# An automation-based strategy for monitoring software requirement changes

Jose Souza
*Requirements Apps*
*Sidia Instituto de Ciência e Tecnologia*
Manaus, Brazil
jose.diogo@sidia.com
0009-0007-9714-9520

Rennan Salgado
*Requirements Apps*
*Sidia Instituto de Ciência e Tecnologia*
Manaus, Brazil
rennan.salgado@sidia.com
0009-0003-3068-4386

Edluce Veras
*Requirements Apps*
*Sidia Instituto de Ciência e Tecnologia*
Manaus, Brazil
edluce.leitao@sidia.com
0009-0005-8222-5653

Bruno Ferreira
*Requirements Apps*
*Sidia Instituto de Ciência e Tecnologia*
Manaus, Brazil
bruno.cardoso@sidia.com
0009-0001-4384-5025

Priscila Barros
*Requirements Apps*
*Sidia Instituto de Ciência e Tecnologia*
Manaus, Brazil
priscila.barros@sidia.com
0009-0008-9082-2871

Lyan Mota
*Requirements Apps*
*Sidia Instituto de Ciência e Tecnologia*
Manaus, Brazil
lyan.santiago@sidia.com
0009-0007-6558-4122

Jullianne Abreu
*Requirements Apps*
*Sidia Instituto de Ciência e Tecnologia*
Manaus, Brazil
jullianne.abreu@sidia.com
0009-0001-6153-5418

Fabio Ramos
*Requirements Apps*
*Sidia Instituto de Ciência e Tecnologia*
Manaus, Brazil
fabio.coelho@sidia.com
0009-0005-5488-3552

***Abstract*—The definition and management of software requirements are essential for the development team to maintain coherence and deliver the expected functionality to the customer in a global software development environment. These requirements define and guide the necessary parameters for the software to function properly. However, during the software lifecycle, requirements may change after initial deliveries or in projects with continuous development. Managing these requirements can become a complex activity, especially when multiple stakeholders and multiple systems are involved during the development process. In this article, we propose an automated software solution to collect changes in requirements using a system used by the customer. This will allow the opening of automatic tasks to notify the requirements management team and the development team appropriately. In this way, the team can analyze the collected changes, conduct detailed assessments to analyze the impacts and the necessary implementations to integrate the change into the software as quickly and coherently as possible.**

***Keywords*—software requirements; monitoring software; change monitoring; requirement changes**

## I. Introduction

Monitoring requirements is a complex process that mainly involves changes to requirements throughout the project's development. Automating this monitoring continuously, whether creating, modifying, or removing requirements, is becoming an increasingly adopted practice in software development projects. Since the first meeting until production, more than 50% of these requirements undergo changes. [3]

With the advancement of technology, people increasingly require systems to meet various needs such as resource management, health, and security. Although there is a wide range of system solutions available, there are still many challenges facing their projects. Most of these problems stem from errors in the requirements management process; some of these problems can be attributed to the fact that people were not keeping track of which requirements were added to the project flow. Studies have shown that implementing manual or automated monitoring practices for these requirements in reference models of software development produced satisfactory results. [1]

Requirements management needs a special attention in the development process of a system, encompassing highly complex processes such as change control, version control, status tracking, and requirement tracing. The importance of each part of the process is crucial, and managing these requirements includes documenting dependencies among requirements, controlling changes over identifiers, and correcting inconsistencies

ISSN 2447-8326. V.1 

between the requirement and project artifacts. [4] [6]

In software development methods, modifications have been frequent to produce faster and more reliable systems. It is more advantageous to automate the inclusion, modification, or removal of these requirements from a project, making it an essential part of successful software creation. [4]. In this context, it is predicted that the use of tools incorporating artificial intelligence, machine learning, among other techniques, will contribute significantly to reducing development efforts. [12]

In the next sections, the concepts used for the development and deployment of this system will be discussed, as well as the benefits that continuous requirement monitoring can offer if a routine of continuous monitoring is adopted in development process. This takes into account software that requires continuous integration of requirements, showing the proposed architecture of the solution and providing a detailed explanation of how the algorithm governing the system works, followed by a brief discussion of the gains and challenges provided by the implementation of this system.

## II. Metodology

Different business rules needed to be considered so that tasks previously performed manually become automated in a software development context.

Due to the large scale of projects developed, one of the factors considered was the use of automated solution for multiple teams because, in the manual process, the information was decentralized and difficult to standardize.

Furthermore, this type of process requires constant checks on the external source, making manual execution even more susceptible to failures. Any inconsistency in requirement monitoring can compromise the quality of the final software and, consequently, fail to meet the customer's expectations [2]

In the next section, the architecture of the proposed solution will be presented, detailing generally what is done at each stage of monitoring.

### A. Solution Architecture

The architecture of the proposed solution is divided into three parts: monitoring, processing, and notification, as shown in Fig. 1, the following subsections detail the operation of each of them, explaining the concept used for the proper functioning of the solution and the reason for the approach taken.

*1) Monitoring:* This work aims to monitor an external information source where requirements related to software projects developed by the team are made available. Such an information base follows a similar concept to an online forum, allowing the addition of texts and attachments, this set of information added by users is called "Post". Each post corresponds to a customer's demand for including, removing, or changing a specific requirement. Demands with different scopes can be organized into what are called "workspaces", where several posts related to a specific subject are grouped together to maintain better organization. Thus, when searching for information in the external source, only those of interest were defined to obtain specific information.

The idea behind the proposed architecture is to monitor the pre-defined workspaces' content while maintaining a constant interval to keep the database's requirement information updated and not miss any posted information from the external source. The defined interval for verification is crucial for the requirement to be analyzed as soon as possible by the development team and integrated into the software rapidly, maintaining a continuous integration flow. On the other hand, this interval should not be too short to avoid excessive access to the external source, potentially overwhelming the server. [9]

*2) Processing:* The search for requirement-related information is performed by checking and filtering the listing of posts present in the workspaces. This process only searches for information about requirements because even though the system is organized into different workspaces, it is possible for a post to be mistakenly inserted along with others related to requirements. After a more accurate listing of the posts, they are filtered to be better identified during the notification process, as depending on their type, different treatments may be required.

From the already filtered posts, the URL extraction is performed for each of them. The URL is the electronic address that will be used to request its contents using extraction techniques such as Web Scraping.

With the content of each post already available, it is necessary to structure and index the data so that it can be easily accessed during the notification process, making use of some data structure present in the programming language used for the system's development.

*3) Notification:* Once the contents of each post and their identifiers are obtained, it is possible to verify the database to identify if a notification related to the post has already been created. This process uses the identifier (ID) of the post and scans the database containing all notifications.

Upon identifying that the post has not yet been notified, another processing step begins, where the previously indexed data is collected, and a notification is built from this data, considering each type of post and following an appropriate treatment routine.

If a post has already been notified before, the found notification is analyzed and updated based on the new content found, preserving the previous content, which helps in the traceability of the requirement as there will be a history of changes present in the notification.

The historical record is important for monitoring the life cycle of the requirement and enabling analysis of its changes over time, contributing to ensuring quality and efficiency during software development.

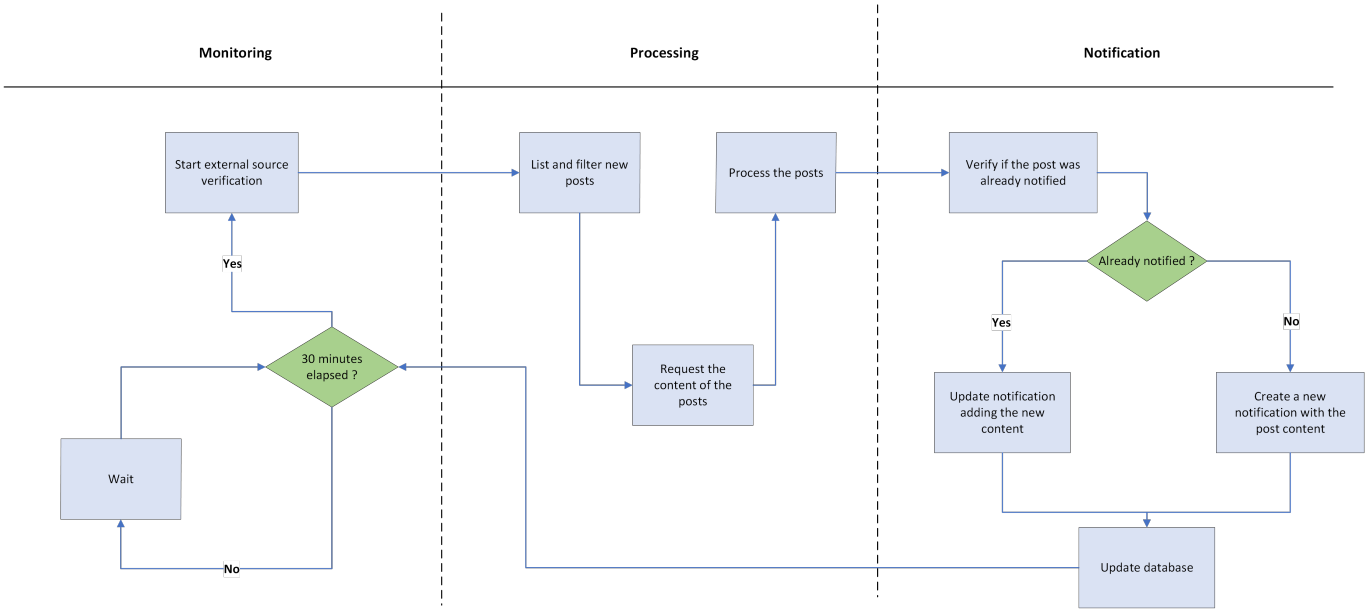


Fig. 1. Monitoring module architecture

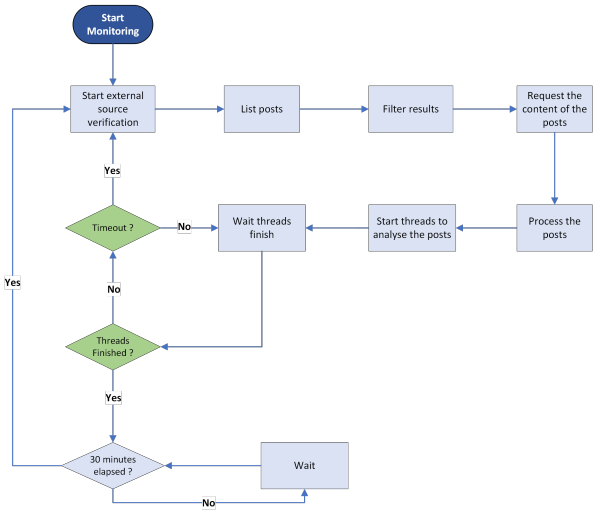


Fig. 2. Flowchart of the monitoring operation

### B. *Requirement monitoring flow*

To keep the database updated with the most recent requirements present in the external source, it is necessary to constantly check for the arrival of a new requirement. To make this happen efficiently, it was necessary to implement a specific library for requesting and handling data from the external source. This library was designed exclusively for this purpose and includes routines for listing, converting, and organizing content into simplified structures that can be easily accessed.

In this context, new requirements arise from new posts or updates that can range from editing the description to include a new attachment also, the system needs to check if the update is relevant or not.

The monitoring process starts with the creation of a main thread responsible for verifying the external source and has a timeout of thirty minutes, set based on previous tests. This prevents the system from getting stuck in some part of the algorithm due to infrastructure problems such as the external source server crashing or connection issues. However, under normal conditions, this time limit is never reached, and the verification is performed correctly. The functioning of the monitoring process is demonstrated in the flowchart in Fig 2.

Threads allow different parts of a program to run simultaneously, sharing resources such as memory and variables. They are useful for performing multiple tasks at once, improving performance and efficiency. Threads are the smallest unit of processing that the operating system can execute independently. [11]

The use of threads brings efficient concepts for continuous improvement and accelerated performance in executing tasks within the monitoring system, guaranteeing the quality of the structure of these tasks. Parallelism allows monitoring multiple tasks simultaneously, making better use of processor resources, while responsiveness ensures that the system can continue to operate and respond to new tasks during monitoring. Additionally, efficiency is increased by reducing waiting time, as different tasks can be monitored simultaneously, preventing the system from idling while waiting for a task to complete. [8]

The mentioned benefits are particularly important in monitoring systems, where agility and the ability to handle multiple tasks are crucial for the efficiency and effectiveness of continuous monitoring.

The thirty-minute time is also used as the interval between checks of the external source. In the normal case where the verification thread is terminated before the time limit, the system waits for the interval to complete, making use of the sleep routine present in the time library native to Python, which causes the system to pause execution for a specified period. [7]

Based on the previously defined workspaces where the requirements we want to monitor are organized, the last posts of each workspace are listed. With this first listing, it is already possible to filter requirements that do not fit certain predefined criteria and organize posts that contain a specific requirement that should follow a different treatment during the notification process.

For each publication present in the obtained lists, a thread responsible for extracting the information is started, using the internally developed library. To request the content, the library makes use of a common technique in the tech industry called Web Scraping. This method aims to automatically collect considerable amounts of unstructured data from websites and then convert it into an easy-to-manipulate format. [5]

During the processing of each post, the collected data is treated and organized using a common Python structure called Dict, which is mutable, unordered, and utilizes the concept of key-value pairs. [10]

The content is divided into keys such as ”author”, ”date”, ”title”, ”description”, and other specific information, so the values can be easily accessed using the respective key.

The notification process starts with the analysis of each post’s content. For this, a thread is initiated for each post, following a specific routine for each filtered and organized post type. The content verification routine differs for each post type, where only relevant information for each type is handled and displayed in the notification.

In the standard notification flow (Fig 3), the system checks if the post has already been notified in a previous check and is in the system’s database. Based on this information, the system decides whether to create a new notification or update an existing one.

In the process of creating a new notification, the relevant information from the post, that was indexed during processing, is read and organized so that it can be presented consistently. Then, to aid in tracking, the system adds a comment to the original post, indicating the ID of the generated notification.

On the other hand, when encountering a post that has already been notified and there is an update, the previous notification is processed to identify the differences between the previous and current content and the differing data is highlighted in notification. At the end of this step, a comment is added to the post by the system just like it is done after creating a new notification.

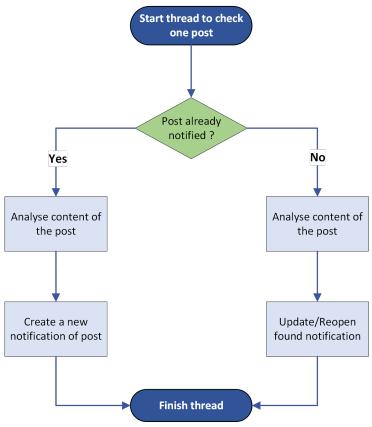


Fig. 3. Flowchart of the analysis thread functionality

## III. Lessons Learned

The deployment process and running the system in a production environment brought a series of improvements and challenges.

Centralization of Information: As foreseen during the requirements gathering phase, the decentralization of information across teams was a major challenge in implementing the system. Even though it required additional effort among the teams to adapt to the new format, establishing a centralized and standardized database was crucial for the correct functioning of the software.

Version Control of Requirements: Storing different versions proved to be a very effective strategy for software requirement traceability by the teams using the tool. The implemented approach showed efficiency in managing large-scale projects prone to frequent changes.

Monitoring: As presented in Section 2, the monitoring process was primarily built using Python libraries for data extraction and thread management. While capable of performing constant information verification tasks, this approach makes the application less scalable and difficult to maintain. Currently, there are more mature tools specifically designed to cater to this type of scenario (such as Apache Airflow, Dagster, and Prefect). Furthermore, in a potential refactoring of the application, it would be worth assessing the feasibility of utilizing external tools for these purposes.

Observability: During the design of the system, scenarios where failures could occur during any of the sub-processes constituting the monitoring were not considered. Since it is an application designed to operate automatically without any human interaction, tracking unwanted behaviors became a challenging task.

## IV. Conclusions and Suggestions

This paper presented the development process of a continuous software requirement monitoring application, highlighting the main components of its architecture and the lessons learned during the project and deployment of the tool.

Overall, the developed application met the primary objective proposed in its conception: to enable automatic monitoring of changes in software requirements. Among the main positive impacts of the proposed solution are the versioning of software requirements managed by the tool's users and the reduction of errors caused by human failure. However, it is worth noting that the main areas for improvement encountered in the deployment and maintenance process of the developed tool were related to its architecture. As a suggestion for a possible refactoring or construction of a new solution with the same purpose, it is believed that the utilization of specialized tools in task execution and scheduling can ensure greater robustness and maintainability to the application.

As a suggestion for future work, with the requirements history already stored in a database, there is also the possibility of integration with Artificial Intelligence models to automate processes that are currently still performed manually by the team, such as redirecting new requirements based on the team's scope and pre-analysis of new requests, in order to make these processes more efficient.

## References


[1] Barros, R. A Importância da Gestão de Requisitos para Projetos de Desenvolvimento de Software. B.S Thesis, Campus São Paulo (IFSP), Instituto Federal de Educação, Ciência e Tecnologia, São Paulo, Brazil, 2018.

[2] Jayatilleke, S & Lai, R. A systematic review of requirements change management. Information and Software Technology. pp 163-185. 2018.

[3] Kotonya, G. & Sommerville, I. Requirements engineering: processes and techniques. Wiley Publishing, 1998.

[4] Machado, F. Gestão de Requisitos de Software: Onde nascem os sistemas. Editora Érica. 2018.

[5] Mitchell, R. Web Scraping with python Collecting more data from the modern web. O'Reilly Media. 2018.

[6] Pressman, R. & Maxim, B. Engenharia de software-9. McGraw Hill Brasil, 2021.

[7] Python Software Foundation. Time: time access and conversions. Available: <https://docs.python.org/3/library/time.html>. Accessed: 02/07/2024

[8] Silberschatz, A. Galvin, P. B. & Gagne, G. Operating System Concepts. Wiley Publishing. 2018.

[9] Sommerville, I. Engenharia de Software. Pearson. 2020.

[10] Sturtz, J. Dictionaries in Python, Real Python. Available: <https://realpython.com/python-dicts/>. Accessed: 02/07/2024.

[11] Pusukuri, K. Gupta, R. & Bhuyan, L. Thread reinforcer: Dynamically determining number of threads via OS level monitoring. 2011 IEEE International Symposium on Workload Characterization (IISWC). *IEEE*, 2011

[12] Umar, M & Lano, K. Automated Requirements Engineering in Agile Development: A Practitioners Survey. 2023 3rd International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME). *IEEE*, 2023