidade automaticamente gerados através do uso de Cucumber; e mostar toda a informação releante sobre os requisitos. Os resultados foram testados através do método supracitado num projeto da Vodafone e, subsequentemente avaliar como estas medidas mudam o ciclo de desenvolvimento de software e determinar as suas vantagens e impacto. Em geral, a ferramenta foi bem recebida e considerada bastante útil.

**Palavras-chave**: Requisitos "Live", Ciclo de Desenvolvimento de Software, Engenharia de Requisitos, metodologia Agile

# Acknowledgments

I want to thank Vodafone for giving me an opportunity. I've learned so much this past year by working with my team, and they made this dissertation possible. I also want to thank Professor Ademar Aguiar and Professor Lucas Soares that accepted to embark on this journey with me. All the advice and guidance were crucial, and I appreciate endlessly the patience and comprehension.

Without some very special people in my life, this journey would not have been the same. So, I want to thank my family for giving me the biggest safety net anyone could ask for. You are all relentless in your support and do the impossible just to try to help me succeed. You help me grow every day and I am better because of it.

I also feel deep gratitude towards my friends Ana Teresa and Maria Caldeira. Your friendship and encouragement mean a lot to me and were vital these past five years. Additionally, I would like to acknowledge someone that even though it's not been in my life for long, it has already impacted it. Your faith in me and your limitless emotional support had an impact on this adventure.

A special thank you goes to Carlos Duarte for his friendship, for giving me directions when I couldn't find them, and for believing in me. Another goes to my brother, Simão Pinho, that truly cares about me and helps me like no one. And I also couldn't leave out my friend, Filipa Ferreira, that made me feel understood and that didn't leave my side. Your help made all the difference and made me feel like I wasn't alone in this.

Margarida Pinho

*"We do not have to become heroes overnight.*
*Just one step at a time,*
*meeting each thing that comes up,*
*seeing it is not as dreadful as it appeared,*
*discovering we have the strength to stare it down."*

Eleanor Roosevelt

# Contents

# List of Figures

# Abbreviations

| | |
|---|---|
| RE | Requirements Engineering |
| SD | Software Development |
| SDLC | Software Development Life Cycle |
| LiveSD | Live Software Development |
| IDE | Integrated Development Environment |
| VS Code | Visual Studio Code |
| JVM | Java Virtual Machine |
| SAFe | Scaled Agile Frameworks |
| UI | User Interface |
| UX | User Experience |

# Chapter 1

# Introduction

**Contents**



This chapter introduces the project's context, motivation, and objectives, and makes an overview of how the dissertation is structured.

## 1.1 Context

Live programming promises a more fluid response between the programmer and the program. Indeed, the importance of nearly instantaneous feedback and accessible evaluation is undeniable, and, today, languages are supported by sophisticated environments that have the potential to significantly improve the programmer's overall productivity. The role of Live Software Development (SD) is, thus, empowering software engineers and assisting their work throughout the Software Development Life Cycle (SDLC), which is a development methodology for creating software that comprises all the necessary phases, steps, and instructions to develop, test, and alter software [28].

There are tools to better identify and fix problems that can be used in any phase of the SDLC. Besides making the engineer's work easier and more fluid, these tools can also prevent project delays [2]. Because of this, Requirements Engineering (RE) is an area that can benefit particularly from live software tools for managing and tracing requirements. Contextual awareness of problems that might arise in different SDLC phases, such as failing tests or unsuccessful deployments

can greatly facilitate the engineer's work. Thus, these tools can make obsolete the need for the engineer to manually assess and diagnose development issues.

## 1.2 Motivation

The importance of feedback from live programming tools is a reality that Vodafone recognizes. Furthermore, at the company, they wish to make use of the increased productivity and work fluidity that such tools would bring to requirements engineering.

At Vodafone, it is already used an Agile methodology, specifically Scrum [29], but there is still a search for a solution that enables the process of making projects to be more effortless and less tedious for the developers during each Sprint. In Scrum, Sprints are fixed-length periods used to develop the requirements for a project, while ensuring inspection and adaptation of progress.

## 1.3 Problem

As it stands, at Vodafone there is a lack of communication between SDLC phases when developing new software. Indeed, tools for managing and tracing requirements are not contextually aware of important events from other life cycle phases such as failing tests or unsuccessful deployments, which might dissatisfy existing requirements. Because of this, developers need to manually assess what is happening with tests and software implementations to find out how to get all the requirements back to being met, which is not the best.

## 1.4 Objectives

The main objective of this project is to develop a tool to bring liveness to the visualization, management, and traceability of requirements. Furthermore, this tool is to be used as an extension of the programmer's integrated development environment (IDE). This is a simple, elegant idea, however, its execution is not as straightforward as it may seem. To achieve this goal, first, an analysis of the state of the art on the topics of Development Environments, Live Software Development, Requirements Engineering (namely requirements management and traceability), and respective tools will be conducted. Then, an analysis of the key phases of software understanding and SD that would benefit from live requirements and tools will be performed. Finally, a toolset to support live requirements visualization, management, and traceability of software systems would be provided, thereby seamlessly improving the process of making projects for the developers during each Sprint, increasing their satisfaction. Likewise, as a result of the optimization of the development process, an increase in productivity and the overall success of the final product is expected.

## 1.5 Document Structure

This report is divided into 6 Chapters. In this one, Chapter 1, the overall project is introduced, including the dissertation's context, motivation, and objectives. Chapter 2, regarding the state of the art, explores themes that allow a better comprehension of the dissertation. It is an exposition of the background, related work, existing technologies, and approaches to the same or related problems. Subsequently, Chapter 3 describes the problem and the corresponding proposed solution is presented in Chapter 4. In Chapter 5, the results of this dissertation are explored, as well as a survey to evaluate the proposed solution. Lastly, Chapter 6 completes this dissertation and forecasts the expected results of the project.

# Chapter 2

# State of The Art

## Contents



This chapter serves to present some relevant background for this dissertation and to inform about the current state of related work.

## 2.1 Background

In this section, relevant topics to the motivation and objectives of this dissertation are presented.

### 2.1.1 Requirements Engineering

Requirements are what define exactly what is going to be developed and accomplished. Every year, projects fail due to unfulfilled requirements. This supports the idea that to guarantee a solid software product, it is essential to pay attention to details and dedicate sufficient effort to RE activities.

The development of requirements can be divided into different phases: these are requirements elicitation, requirements analysis, requirements specification, and requirements verification and validation. Another important aspect that should be taken into consideration is the management of requirements. In fact, during SD, it is necessary to acknowledge changes in requirements and tackle the ensuing issues: tracing, conflicts, prioritization, and implementation. The following subsections describe each one of these phases.

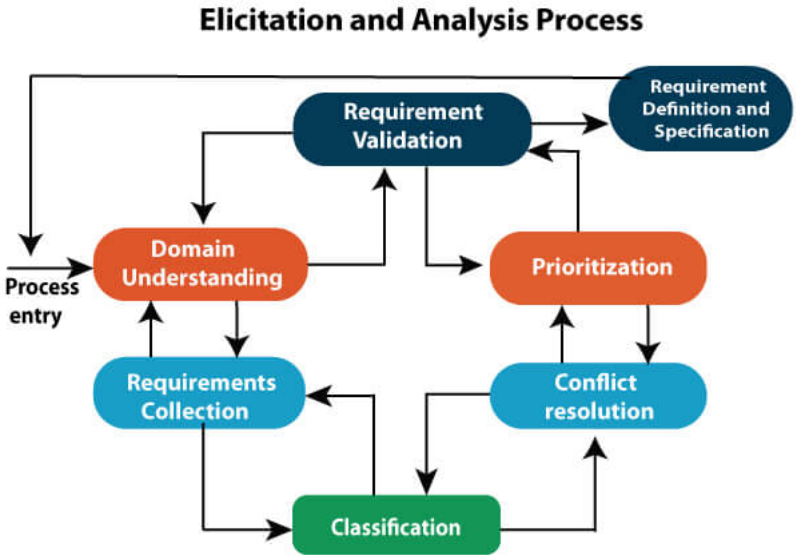


Figure 2.1: First two Requirement Engineering Processes [19]

#### 2.1.1.1 Requirements Elicitation

During this phase, the basis of the product is defined, and requirements are identified. Through effective communication with the stakeholders, one should identify what is pretended, along with what users will need. This is the most crucial part of SD activities. To guarantee a project's success and stakeholders' satisfaction, it is necessary to have a deep understanding of the specific project at hand, as well as expert background knowledge. In fact, should information happen to be missing or poorly explained, devastating consequences may arise, compromising the success of the whole endeavor. Nonetheless, it is common, and acceptable up to a certain point, for requirements not to be clear at the beginning of a new project. This issue should, therefore, be explicitly discussed and solved in the development phase of the software [31].

#### 2.1.1.2 Requirements Analysis

During this phase, it is necessary to consider how the different requirements work together. Overlaps and conflicts among requirements are not desirable and should be identified. Through an exploratory, detailed approach, it is fundamental to eliminate them [31].

#### 2.1.1.3 Requirements Specification

This is the phase when the formal software requirements are defined. The functional requirements (a function that a system or system element must perform)[19], as well as the non-functional (necessities that specify the criteria that can be used to decide the operation)[19] and both their constraints are identified and specified.

Sometimes, the elicitation process is again triggered, owing to the need for more data about the project. Most commonly, user stories are employed to solve this issue. According to scrum.org, " A user story is an informal, general explanation of a software feature written from the perspective of the end user. Its purpose is to articulate how a software feature will provide value to the customer." [4] They are often simple, but elucidating sentences, structured as follows: "As a [persona], I [want to], [so that]." [4]

When user stories depend on one another, together they form an Epic. It is important to have tools to trace them so that an understanding of the relation between them is formed.

#### 2.1.1.4 Requirements Verification and Validation

The goal of this phase is to guarantee that each requirement was correctly implemented (verification) and that the product that has been implemented is traceable to the requirements (validation).

### 2.1.2 Agile Software Development

The process through which software is created and maintained is known as the System or Software Development Life Cycle [16]. There are different models regarding the optimal structure of one such cycle, each with its advantages and disadvantages [10].

One popular model is Agile Software Development. This model incorporates a methodology that favors adaptiveness and the capability to respond to change. It allows for short feedback loops that promote collaboration between self-organizing cross-functional teams, thereby favoring a versatile, iterative approach to software development. At the root of the Agile Manifesto [14] are 12 principles that encompass the guiding practices that support teams in implementing and executing software development with agility. These focus on simplicity, communication, self-organization, continuous project evaluation, and behavior adjustment.

### 2.1.3 Behaviour-Driven Development (BDD), Cucumber and Gherkin

BDD is a work philosophy that enhances Agile processes (previously described - 2.1.2). It encourages teams to have the right conversations at the right time, by dividing issues it into necessary

smaller changes that culminate in new functionalities. BDD is the software development process that Cucumber was built to support. Explained succinctly, BDD is an iterative process consisting of three steps that work to make large system changes rapidly. The three steps are Discovery, Formulation, and Automation.

Discoveries are part of discovery workshops, which are structured discussions that focus on real-world examples of the system from the users' perspective. From these, emerges a shared, growing understanding of the users' needs, the system-governing rules' function, and the necessary scope of future changes. Discoveries are the finding of concrete examples that can be used to illustrate what the system is supposed to do.

After identifying valuable examples from discovery workshops, it is necessary to correctly determine and describe each example, making sure every member of the team understands it. The formulation is, thus, a means of translating examples into structured documentation. This consists of describing Discoveries in Gherkin syntax. Gherkin is a language that uses plain English (or other supported languages), to describe use cases for a software system in a way that can be read and understood by almost anyone. This is extremely valuable because it allows feedback from the whole team, even non-developers and people unfamiliar with programming language.

Lastly, as part of Automation, the structured examples are connected to the system, as tests, using step definitions. These connect Gherkin's steps to programming code and hard-wiring specifications to implementation. When following BDD using Cucumber, examples, called scenarios, are used to specify what the software is meant to do. Scenarios are, therefore, written before production code, as executable specifications, defined in .feature files. As production code is written, scenarios become automated tests. Fast tests enable the developers to run them frequently to obtain fast feedback on what they are building, without losing focus or flow.

### 2.1.4 Liveness

In line with Agile's philosophy is the concept of Liveness in software development. Liveness is defined as "an always accessible evaluation and nearly instantaneous feedback, usually focused on coding activities" [2]. Consequently, one can interpret liveness as the effort of ever-tightening the feedback loops that comprise Agile's philosophy of continuous project evaluation and permanent versatility.

Traditionally, to verify their work, a code developer would have to write code, then run the program and check the results [31]. These evaluation cycles or loops can often be repetitive and frustrating. As such, a live programming environment where code evaluation is constantly happening as the developer writes their code and gets nearly instantaneous feedback on the results of the new changes or addition to the code is an undeniably useful environment.

Liveness is not, however, a binary concept. The software can have different levels of liveliness, Figure 2.2 from merely informative add-ons to the environment to strategically predictive tools that can predict the programmer's intentions and the goal of the software being developed [31]. This is an important concept to keep in mind when evaluating existing tools and deciding what level of liveness would be most useful for a specific project.

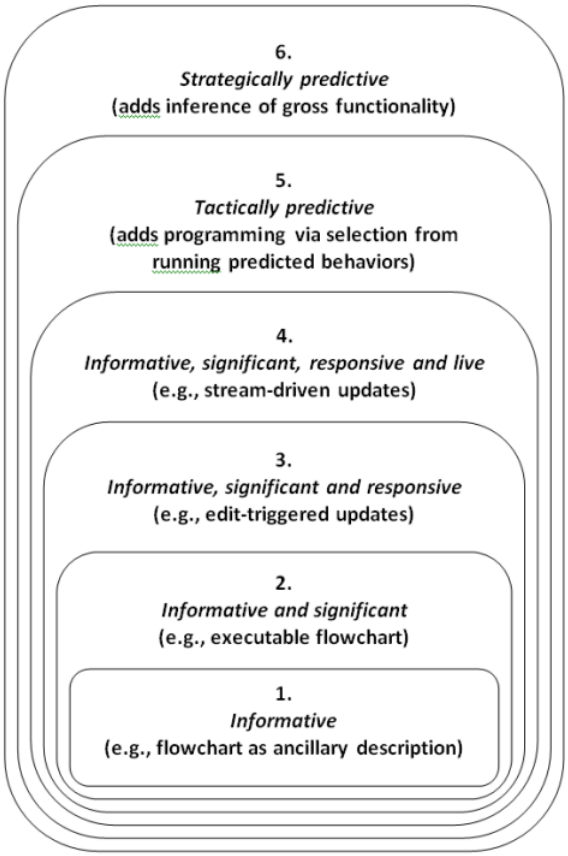


Figure 2.2: Proposed extended version of the liveness hierarchy [31]

### 2.1.5 Scrum

Scrum is an Agile project management framework that describes the necessary set of meetings, tools, and roles that helps teams architect and manage their work, allowing them to self-organize and continuously learn from experience [3].

A Scrum Team consists of one Scrum Master [8], one Product Owner [7], and Developers [6]. The Scrum Master is responsible for establishing Scrum as defined in the Scrum Guide and helping understand Scrum concepts and methods. The Product Owner has the goal of maximizing the value of the product, and the Developers create and improve the project in each Sprint.

Another aspect of the Scrum framework is that there are defined Events: Sprints, Sprint Planning, Daily Scrum, Sprint Review, and Sprint Retrospective [9]. A Sprint is a fixed-length period, during which the developers implement user stories. This is done in this way to create consistency and an exact schedule. At the beginning of each Sprint, there is a Sprint Planning meeting. There, it is decided what exactly will be done; decisions made can't be changed until the end of the Sprint. When a Sprint finishes, the outcome of the work is discussed during the Sprint Review.

At the Sprint Retrospective, the team debriefs and discusses what went well and what they think could be improved, planning ways to increase quality and effectiveness for the next Sprint.

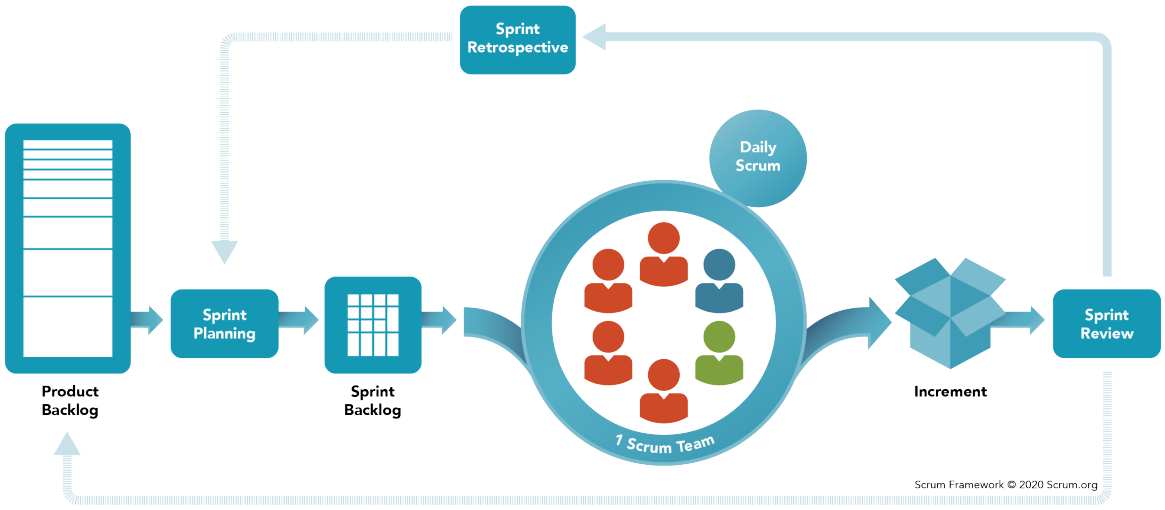


Figure 2.3: Scrum Framework - graphical view of how Scrum works [5]

## 2.2 Pre-existing Tools

Several frameworks aim to improve liveness in software development. It should be noted that some elements of liveliness have been used for decades [20].

Web-related liveness is particularly prevalent. Indeed, certain JavaScript frameworks such as React.js [21] and Vue.js [32] allow for immediate feedback on code changes.

A step further are what can be called live programming languages, such as Pharo [18], Circa [17], Interstate [27], or Euclase [26]. These were built with liveness in mind and come with an Integrated Development Environment, making the developer's work experience significantly more immersive.

All types of changes have an impact on code development, and there are already Agile methods and tools that have the goal of making changes more visible and of facilitating the necessary actions to deal with them, like Microsoft's IntelliSense. Likewise, some tools already allow developers to visualize tests, check if they are failing, and then manually go over the code to try to come up with a solution. Software used for automated test generation has also been advancing over the last decade [15].

Furthermore, services such as GitHub and GitLab CI give developers live logs and notifications of failures in real-time. This helps reassure developers, making them more prone to build, manage and deploy projects.

There are also tools to assist in Requirements management that are available. These make it easy to capture, trace, analyze, and manage changes to information. Some of them are DOORS, Rational RequisitePro, and TopTeam Analyst. They grant documentation support and the capability to link requirements to design items, test plans, test cases, and other ways to trace the project. This is essential to rapidly answer to unexpected events [15].

### 2.2.1 GitLab

To maintain organization, SD teams benefit from the help of tools such as GitLab [13]. Indeed, GitLab is great for teams that want to follow an Agile mindset, bettering the way teams collaborate, build software, track key decisions, and more.

GitLab has a few fundamental concepts that one needs to be familiar with to understand how it facilitates Agile approaches to project development: - The issue in GitLab is the basic planning object. Issues are used to collaborate on ideas, solve problems, and plan work. - Boards, on the other hand, allow for visual project management. They can be used to organize items and, by calculating the total size of issues, provide an understanding of how much work is needed. Labels are concepts used to define columns in the issue boards and make it easy to search and find issues. - Merge requests are the way issues and actual code changes are linked. Each request captures code changes and reviews, as well as approvals and security scans. - Milestones establish a target date for a sprint or specific bundle of issues and code changes to be delivered. - Continuous integration and continuous deployment (CI/CD) pipelines are a series of tasks needed to develop and implement new software versions. Even though it would be possible to manually execute each of the steps of a pipeline, its true value is realized through automation.

In GitLab, the project is the keystone, where the entire work cycle is materialized. As such, teams can collaborate and thus make use of their backlog of use cases (issues), assign tasks (on boards), work on features (issue discussions), develop and submit code (with merge requests), review modifications (merge request discussions), integrate applications and, finally, deploy them (continuous integration and continuous deployment pipelines).

GitLab is especially useful for teams following an Agile mindset. No matter what their chosen Agile methodology is, it enables teams to manage and organize their work and apply Agile practices and principles. Development teams usually adopt iterative, incremental, and lean project 30 methodologies, such as Scrum, Kanban, and Extreme Programming (XP). While large enterprises, adopt a variety of frameworks, including the Scaled Agile Framework (SAFe) which allows large enterprises to adopt Agile delivery practices on a large scale. The interplay between GitLab and the Agile mindset is very smooth and effective. GitLab can be used whether you just want to track a 34 few issues or want a single application for the complete DevOps lifecycle

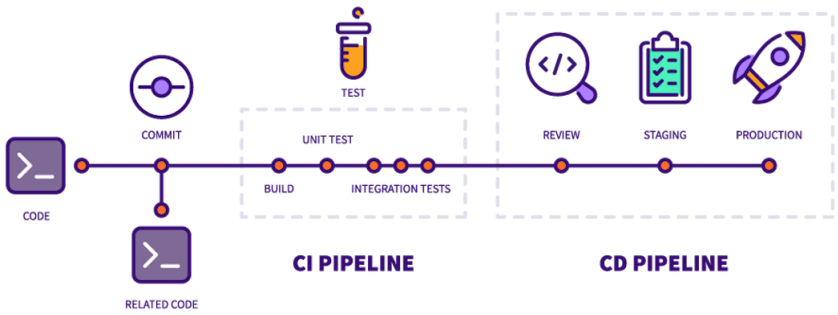


Figure 2.4: Agile model on GitLab [11]

An Agile iteration can be easily managed using GitLab. In Agile, you often start with a user story that captures a single feature, in GitLab, an issue within a project can be used for this purpose, and in its description, a list of tasks can be added. Then the user stories are prioritized in a product backlog, in GitLab this can also be done. Labels can be created and assigned to individual 4 issues, which then allows you to filter the issue lists by labels. Another thing done in Agile is the estimation of the level of effort for each user story, and this can be done in GitLab by giving 6 issues a weight attribute. During the sprint (GitLab milestone), the development team distributes user stories to work on, and in GitLab issues have assignees that represent the person who is currently responsible for it.

### 2.2.2 Integrated Development Environments

This dissertation's end goal, which will be further detailed in Chapter 3, requires an exploration of what an Integrated Development Environment (IDE) is. Generally, an IDE consists of a source code editor, local build mechanism, and debugger. It enables developers to consolidate the different aspects of writing a computer program. It allows them to promptly start programming new applications because multiple services are already configured and integrated.

Some live programming concepts have already been integrated into IDEs. In "A perspective on the evolution of live programming" [31], Steven L. Tanimoto mentions that in current IDEs like Eclipse for Java programming, feedback is provided to the developers by many features that work during the process of coding. Such features include syntax highlighting, code completion suggestions, and indications of problems associated with various locations in a source file. Another case presented by Steven L. Tanimoto is a feature of The Java Virtual Machine (JVM) from version 1.4 onwards that allows the replacement of a class file by another while the overall program is running. With this, a new version integrated is immediately compiled and inserted into the running JVM. However, these solutions could benefit a lot from increased liveness.

### 2.2.3 Visual Studio Code

The IDE explored in this dissertation is Visual Studio Code (VS Code), as this is the IDE used at Vodafone. It has the advantage of being a lightweight, but still powerful source code editor, and is

available for different operating systems: Windows, macOS, and Linux. It is very accessible and runs directly on the computer's desktop.

VS Code is an editor that, like many others, has a simple and easy-to-use user interface (UI). The main components of the UI together are called the VS Code Workbench and they are the following: the Editor is where you edit your files, you can have more than one editor open side by side; the Activity Bar lets you switch between views and gives you context-specific indicators; the Side Bar contains different views that assist you when working on your project; the Status Bar provides additional information about the opened project and the files you edit; the Panels serve for output or debug information, errors, and warnings, or an integrated terminal (Figure 2.5).

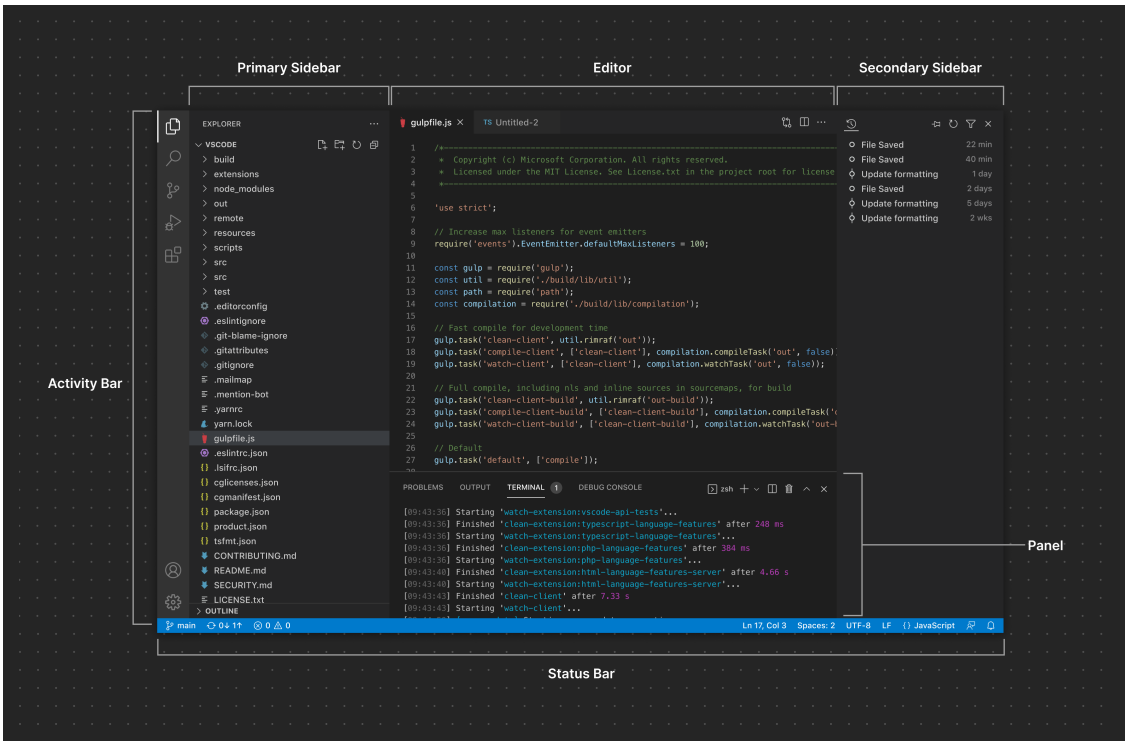


Figure 2.5: Basic Layout of VS Code [22]

VS Code supports almost every major programming language and is very versatile. Its extensions let you add languages, debuggers, and tools to your installation to support your development workflow [25]. This IDE is particularly interesting for this dissertation's scope because it lets you extend the Workbench by adding your components and views.

This IDE is built with extensibility in mind and almost every part can be customized and enhanced through the Extension API. Several core features of VS Code are themselves extensions and use the same Extension API. Taking this into consideration, this project aims to use this extensibility to create a tool to support requirements visualization, management, and traceability

## 2.3 Similar approaches to the problem

To reflect on my approach to the problem, research on previous work was conducted and is discussed in this section.

In an article from 2021 [15], Carlos Duarte discusses the same issues and problems of this dissertation, as detailed in Chapter 3, exposing the benefits, of a live environment. He argues that creating a set of integrated tools in the development environment would help developers better follow all the SDLC activities.

Another article by Ademar Aguiar [2] also aims to incentivize contributions for an approach that incorporates more liveness aspects. This article suggests that it would be interesting to enrich already existing tools instead of creating new ones that might be incompatible with existing practices, which goes in hand with my solution detailed in Chapter 3 of this dissertation.

It should also be noted that previous works have produced extensions that have a similar purpose to the one this work aims to develop, showing that such an aim is realistic and achievable. One such example is the GitLab VS Extension [23], which is the official, GitLab-maintained, extension for Visual Studio Code. This extension specifically allows the user to integrate GitLab into Visual Studio Code, allowing such operations as viewing issues and managing pipelines, among others.

# Chapter 3

# Problem

Contents



In this chapter, the problem introduced in Section 1.3 is further explored. A solution is proposed, bringing liveness to IDEs. Finally, a work plan to implement it is presented

## 3.1 Lack of communication between SDLC phases

Requirements tasks influence the whole SDLC. It's rare to see a process in which requirements are simply sequentially elicited, set, allocated, and delivered to the developers. This happens since many requirements will unavoidably change [2]. Indeed, change is recognized as a factor that greatly influences requirements and permeates all SDLC phases.

SDLC phases already benefit from tools that help engineers better identify and fix problems, providing them with useful insights and, in some cases, taking care of those issues automatically. Unfortunately, these are not contextually aware of important events from other life cycle phases such as failing tests or unsuccessful deployments, which might dissatisfy other requirements. Moreover, these tools often require manual intervention, with developers needing to assess "by hand" what is happening with tests and software implementations to find out how to get the requirements back to being met, which quickly becomes demanding, exhausting, and prone to human errors. Nowadays, development systems are of high complexity levels and have a high number of requirements, making it very tiresome to handle issues manually. Simply put, the faster

the developers are conscious of the way the code they are producing is affecting the accomplishment of the requirement, the better.

## 3.2 Problem Contextualization

At Vodafone, Scrum, an AGILE methodology, is used for SDLC. However, the management process can be improved, and the lack of communication between SDLC phases is a particularly poignant issue. An ideal solution enables the process of making projects to be more effortless and less tedious for the developers during each Sprint. As such, finding tools that would enable developers to automatically know if the requirements they defined with their clients are being met would be very useful. Customer satisfaction is always the end goal, and in an era where every second counts, seemingly small improvements to the work process can make a big difference.

Moreover, Vodafone is a very large company, with equally large and resourceful competitors, making it paramount to optimize SDLCs. Keeping up with the newest developments to the requirements of their projects and managing these changes efficiently is not only useful but a necessity. In the development of this dissertation, it's used GitLab [12] for managing the workflow, and Visual Studio Code (VS Code) [25] to develop software. The team consists of people with different levels of knowledge and areas of expertise, making it interesting to observe how each one will perceive the importance of the application of this dissertation's findings.

## 3.3 Proposed Solution

Managing requirements can be improved by LiveSD; in fact, each phase of the SDLC can benefit from receiving other phases' feedback. This could be achieved by making changes directly in the IDE. This adoption of LiveSD features would lead to an improvement in the developer's experience. An important aspect of SD is testing, which is a process that serves as the basis for linking requirements with other SDLC activities. IDEs could improve the requirements change management process, by better integrating them with the environment. Allowing the recognition of alterations and assessing tests' impact will lead to programmers requiring less effort to do the same work. These improvements make it possible to directly visualize the output of the running system, simplifying the overall process. Indeed, the faster it is possible to go over the problem, the faster one may find the appropriate solution. For this to happen, first, the developer would need to be able to visualize all the relevant requirements information directly in the IDE. Additionally, this list of the project's requirements would have to auto-update instantly and automatically. Finally, the developer would have to be able to manually link the requirements to the respective unit tests, automatically generated through the usage of tools such as Cucumber [30], requirements using natural language. Live environments are a possible solution for automating processes, as well as aiding the developer [15].

## 3.4 Methodology

As detailed previously, this project aims to implement live programming in a Scrum setting at Vodafone. To achieve this, initially, a literature search was performed to attain an overview of existing tools for managing and tracing requirements. Secondly, using this knowledge and applying it in the case of an attributed project, the existing tools previously researched were used to create an extension, that better fit Vodafone's necessities and specifications. With this, an optimization of the programmer's development environment was conducted. Lastly, to guarantee the work's quality and usefulness, its results were tested and evaluated through a survey to access the project's capability to improve the Software Developing Cycle and its overall usefulness and impact on the engineer's ease of work.

## 3.5 Summary

As discussed in this chapter, Requirements tasks influence the whole SDLC phases and the lack of communication between SDLC phases is a particularly caustic problem. Tools for managing and tracing requirements are not contextually aware of important events from other life cycle phases such as failing tests or unsuccessful deployments, which might dissatisfy existing requirements. Developers need to assess ”by hand” what is happening with tests and software implementations to find out how to get the requirements back to being met.

# Chapter 4

# Solution

**Contents**



In this chapter, a solution to the problem previously discussed is presented. As such, the basis and details about the VS Code extension developed as part of this thesis are presented. The components of the extension and its impact on the overall programming environment are explained.

## 4.1 Extension Aims

It is necessary to have tools that help developers cope with change by incorporating requirements visualization, traceability, and management activities[15]. To come up with a smart, useful solution to the Problem previously described in 1.3, a list of well-defined, realistic aims for the Extension was drafted. Consequently, the culmination of this work is the implementation of an extension on the Sidebar of VS Code, that would allow the user to - log in to their GitLab account; - choose a project repository; - visualize the chosen project's requirements, written in Gherkin, and fetched from its GitLab Issues; - tweak and manage the requirements directly in the IDE; - trace the requirements to the respective tests, generated through the usage of Cucumber [30].

## 4.2 VS Code User Interface

To understand the process of implementing a VS Code extension, it is important to better define a few concepts of the VS Code's UI. Firstly, it should be established that the VS Code interface is composed of two main elements: containers and items. Containers are large sections of the interface that render one or more Items. Thus, the Activity Bar, Side Bar, Editor Area, Panel, and Status Bar are all Containers. Then extensions can add items to the various containers listed above. All these elements are illustrated in Figure 4.1.

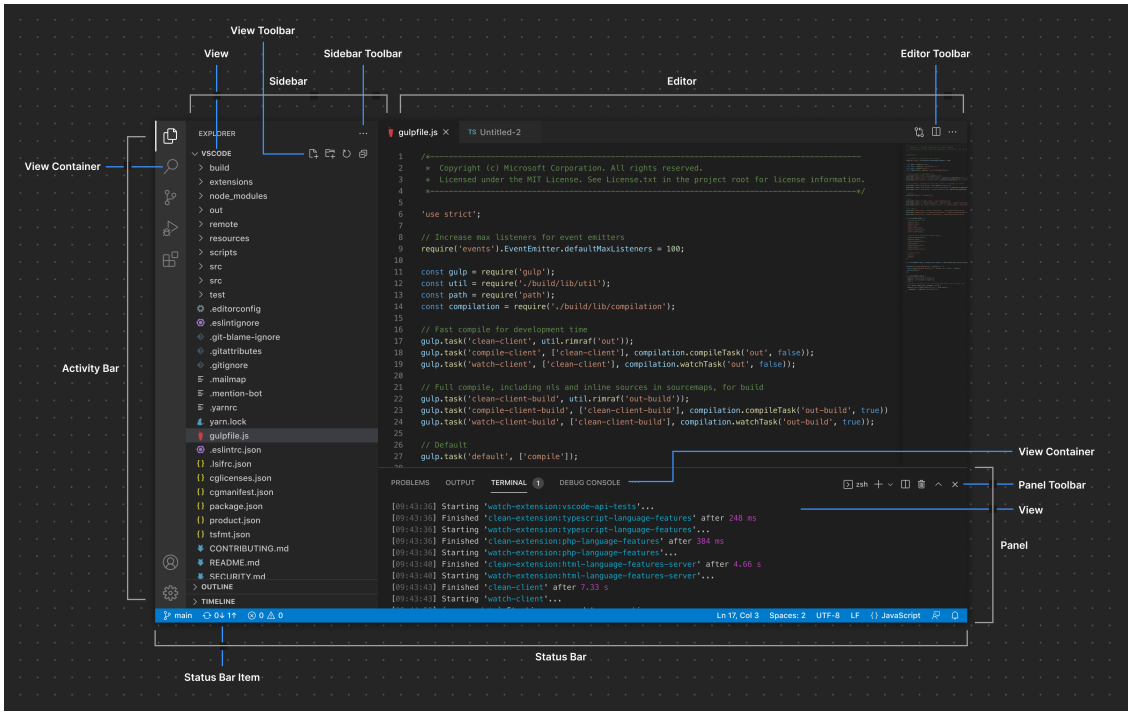


Figure 4.1: VS Code interface

,

Moreover, Views are containers of content that can appear in the Sidebar or Panel. These can be contributed in the form of Tree Views, Welcome Views, Webview Views, or even show View Actions. Views can also be rearranged or moved to another View Container (for example, from the Primary Sidebar to the Secondary Sidebar). View Containers, as the name implies, are the "parent" container in which Views are rendered. Extensions can contribute to existing View Containers. Some of them can be found in the Activity Bar and appear as Activity Bar Items such as the File Explorer and the Source Control (SCM), while others are in the Panel. Nevertheless, Views can also be added to custom View Containers.

Tree Views are particularly relevant in the context of this thesis' extension since it was decided to develop an extension for displaying data that customized the Side Bar. The Tree View API, simply, allows content to be showcased on the Sidebar, structured as a tree and you can add everything from simple flat lists to deeply nested trees. When a View is empty, you can add content to guide users on how to use your extension or get started such as links and icons. These

are called Welcome Views and should only be used strictly when necessary. If you need to display custom functionality that is beyond what the VS Code API supports, you can use webviews, which are fully customizable. It's important to understand that webviews should only be used if you need them.

Other common UI elements are the Command Palette where all the commands quickly execute some functionalities; Quick Pick that captures a user's input in several different ways; Notifications that communicate information, warning, and error messages to users; and Context Menus that enable users to perform actions or configure something from a specific location.

## 4.3 Vs Code User Experience Guidelines

When creating an extension for VS Code, one must take into account several guiding considerations. Only by following pre-defined best practices for creating extensions is it possible to create an extension that seamlessly integrates with VS Code's native interface and patterns.

For the VS Code extension, this thesis is based on, it was decided to customize the Side Bar. This was chosen as it would allow for an interactive experience without hampering the user's work or making the end product too cluttered. It was also established to not contribute to an existing View Container, but instead to add a custom one via the Activity Bar. Even though the Panel also functions as another area to display View Containers, the panel is better suited to render Views that benefit from more horizontal space. This is not our case since we want to list all the requirements of a project and their tests.

User Experience (UX) guidelines [24] are intrinsically connected to the development process of a good VS Code extension. Regarding the present thesis' extension, the guidelines pertaining specifically to sidebars are, logically, the most relevant. According to the VS Code's UX Guidelines, when developing a sidebar-oriented extension, one should aim to group related Views and content and use clear descriptors for View Containers. Likewise, the usage of too many containers or too many views should be avoided. It is also important to avoid repetitiveness and the addition of unnecessary content, for example, content that could be replaced by a simple command.

Regarding the Views themselves, it's advised to use existing icons when possible and to add an icon to every View. Also, in the case of Tree Views, it's important to have descriptive labels to give context to items and to use product icons to distinguish between item types. Many extensions choose to contribute to the Primary Sidebar given the high visibility. However, adding here too much contributed UI can lead to a cluttered experience that can worsen the developer's productivity. Hence the need to have good judgment and in general to be careful to not add too many actions to reduce clutter and confusion.

## 4.4 Implementation basics

To develop a VS Code extension, firstly it's necessary to have Node.js, and Git installed. In the case you are building one from scratch, Yeoman and VS Code Extension Generator are also useful

to have installed. The generator scaffolds a TypeScript or JavaScript project, after running it you need to fill out a few fields, and then it creates an initial project ready for development. For this thesis, it was decided that the development was going to be made with TypeScript. To compile and run the extension in a new Extension Development Host window you just need to press F5.

Some concepts are crucial to writing extensions in VS Code: the Activation Events which are events that when they happen the extension becomes active; the Contribution Points which are static declarations that you make in the package.json used to extend various functionalities within VS Code; the VS Code API which is a set of JavaScript APIs that you can invoke in your extension code. In general, to extend VS Code's functionalities you need to use a combination of Contribution Points and VS Code API.

Each VS Code extension has a package.json as its Extension Manifest. It contains a mix of Node.js fields, Activation Events, and contributes. The extension also has an entry file, estension.js, that exports two functions: activate and deactivate. Activate is executed when your Activation Event happens, while deactivate, which is not mandatory, cleans up before your extension becomes deactivated.

## 4.5 Tree View API

The Tree View API is useful for this dissertation because it allows extensions to show content structured as a tree in the Sidebar. To implement a Tree View, the first step is to add a Contribution Point using the contributes.views in the package.json file. The second step is to provide data to the view you registered so that it can be displayed. This can be done by implementing a TreeDataProvider. There are two necessary methods in this API that you need to implement: getChildren(element?: T): ProviderResult<T[]> that returns the children for the given element or root (if no element is passed); and the getTreeItem(element: T): TreeItem | Thenable<TreeItem> that returns the UI representation (TreeItem) of the element that gets displayed in the view. The third and final step is to register the above data provider to your view.

## 4.6 Mockups

As a first step in the planning of the development of the VS Code extension, Mockups of the user interface were drawn. The purpose of these was to come up with a preliminary, simple method to visualize what the end product might look like. These were useful for streamlining the extension development, as they allowed the early identification of possible difficulties and the trimming down of unnecessary features.

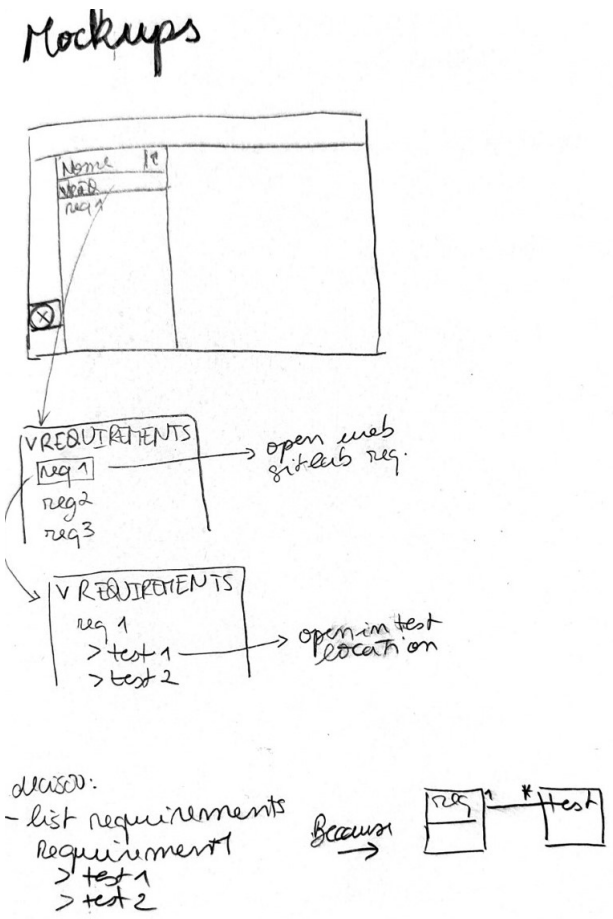


Figure 4.2: Example of one of the Mockups drawn

## 4.7 Pre-existing tools - GitLab Workflow extension

To develop an extension that would comply with the previously mentioned objectives in 4.1, a search of pre-existing tools that had similar goals was conducted. GitLab Workflow was the extension that had more functionalities that aligned with this thesis' objectives. This extension is open source, hosted on GitLab, and integrates GitLab into Visual Studio Code.

GitLab Workflow already enables: (1) the visualization of issues, comments, merge request details, and changed files in VS Code; (2) the direct creation of merge requests and reviews in the editor; (3) filtered searching; (4) local command-enabled Gitlab CI/CD configuration validation; (5) the autocompletion of CI/CD variables; (6) the creation, insertion and patch application of snippets; (7) the management and status visualization of pipelines; (8) the cloning of Gitlab Projects through integration with the built-in Git Extension; (9) read-only Gitlab repository browsing directly in VS Code without necessitating cloning; and, finally, (10) authentication.

In the Extensions repository an architecture.md file [1] describes the architecture of the extension and gives a high-level overview of its main components. It was an important file to better understand how to place my work in the codebase.

## 4.8 GitLab Workflow Further Development

This project improved upon and further developed the GitLab Workflow extension functionalities and enabled: (1) live requirement visualization, management, and software system traceability based on a listing of the project's requirements directly in an instantly auto-updating tool window of the IDE; (2) the linking of the requirements written in Gherkin to the respective tests generated through Cucumber;

You can define custom queries in your VS Code configuration by defining a search expression in the file settings.json. These expressions create the list of items shown in the Issues and Merge requests section of the sidebar of Visual Studio code. The default custom queries are defined in the extension package.json. The default is to only show the issues "assigned to me" or "created by me", which is not ideal since it doesn't allow you to have an overall picture of the project requirements and you wouldn't be contextually aware of all the project changes. In this case, to see them you would have to manually look them up on the GitLab web page. Therefore, for point (1) discussed above, a custom query to show all the project issues/requirements was included in the Sidebar.

The extension will be applied to projects created with the use of Gherkin and Cucumber. This type of project's characteristics was explored in Chapter 2 and will be further explained in Chapter 5. Basically, in Cucumber, .feature files are stored in the src/test/features/ folder and these can contain one or more scenarios to test the requirements. These scenarios are what we want to link to the project's Issues. Another important thing is that the .feature files have names that are related to their corresponding requirement, making it easy to associate them. With the GitLab Workflow extension, we already have access to the working project's file system. Therefore, the next step is to parse the .feature files to get each Scenario for each Feature and next to add below each listed Issue in the Sidebar an item for the Scenarios found. This can be done by adding a new type of TreeItem that compose the children of the requirements TreeItem. These steps were not chosen to be implemented yet, but to be in future work, and are the ones that make the point (2) discussed above possible.

## 4.9 Summary

The development of an extension for VS Code [25] that integrates GitLab [12] and links requirements to their tests are described in this Chapter. The primary purpose of the extension is to integrate GitLab features into the editor, so users don't have to leave the editor to perform basic tasks such as reading an issue description. The extension is trying to plug in the GitLab features into an existing VS Code Extension API to both minimize the need for custom code and to make the experience as VS Code-like as possible. This allows for Requirements Visualization and Management and ensures its Traceability.

# Chapter 5

# Evaluation

**Contents**



In this chapter, the results of this dissertation evaluation are explored. It includes a description of the extensions user's project that will serve to test the developed extension. As well as a survey to access the proposed solution.

## 5.1 Extension User's Project

To test the extension, it was applied to a project. For confidentiality reasons, no identifying details of the project can be disclosed. Consequently, from here on it will be referred to as Project X. Project X was chosen as the ideal project to apply the extension to because it follows a BDD philosophy and was created using Gherkin and Cucumber ( 2.1.3).

In Cucumber, an example is called a Scenario. Scenarios are defined in .feature files, which are stored in the src/test/features/xxxx directory (or a subdirectory) (Figure 5.1 located at the end of this Chapter). The necessary steps for the .feature file to pass the test are in the $xxxxxx_{stepdefs.jsfile}$.

According to Gherkin rules, the first line of the file in Figure 5.2 (located at the end of this Chapter) starts with the keyword Feature: followed by a name. The second line is a brief description of the feature. Cucumber does not execute this line because of its documentation. The line

Background: Given a XXX with id XXXand pin XXX exists establishes a condition valid for the whole feature, making it unnecessary to repeat that step for every Scenario. The lines Scenario: Correct id and pin and Scenario: Incorrect pin are concrete examples illustrating how the software should behave, in these examples either a situation where the software's user inputs a valid id and pin and another where the pin is incorrect. The Scenario instructions are what Cucumber will execute and, thus, verify its correctness.

To evaluate the usefulness of this extension, this project must be managed with GitLab. For that to happen, it's necessary to create a GitLab repository and to list all its requirements by adding issues with User-Stories as titles (which can later be assigned to the developer responsible for implementing them). The final step necessary to test the extension is to open VS code (if already installed, if not install it first) and clone the GitLab project and open it.

## 5.2 Survey on Live Requirements Visualization, Management and Traceability

To better understand how to develop a useful extension, as well as develop an extension as close to the ideal one as possible, a survey on Live Requirements Visualization, Management, and Traceability was constructed. Two developers with experience in Requirements Engineering were asked to answer it. This survey was also useful to complement the State of the Art section of this work (previously discussed in Chapter 2).

### 5.2.1 Rationale

Since the aim of this survey was to guide the development of the extension, it was designed as a tool to gather data about a potential, idealized extension, and to confront it with the real extension that was developed.

### 5.2.2 Design

The survey was structured as having three sections: (1) Characterization - this is, simply, the section where basic demographic data from the survey participant was gathered, as well as their role in software development teams; (2) General Perspectives - in this section, the objective was to understand what the participant's views on the Requirements' role in SDLC, software management and testing, and relevant timing of testing. (3) Extension Expectations - finally, the survey would focus on the perceived usefulness of a VS Code extension that would allow for Requirements management and traceability.

### 5.2.3 Questions

Below is a transcription of the Survey:

Please answer the following questions:

>Characterization

1. What roles have you had in a software development team?

2. How many years of experience as a software developer do you have?

3. What work methodology, platforms (Jira, GitLab (...)) and testing methods do you have experience with?

>General Perspectives

1. How important do you think defining Requirements is in the Software Development Life-Cycle (SDLC)? Why?

2. In your opinion, is the timing of software testing important? In the context of the workflow of SDLC, is Requirement testing something that should be done continuously, at defined points in the development process, or just after all the coding is done? Why?

3. Do you use any tools to track Requirements and make sure the workflow is respecting them? Which ones?

4. Regardless of whether you use Requirements tracking tools, do you believe such a tool is important or useful? Why?

>Extension Expectations

1. Do you think your current or past teams would benefit from using a VS Code extension that would allow for Requirements visualization? What about their management and traceability?

2. Would you use an extension that would inform you in real-time when Requirements stop being respected? Why/why not?

3. As someone with experience in the area, what is your opinion on the proposed extension you were shown? How would you improve it?

### 5.2.4 Survey Results

In this section, the results collected from 2 participants of the survey are presented as well as some knowledge about themselves. Both are software developers and have more or less one year of work experience. Another characteristic was that the two had already worked with a Scrum methodology, but one of them had a background in Extreme Programming as well. About the platforms that they had contact with, GitHub and GitLab were mentioned, and one of them also included Jira. Finally, they were also asked about the type of testing methods they had worked with, one of them only named tests done by clients, and the other named unit testing, mock testing, mutation testing, and system testing.

Regarding their opinions about the SDLC, they were asked about the different phases. Generally, they thought that defining Requirements in the SDLC is a crucial phase. Without it, the final product goals aren't defined and if they aren't defined the development team won't know what must be done. Thus, this could represent a huge setback to the overall project and, could, later on, invalidate it. They were also of the opinion that Requirement testing is something that should be done continuously, at defined points in the development process. Continuous testing was considered very important because it brings robustness to the project and the earlier an error is detected, the easier it is to fix it.

Requirements tracking tools were also discussed and the tools they currently use to track Requirements are Jira, GitHub, and Gitlab. They believe that it's important to use them because it is important to know what the Requirements of the project are and who in the team is working on one of them. For them, keeping track of the requirements is a must. If not, forgetfulness, misunderstanding, and issues of the sort are very likely to happen. Nevertheless, in their experience, most developers don't like to spend much time on these types of tools updating what they are working on, or sometimes, they simply forget to use them. When asked about it, they expressed that their current or past teams would benefit from using a VS Code extension that would improve Requirements visualization, management, and traceability. They showed interest in using an extension that would inform them in real-time when Requirements stop being respected was, that way they would not have to leave their working environment to check if anything has changed. In their experience, most developers waste valuable development time checking the management tools and if there was a tool that could facilitate the visualization of the Requirements in the development environment it would make the process much quicker, and it would be much more efficient.

After being presented with this thesis's proposed solution the extension seemed like a really good solution to improve efficiency in a development team and a great solution for the problem at hand. When asked about what other improvements they would include to the extension, they noted that at the moment couldn't think of anything else.

## 5.3 Summary

This section considers the participants' general impressions of the VS Code tool that they evaluated. As expected, participants recognized, in general, that it would positively impact their development workflow. Testing is a process that serves as the basis for linking requirements with other SDLC activities and the facilitation of this was seen as a big improvement for development teams. Overall, they had a positive opinion about the utility and user experience of the tool.

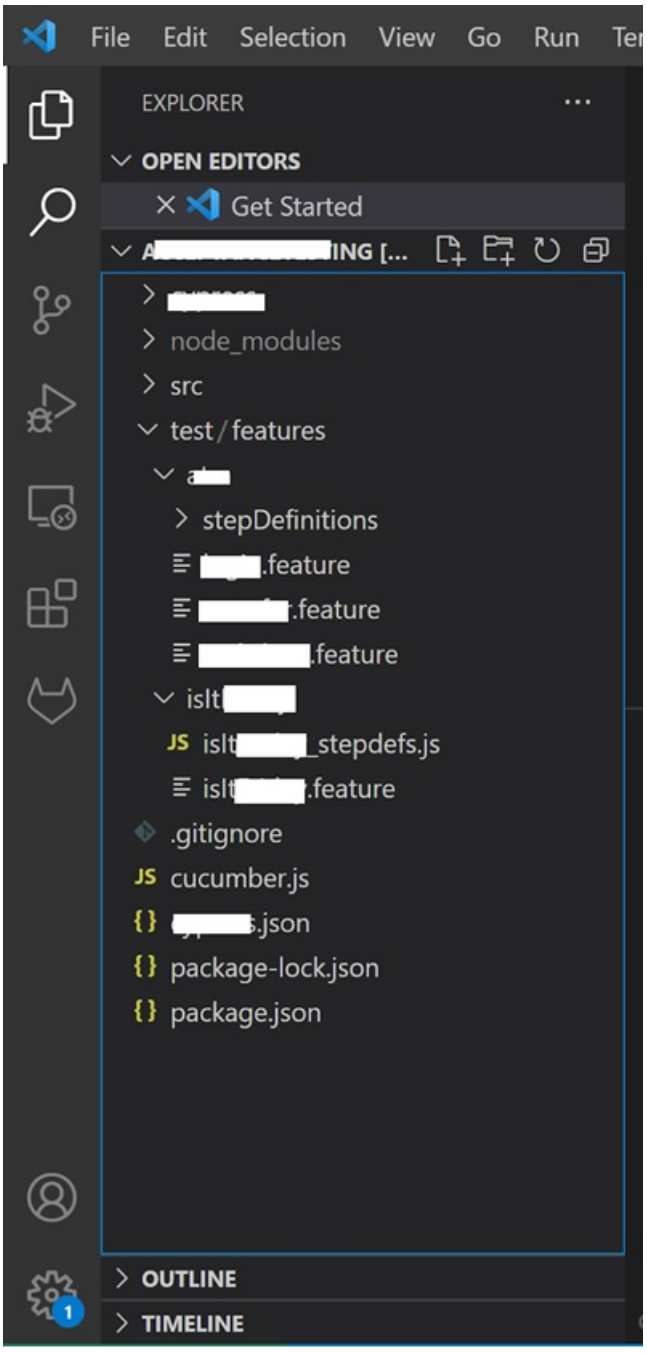


Figure 5.1: File System Structure

,

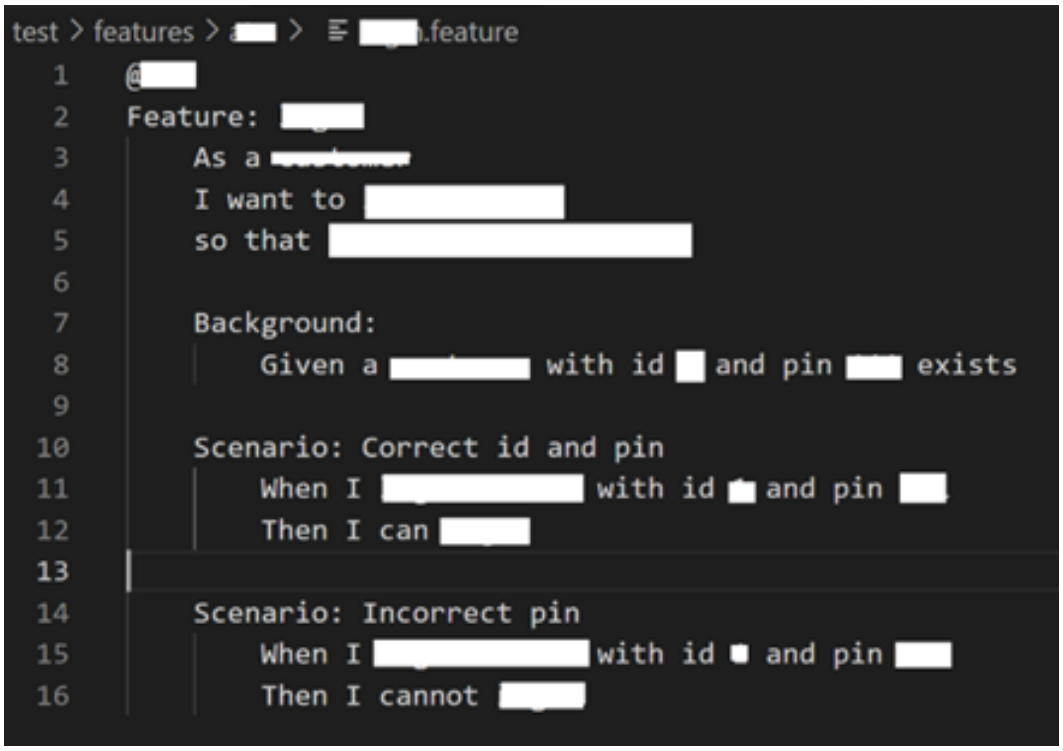


Figure 5.2: User's Project Feature

,

# Chapter 6

# Conclusions and Future Work

**Contents**



In this chapter, conclusions about live feedback are made and the expected results of the dissertation are anticipated

## 6.1 Contributions

Communication is decisive in creating good software. In fact, without feedback loops to validate and get information about the software and its course, the SD process is more limited. Getting evaluations and then instantly supplying them back to the process quickens and improves its overall evolution. As such, adding liveness to practices that already exist would improve their efficiency and stability, benefiting the SD process.

After the examination of relevant projects and application domains, and the identification of their present level of liveness, this dissertation proposes a solution to close the liveness gaps. Indeed, using this valuable data, ways to reach greater liveness levels are designed. Tools that help in coping with software challenges were presented, and evidence was provided for how LiveSD can help by better integrating requirements traceability and management actions into the work environments.

## 6.2 Expectations

With this work, it is expected an improvement to the current SDCs at Vodafone. The proposed VS Code extension will contain a set of instruments, capable of bringing insights into the software system at hand. Thus, one can achieve the objective of visualizing, managing and tracing requirements.

This solution is also designed to meet Vodafone's expectations and goals. The company has an interest in providing their development teams with more fitting tools to assist their work, which, in the long run, is essential to maintaining customer relations and building their trust.

## 6.3 Future work

Beyond its use at Vodafone, the result of this thesis can be important in assisting students and other companies, by making sure their time is being used for crucial tasks and not being simply wasted with aspects that can be automated. Also, the extension developed for this thesis can be further developed with more functionalities as their need arise. Furthermore, only some of the potential advantages of this form of developing software were explored in-depth. The LiveSD domain is still being tested, and in the future, more investment will certainly be dedicated to these types of practices.

5

# References


[1] gitlab-vscode-extension repository architecture.md. Available at https://gitlab.com/gitlab-org/gitlab-vscode-extension/-/blob/main/docs/developer/architecture.md, Accessed last time in September 2022, 2022.

[2] Ademar Aguiar, André Restivo, Filipe Figueiredo Correia, Hugo Sereno Ferreira, and João Pedro Dias. Live software development: Tightening the feedback loops. In *Proceedings of the Conference Companion of the 3rd International Conference on Art, Science, and Engineering of Programming*, Programming '19, New York, NY, USA, 2019. Association for Computing Machinery.

[3] Atlassian. Atlassian agile coach - scrum. Available at https://www.atlassian.com/agile/scrum, Accessed last time in March 2022, 2022.

[4] Atlassian. Atlassian agile coach - user stories. Available at https://www.atlassian.com/agile/project-management/user-stories, Accessed last time in March 2022, 2022.

[5] Atlassian. Scrum framework. Available at https://www.scrum.org/resources/scrum-framework-poster, Accessed last time in March 2022, 2022.

[6] Atlassian. What is a developer in scrum? Available at https://www.scrum.org/resources/what-is-a-scrum-developer, Accessed last time in March 2022, 2022.

[7] Atlassian. What is a product owner? Available at https://www.scrum.org/resources/what-is-a-product-owner, Accessed last time in March 2022, 2022.

[8] Atlassian. What is a scrum master? Available at https://www.scrum.org/resources/what-is-a-scrum-master, Accessed last time in March 2022, 2022.

[9] Atlassian. What is scrum? Available at https://www.scrum.org/resources/what-is-scrum, Accessed last time in March 2022, 2022.

[10] Sundramoorthy Balaji and M Sundararajan Murugaiyan. Waterfall vs. v-model vs. agile: A comparative study on sdlc. *International Journal of Information Technology and Business Management*, 2(1):26–30, 2012.

[11] GitLab B.V. Gitlab agile planning. Available at https://about.gitlab.com/_nuxt/image/d326c1.png, Accessed last time in September 2022, 2022.

[12] GitLab B.V. The one devops platform | gitlab. Available at https://about.gitlab.com/, Accessed last time in September 2022, 2022.

[13] GitLab B.V. Why gitlab. Available at https://about.gitlab.com/why-gitlab/, Accessed last time in September 2022, 2022.

[14] Ward Cunningham. Agile manifesto. Available at https://agilemanifesto.org/, Accessed last time in March 2022, 2022.

[15] Carlos Nova Duarte. Improving requirements engineering with live software development environments, 2021.

[16] Shokhista Ergasheva and Artem Kruglov. Software development life cycle early phases and quality metrics: A systematic literature review. *Journal of Physics: Conference Series*, 1694(1):012007, dec 2020.

[17] Andrew Fischer. Introducing circa: A dataflow-based language for live coding. In *2013 1st International Workshop on Live Programming (LIVE)*, pages 5–8. IEEE, 2013.

[18] international community of open-source developers. Pharo. Available at https://pharo.org/, Accessed last time in March 2022, 2022.

[19] JavaTpoint. Requirement engineering. Available at https://www.javatpoint.com/software-engineering-requirement-engineering, Accessed last time in March 2022, 2022.

[20] Juraj Kubelka, Romain Robbes, and Alexandre Bergel. The road to live programming: insights from the practice. pages 1090–1101, 05 2018.

[21] Inc Meta Platforms. React. Available at https://reactjs.org/, Accessed last time in March 2022, 2022.

[22] Microsoft. architecture-containers.png. Available at https://code.visualstudio.com/assets/api/ux-guidelines/examples/architecture-containers.png, Accessed last time in September 2022, 2022.

[23] Microsoft. Gitlab workflow. Available at https://marketplace.visualstudio.com/items?itemName=GitLab.gitlab-workflow, Accessed last time in September 2022, 2022.

[24] Microsoft. Ux guidelines | visual studio code extension api. Available at https://code.visualstudio.com/api/ux-guidelines/overview, Accessed last time in September 2022, 2022.

[25] Microsoft. Visual studio code. Available at https://code.visualstudio.com/, Accessed last time in March 2022, 2022.

[26] Stephen Oney, Brad Myers, and Joel Brandt. Euclase: A live development environment with constraints and fsms. pages 15–18, 05 2013.

[27] Stephen Oney, Brad Myers, and Joel Brandt. Interstate: A language and environment for expressing interface behavior. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, page 263–272, New York, NY, USA, 2014. Association for Computing Machinery.

[28] Ken Schwaber and Jeff Sutherland. Itl bulletin the system development life cycle (sdlc). Available at https://csrc.nist.gov/csrc/media/publications/shared/documents/itl-bulletin/itlbul2009-04.pdf, April 2009.

[29] Ken Schwaber and Jeff Sutherland. The scrum guide. Available at https://scrumguides.org/scrum-guide.html, November 2020.

[30] SmartBear Software. Cucumber. Available at https://cucumber.io/, Accessed last time in March 2022, 2021.

[31] Steven L Tanimoto. A perspective on the evolution of live programming. In *2013 1st International Workshop on Live Programming (LIVE)*, pages 31–34. IEEE, 2013.

[32] Evan You. Vue.js. Available at https://vuejs.org/, Accessed last time in March 2022, 2022.