www.infosupport.com

Rapport

Project Report

Improving Azure Pipelines DX: A Smarter DevOps Experience

Serggio Pizzella

14 January 2025

Draft

Solid Innovator

Info Support B.V.

Kruisboog 42
3905 TG Veenendaal (NL)
www.infosupport.com

K.v.K 3013 5370
BTW NL8062.30.277B01
IBAN NL92 RABO 0305 9528 89
BIC RABONL2U

Hoofdkantoor
Tel. +31(0)318 - 55 20 20
info.nl@infosupport.com

Kenniscentrum
Tel. +31(0)318 - 50 11 19
training.nl@infosupport.com

Project Report

Improving Azure Pipelines DX: A Smarter DevOps Experience

Details

Title	Project Report
Project	Improving Azure Pipelines DX: A Smarter DevOps Experience
Version	1.0
Status	Draft
Date	14 January 2025

GRADUATION-INTERNSHIP PORTFOLIO BACHELOR-ICT

FONTYS UNIVERSITY OF APPLIED SCIENCES

Student:	
Family name , initials:	**Pizzella Ricci, S.V.**
Student number:	**4449681**
project period: (from – till)	**02-09-2024 – 03-02-2025**
Company:	
Name company/institution:	**Info Support**
Department:	**Business Unit Industry**
Address:	**Kruisboog 42, 3905 TG Veenendaal**
Company mentor 1:	
Family name, initials:	**Thissen, N.**
Position:	**Afstudeer Coodinator**
Company mentor 2:	
Family name, initials:	**Horsman, L.**
Position:	**IT Consultant**
University teacher:	
Family name , initials:	**Schriek, E.**
Final portfolio:	
Title:	**Improving Azure Pipelines DX: A Smarter DevOps Experience**
Date:	**10-01-2025**

Approved and signed by the company mentor: Luuk Horsman

Date: 14-01-2025

Signature: [illegible]

History

Version	Status	Date	Author	Updates
1.0	Draft	06-01-2025	Serggio Pizzella	Creation
1.1	Draft	13-01-2025	Serggio Pizzella	

Distribution

Version	Status	Date	To
1.0	Draft	11-01-2025	First assessor
1.1	Final	14-01-2025	

Content

1. Context ... 6

1.1 How to read ... 6
1.2 Company ... 6
1.3 Problem statement ... 6
1.4 Research ... 7
1.5 Approach (local) ... 8

2. Conclusion ... 9

2.1 Research ... 9
2.1.1 How do Azure DevOps pipeline files function, what are their key components and what are the processes that result in their execution? ... 9
2.1.2 What are the common mistakes and errors occurring in Azure DevOps pipelines, and how can they be identified? ... 9
2.2 Which static analysis techniques and tools are best suited for detecting mistakes in Azure DevOps pipelines ... 9
2.2.1 How can a static analysis tool be developed to integrate into development workflows while ensuring high performance and compliance with internal guidelines? ... 10
2.3 Recommendations ... 11
2.3.1 Integrate with Microsoft's Existing Azure Pipelines Language Server ... 11
2.3.2 Embed Functionality Directly into GitHub Runner ... 11

3. Process ... 12

3.1 Planning Phase ... 12
3.1.1 Project Initiation & Early Uncertainties ... 12
3.1.2 Defining the Project Scope ... 12
3.1.3 Finalizing the Project Plan ... 12
3.2 Research Phase ... 13
3.2.1 Building Understanding of Azure Pipelines (Sub-question 1) ... 13
3.2.2 Identifying Common Mistakes (Sub-question 2) ... 13
3.2.3 Survey Analysis & Reporting ... 13
3.3 Development Phase ... 15
3.3.1 Initial Attempt: Focusing on Compile-Time Expressions ... 15
3.3.2 Revisiting the Project Plan & Scope ... 16
3.3.3 Implementing a Structured Workflow with Sprints ... 16

4. Reading Guide ... 18

4.1 Professional Products ... 18
4.1.1 Project plan (local) ... 18
4.1.2 User Requirements Specification (local) ... 19
4.1.3 Software Architecture (local) ... 20
4.1.4 Survey Report (local) ... 21
4.2 Additional Activities ... 22
4.2.1 Kanban board ... 22
4.2.2 Meeting notes ... 22
4.2.3 Progress Update Meetings ... 23

4.2.4 Graduation requirements 23
4.2.5 Company Training 24
4.3 Learning Outcomes 25

5. Evaluation and Reflection 26

1. Context

1.1 How to read

This report provides the most complete walkthrough of the entire project from start to finish. However there is an accompanying website (https://grad.serggio.dev). Most links point to the portfolio website for additional content; an additional 'local' link is sometimes provided, this link is relative, thus will only work in the submission folder.

1.2 Company

Info Support is a leading consultancy agency founded in 1986. Info Support provides consultancy services and custom-made software solutions to large, well-known companies such as OVPay, Albert Heijn, Jumbo, and Enexis. The company operates internationally, with over 800 employees across five companies. Info Support works across five key sectors: Agriculture & Food, Finance, Healthcare & Insurance, Industry, and Mobility & Public. In addition to its consultancy and software development services, Info Support also focuses on training both internal and external personnel. The company offers certification training in seven fields and provides specialized minors for students, helping them gain expertise during their education.

1.3 Problem statement

Developers often struggle with identifying and resolving errors in Azure DevOps CI/CD pipelines due to the lack of robust local validation tools for the YAML files defining these pipelines. Currently, the only method for validation is committing and testing changes directly in the Azure DevOps environment, which is time-consuming and disrupts workflow efficiency.

This absence of local validation creates repetitive and slow iterations, requiring developers to submit changes to version control, await feedback from CI runs, and address errors. The situation becomes even more challenging when pipelines incorporate multiple shared templates, each introducing unique parameters and considerations.

Common errors such as typos, missing or incorrect parameters, and misuse of variables exacerbate the problem. Each template can introduce its own variables, including nested ones, making the environment increasingly complex and unmanageable. Developers often face long wait times for CI jobs to run, slowing down even minor corrections, such as fixing a typo. This process: saving the file, committing changes with a message, pushing to the repository, waiting for CI to pick up the changes, and running the pipeline to locate the error; can take several minutes for each iteration. Consequently, trivial issues can result in significant delays, productivity losses and developer frustration.

Improving this workflow with real-time, in-editor feedback would drastically reduce these inefficiencies, enhance developer productivity, and reduce frustration.

1.4 Research

Throughout the assignment research will be conducted using the DOT framework. We primarily wish to create a tool that will eliminate or reduce pain points.

To manage the scope of the project we first need to find out what the main pain points even are, in the development process of Azure pipelines, and how they can be alleviated by using static analysis. In other words, how can we best alleviate frustration without running the pipeline.

The main question for the project will be:

How can static analysis be applied locally to Azure DevOps pipeline files to maximize mistake prevention before pipeline execution?

From this question stem the following sub-questions:

1. How do Azure DevOps pipeline files function, what are their key components and what are the processes that result in their execution? (Literature study)
   a. Dive through the Azure DevOps official documentation to understand the syntax and features available.
   b. Explore how Azure DevOps parses and executes pipeline YAML files.

   Outcome: Flowcharts or diagrams to visually represent the execution flow of a pipeline.

This question was ruled out during the course of the project.

After further discussion, it was determined that summarizing Microsoft's Azure DevOps documentation did not provide new insights or added value to the company. Instead, this question served as a guide for personal learning, contributing indirectly to the tool's development. The knowledge gained was incorporated into the tool and its accompanying documentation but was not pursued as a standalone deliverable.

2. What are the common mistakes and errors occurring in Azure DevOps pipelines, and how can they be identified? (Problem analysis, Root cause analysis)
   a. Gather internal documentation, incident reports, and feedback from developers at Info Support or external to identify common errors.
   b. Analyse patterns of mistakes.

   Outcome: A list of errors, ranked by the impact they would have if resolved. The impact would be informed by the frequency of the error among frequent developers. This list will inform the next stages of development.

This research went beyond providing a list of errors and assembled a comprehensive set of recommendations along with their impact.

3. Which static analysis techniques and tools are best suited for detecting mistakes in Azure DevOps pipelines? (Available product analysis; Choosing fitting technology)
   a. Evaluate existing static analysis tools, techniques, and methods (e.g., linters, syntax checkers)
   b. Assess the feasibility of incorporating or adapting these techniques into the tool being developed.

   Outcome: A clear decision on which analysis techniques will be used and what custom rules need to be implemented.

As opposed to writing a single research document, separate ADR's were written as new techniques or technologies were introduced in the project.

4. How can a static analysis tool be developed to integrate into development workflows while ensuring high performance and compliance with internal guidelines? (Prototyping, Non-functional test, Unit test, Product Review)
   a. Develop the prototype of the static analysis tool, incorporating selected techniques and custom validation rules.
   b. Ensure the tool integrates well into existing toolchains and adheres to company guidelines.
   c. Keep performance in mind, as to aim for below 1 second validation.

Outcome: A working prototype of the static analysis tool tailored to Info Support's needs.

1.5 Approach (local)

The project follows an agile methodology using Kanban with sprints. Kanban is well-suited for individual work due to its flexibility and simplicity. It allows for continuous task management and adaptation to changing priorities without the need for formal iterations or roles, which would mostly be taken by the student. Throughout the project, the approach shifted to emphasize **vertical slices** within sprints, delivering incremental functionality.

Initially, the project followed a phased plan that separated research, development, and documentation. However, this structure led to challenges in managing concurrent progress, creating a chicken-and-egg problem: in some cases, it was unclear what to develop without first knowing which techniques to use, while in other cases, it was difficult to determine which techniques to research without understanding the requirements of the implementation. This circular dependency caused delays. By Week 12, the approach was revised to integrate research and development into sprints, allowing for iterative progress and addressing techniques as needed for each feature.

By Week 15, the plan was further refined to focus on **template parameters** and **variables**, reducing scope to accommodate the remaining timeframe while ensuring the delivery of core functionalities. A contingency sprint (Sprint 5) was added to allow further coding efforts after the documentation deadline, targeting features that were deprioritized earlier.

The iterative evolution of the approach enabled the project to remain aligned with its goals despite setbacks, delivering a functional tool and comprehensive documentation while maintaining flexibility for future enhancements.

A more extensive description of the approach can be found in the portfolio site.

2. Conclusion

2.1 Research

Main Question:

How can static analysis be applied locally to Azure DevOps pipeline files to maximize mistake prevention before pipeline execution?

Static analysis can be applied locally to Azure DevOps pipeline files through a dedicated tool that parses pipeline YAML files and evaluates their syntax, structure, and embedded expressions. By leveraging real-time diagnostics via a Language Server Protocol (LSP) implementation, the tool provides developers immediate feedback on potential errors, enhancing mistake prevention without requiring pipeline execution.

2.1.1 How do Azure DevOps pipeline files function, what are their key components and what are the processes that result in their execution?

This question was removed as a standalone deliverable since its answer would primarily involve summarizing Microsoft's documentation, which would not add significant value to the company. However, it identified a knowledge gap that was addressed through company training and further learning during the project's development.

2.1.2 What are the common mistakes and errors occurring in Azure DevOps pipelines, and how can they be identified?

The most problematic features (where errors occur the most), were compile-time expressions, stage-level variables, predefined variables, and template parameters. These errors were identified through a survey and interviews with Info Support developers, using a weighted scoring model to prioritize issues by their impact and frequency.

This is further explained here.

2.2 Which static analysis techniques and tools are best suited for detecting mistakes in Azure DevOps pipelines

A parser was determined to be the most suitable tool for handling compile-time expressions, more specifically Tree-sitter was chosen as the most suitable parser. This due to its ability to recover from errors; it offers an easier implementation process using JavaScript for grammar; aligns well with the need for real-time feedback; and supports integration with multiple programming languages.

Further, custom diagnostic rules were implemented to address pipeline-specific errors. Decision records documented the rationale behind these choices. In the 'Parsing strategy', we determine how to tackle parsing, and in 'Parsing Solution', we determine the best suited technology.

2.2.1 How can a static analysis tool be developed to integrate into development workflows while ensuring high performance and compliance with internal guidelines?

The development of a static analysis tool for Azure Pipelines has been guided by several core principles aimed at ensuring seamless integration into existing development workflows, maintaining high performance, and adhering to internal guidelines.

From a technical perspective this is detailed in the Software Architecture, and a practical rundown is available in the Approach. Additionally the rationale for the decisions have been recorded using ADR's, which are available here.

2.2.1.1 Establishing guidelines

To ensure a project like this meets all its goals, it is crucial to establish a detailed set of requirements at the start, particularly non-functional requirements. These requirements define the standards for performance, usability, scalability, and maintainability, serving as a foundation for design and development decisions.

These can be found here.

2.2.1.2 Integration into Development Workflows

To integrate smoothly into the development environment, the tool implements the **Language Server Protocol (LSP)**. LSP allows the tool to communicate with multiple Integrated Development Environments (IDEs) such as Visual Studio Code and Neovim, without the need for custom solutions for each environment. This ensures **IDE independence** and reduces setup time for developers, meeting the **Seamless Integration** requirement of the project, as detailed in the non-functional requirements.

2.2.1.3 Ensuring High Performance

Performance is a critical aspect of the tool, particularly in the context of real-time feedback. To achieve this, the tool utilizes Tree-Sitter, a fast and efficient parser that ensures minimal delay in feedback even for large and complex files.

Additionally, the tool's modular architecture separates the **Language Server (I/O layer)** from the **core analysis logic**, allowing for optimized performance and scalability. This ensures that the tool can provide rapid diagnostics without compromising accuracy or performance.

This is further explained here.

2.3 Recommendations

This section proposes two potential paths for expanding or integrating the functionality developed during this project. These recommendations stem from both the lessons learned in the Development Phase and the feedback gathered through discussions with Info Support's developers.

2.3.1 Integrate with Microsoft's Existing Azure Pipelines Language Server

During this internship, a standalone Language Server was created using JavaScript (and Tree-sitter for parsing) to provide real-time diagnostics. However, Microsoft also maintains its own Azure Pipelines Language Server for Visual Studio Code.

The recommendation is to **Merge or Extend** with Microsoft's, integrating the functionality from this project into Microsoft's Language Server, leveraging a shared JavaScript codebase. The rationale for this approach is threefold. First, because both the existing Microsoft server and this project are written in JavaScript, the integration is potentially straightforward. Second, aligning with Microsoft's codebase may reduce the maintenance burden on Info Support, as new Azure Pipelines features or breaking changes could be addressed upstream. Finally, contributing enhancements to an established tool has the added benefit of community visibility, enabling a wider developer audience to benefit from these improvements while also providing Info Support with greater industry recognition.

2.3.2 Embed Functionality Directly into GitHub Runner

Another approach would be to integrate this static analysis logic directly into GitHub Runner, the open-source project that runs Azure Pipelines. Rather than maintaining a separate Language Server, the runner itself could handle diagnostics as part of the CI/CD pipeline execution. This would place the analysis closer to the actual runtime conditions, enabling more accurate, context-aware diagnostics; errors encountered during pipeline runs could be tied directly to code changes, simplifying debugging for developers. Additionally, users would not need to install a separate extension or language server, since all the functionality would be managed within the DevOps environment, providing a more unified and streamlined workflow.

3. Process

This section describes how the project plan was executed across the three phases: **Planning**, **Research**, and **Development**. It explains the rationale behind decisions and the activities carried out.

3.1 Planning Phase

3.1.1 Project Initiation & Early Uncertainties

At the start of the project, I introduced myself to the company, met the stakeholders, and tried to clarify the scope of the assignment. Initially, I believed the project's scope was modest; focusing solely on providing diagnostics for missing parameters in Azure Pipelines; so I was concerned about whether it would be substantial enough for a graduation internship.

To address these concerns, I spoke with my first assessor and my company mentor. After these discussions, uncertainty remained about whether the project's goals were large enough. My second company mentor then suggested consulting additional team members who had been part of the original project ideation. I reached out to them, arranged a meeting with one member and my first company mentor, and clarified that the intention was to create a form of "compiler" for Azure Pipeline files. Particularly aimed at compile-time expressions. This alignment significantly shifted my understanding of the project's potential impact.

3.1.2 Defining the Project Scope

After recognizing the value of locally evaluating compile-time expressions, I began drafting the Project Plan. Initially, I sketched out the main and sub research questions, focusing on local diagnostics for template parameters. Around this time, I also participated in a joint meeting with other Info Support interns to compare scopes and collect feedback on my proposed research approach. They urged me to broaden my scope, anticipating the many kinds of issues that could arise in Azure Pipelines.

I relayed this feedback to my company mentor, and together we decided to extend the project beyond just "compiling" Azure Pipelines or checking template parameters. We would aim to prevent and fix a broader range of common Azure Pipeline errors, creating a more comprehensive static analysis solution.

3.1.3 Finalizing the Project Plan

With the new focus established, I refined my main and sub research questions to:

1. Gain enough knowledge of how Azure Pipelines work (syntax, structure, compile-time evaluation).
2. Identify which mistakes are most common and worthwhile to address.

These were designed to ensure I had sufficient technical understanding while keeping the scope tangible. I then submitted the updated Project Plan for feedback to my first company mentor and both assessors, applying their comments and further detailing my approach.

To help my company mentor grasp Fontys's requirements, I compiled a Graduation Requirements Document outlining the critical deadlines and deliverables. With these activities completed, I concluded the **Planning Phase**.

3.2 Research Phase

3.2.1 Building Understanding of Azure Pipelines (Sub-question 1)

With the planning complete, I shifted my focus to Sub-Question 1: “How do Azure DevOps pipeline files function, what are their key components and what are the processes that result in their execution?”

Self-Study & Internal Diagrams

I reviewed Microsoft’s documentation and created personal diagrams illustrating the flow of Azure Pipelines and their compile-time expressions. Although these diagrams were initially just for my own clarity, they helped me conceptualize how local “compilation” or static analysis might be implemented.

Training

On the advice of my company mentor, I enrolled in a two-day **Azure DevOps** training program offered by Info Support’s internal Knowledge Centre. This hands-on course enabled me to set up my own Azure DevOps environment, build a simple pipeline, and experience common frustrations firsthand. This practical exposure was invaluable in deepening my domain knowledge.

3.2.2 Identifying Common Mistakes (Sub-question 2)

Simultaneously, I addressed Sub-Question 2: “What are the common mistakes and errors occurring in Azure DevOps pipelines, and how can they be identified”?

Choosing a Research Method

My company mentor and I initially discussed analysing commit histories across multiple client projects to find pipeline-related fixes. However, obtaining access to those repositories proved difficult because of Info Support’s consultancy model and client confidentiality. Moreover, deciding which commits **truly** reflected pipeline errors was subjective and would not necessarily yield conclusive data.

Designing & Testing the Survey

We then decided to create a **survey** aimed at Info Support developers. I structured the survey around specific pipeline “features” (such as compile-time expressions, template usage, etc.) to see which caused the most trouble in real-world scenarios.

Before distributing, I asked a coworker (a potential respondent) to review the draft. Their feedback highlighted a significant issue: many developers do not know Azure Pipeline features by name. Therefore, I added direct links to Microsoft’s official documentation for each feature, allowing participants to quickly confirm whether they recognized or had used that feature.

Survey Distribution & Challenges

Initially, I posted the survey in relevant Slack communities focusing on Microsoft, Azure, or DevOps topics. However, submissions were fewer than expected (<10). I then broadened the distribution, posting in additional channels and personally asking colleagues to complete the survey. Submissions rose to around 30, which provided a more meaningful dataset.

3.2.3 Survey Analysis & Reporting

After collecting enough responses, I exported the data to a Google Sheets document. By ranking features based on frequency and severity of problems, I created a prioritized overview of which parts of Azure Pipelines cause the most developer pain. Unsatisfied with just a numeric ranking, I wrote a **Survey Report** detailing:

1. The overall survey process and methodology.
2. A breakdown of the most problematic features.
3. Recommendations on which areas to tackle first to reduce pipeline-related frustrations.

Although I planned to conduct follow-up interviews for deeper insights, the Survey Report was reaching it's delivery deadline. I added an optional "contact me" field within the survey for participants willing to share more details. While these interviews were not included in the final Survey Report, I used them later to validate the survey findings.

With the Research Phase complete, I had:

- A clearer understanding of Azure Pipelines and its features.
- A ranked list of problematic features, backed by real developer feedback.

These outcomes laid the groundwork for what the next stages of the project would focus on.

3.3 Development Phase

After completing the Planning and Research phases, we moved on to executing the tool's implementation. This phase aimed to address the primary pain points identified in the Survey Report, starting with the most impactful: compile-time expressions. Below is an overview of the Development Phase, broken into chronological segments, including pivotal decisions, challenges faced, and how we ultimately refined the project scope into well-structured sprints.

3.3.1 Initial Attempt: Focusing on Compile-Time Expressions

Tackling the Parser Problem

Based on the Survey Report recommendations and internal discussions, we decided to address **compile-time expressions** first. These expressions let users programmatically define their pipelines and thereby introduce significant complexity. My initial plan was to parse these expressions locally, allowing the tool to provide real-time diagnostics and catch potential errors.

First I evaluated parsing approaches, as outlined in this ADR.

1. **Custom Parser**: Building a parser from scratch was too time-consuming and beyond my expertise.
2. **Off-the-Shelf Parser**: No standalone parser for Azure Pipeline expressions exists, and extracting the logic from GitHub's codebase (for Azure Pipelines) was complicated and ill-advised.
3. **Parser Generator**: Finally, I opted to use a parser generator and identified **Tree-sitter** for its solid error recovery, performance, and relatively easy setup.

Language Server:

After creating a working parser, I moved on to implementing a Language Server Protocol (LSP) solution:

- **Choice of Language**
  Per discussions with my company mentor, we selected **C#**, which is Info Support's primary development language and my mentor's area of expertise. This choice would facilitate better feedback and long-term maintainability.
- **Selecting LSP & YAML Libraries**
  I explored various LSP libraries in C#, settling on **OmniSharp** for its ongoing maintenance. However, I found few robust libraries to parse YAML with partial or incomplete input in real time, especially when compile-time expressions diverged from standard YAML structures.

Throughout this process I encountered several Roadblocks

  - **Sparse Tree-Sitter Documentation in C#**: Although Tree-sitter officially supports C#, its bindings lacked detailed documentation, making integration difficult.
  - **Incomplete LSP Features**: The OmniSharp library did not fully implement diagnostics, the most crucial feature for on-the-fly error reporting.
  - **YAML Inconsistency**: Azure Pipelines use YAML syntax but also compile-time expressions that do not conform strictly to YAML schemas.

Facing these three significant hurdles, I consulted my company mentor. We concluded that continuing down the same path would be highly inefficient. This decision is documented in a dedicated ADR, where we weighed the pros and cons of persisting in C#. In the end, we opted to transition to Typescript, for the better LSP support and well documented libraries.

3.3.2 Revisiting the Project Plan & Scope

Reassessing the Approach

By this stage, the project was falling behind the original schedule. I met with my second company mentor and my first assessor to discuss next steps. The consensus was that while the core idea still had potential, the approach needed streamlining, as building a compiler-like tool from scratch was too ambitious given my experience and the remaining time.

Creating a New Project Plan

We agreed on crafting an updated Project Plan with clearer goals, drawing from:

- The Survey Report's prioritized feature list.
- Realistic time constraints and my skill level.
- The recognition that full compile-time expression parsing might be more complex than initially estimated.

I introduced User Requirements Specifications (URS) to outline precise functionality and success criteria for each feature. This plan divided the project into defined sprints, each targeting a specific set of requirements.

Final Pivot: Dropping the "Compiler" Ambition

Despite the revised plan, implementing a compiler to fully handle compile-time expressions was still unfeasible within the internship period. We decided to deprioritize that component and shift focus to more attainable goals, such as template parameters and variables, which were still high on the list of developer pain points. This pivot was formalized in a final Project Plan, incorporating lessons learned from the earlier missteps.

3.3.3 Implementing a Structured Workflow with Sprints

With the final approach in place, we organized our work into Sprints. Note that Sprint 1; the attempt to handle compile-time expressions fully; was effectively discarded. We then started fresh from Sprint 2 with a refined scope and smaller achievable goals.

Sprint 2: Template Parameters

- **Objective**: Provide support for Azure Pipelines' template parameters, focusing on diagnosing missing or extra parameters in local templates.
- **Activities**:
  1. Integrated minimal LSP functionality for real-time diagnostics.
  2. Checked for unmatched parameters (i.e., parameters defined but not used and vice versa.)
  3. Conducted stand-up meetings every other day with my first company mentor to demo progress and collect immediate feedback.
- **Outcome**:
  1. Successfully delivered "must-have" URS features for template parameters.
  2. Deprioritized external template features (e.g., those in different repositories) due to authentication complexity.
  3. Documented changes in the **Architecture Document**, ensuring it reflected updated design decisions.

Sprint 3: Variables

- **Objective**: Diagnose issues related to variables within Azure Pipelines, such as undefined variables being referenced or unused variables.
- **Activities**:
    1. Continued the stand-up cycle for feedback loops.
    2. Focused on “must-have” user stories.
- **Outcome**:
    1. Variable diagnostics were successfully implemented.
    2. Incremental approach helped maintain momentum and meet updated timelines.

Sprint 4: Documentation & Final Reporting

- **Objective**: Finalize core documentation, update professional products, and write this Project Report.
- **Activities**:
    1. Updated architectural diagrams to reflect the final design.
    2. Ensured **Software Architecture** and user documentation matched the implemented features.
    3. Coordinated with mentors to confirm the contents of the final deliverables.
- **Outcome**:
    1. Delivered a comprehensive documentation set, including architecture, user guide, and reflection materials.

Sprint 5: Post-Portfolio Development

- **Objective**: (Planned) Continue working on leftover or lower-priority user stories after the portfolio’s submission.
- **Activities**:
    1. Expand testing coverage for existing features.
    2. Potentially revisit compile-time expressions, if time permits.
    3. Incorporate any final round of user feedback to polish the tool further.

4. Reading Guide

This section aims to provide a comprehensive overview of all the that was done and produced throughout the entire internship.

4.1 Professional Products

This section links to the professional products, and describes which learning outcome they contribute to and why.

4.1.1 Project plan (local)

Learning Outcomes Addressed:

- Professional Duties
- Future-Oriented Organisation
- Investigative Problem Solving
- Personal Leadership
- Targeted Interaction

Initiating and Defining the Project Scope:
At the start of the project, I recognized gaps in my understanding of the intent and requirements. This led me to proactively engage with stakeholders, including Romijn M. and Brandt J., and arrange a meeting with my company mentor and Romijn M. Through this initiative, I clarified the project scope and refined the expectations. The notes for this meeting can be found here. This demonstrates **Targeted Interaction** through stakeholder engagement and **Personal Leadership** in taking ownership of unclear requirements and clarifying them.

Structured Research Approach:
I aligned my research activities with the HBO-ICT research framework, selecting appropriate patterns and methods for each sub-question. This structured approach reflects **Future-Oriented Organisation** by ensuring a methodical and sustainable research process, and **Investigative Problem Solving** through evaluation of research methods best suited to the question.

Iterative Development and Feedback Application:
The project plan underwent four iterations. For each, I actively sought feedback from stakeholders and integrated their suggestions to improve the plan. While the final version was delivered within the available time constraints, I noted areas for future improvement, showing my ability to balance quality with deadlines. This iterative process showcases **Professional Duties** through delivering professional-grade documentation, **Targeted Interaction** by effectively communicating and integrating feedback, and **Personal Leadership** in evaluating and accepting constructive criticism.

4.1.2 User Requirements Specification (local)

Learning Outcomes Addressed:

- Professional Duties
- Situation orientation
- Future-Oriented Organisation

Building on Quality Requirements:
The user requirement specification formalizes the functional (FR) and non-functional requirements (NFR) identified during the project planning phase. By adopting user stories and acceptance criteria, I ensured the requirements were both specific and measurable. This demonstrates **Professional Duties** by creating a professional-grade deliverable, essential for guiding development.

Use of Known Techniques:
I utilized established techniques like user stories and acceptance criteria to structure requirements in a way that ensures clarity and aligns with best practices. This approach reflects my ability to apply known methodologies effectively in new contexts, showcasing **Situation-Oriented** skills.

Incorporating Stakeholder Feedback:
Through regular demonstrations and discussions with my company mentor, I refined and expanded the requirements and acceptance criteria based on their input. For example, adjustments were made to ensure alignment with users expectations; multiple fallbacks were determined for the positioning of diagnostics. This iterative collaboration exemplifies **Targeted Interaction**, where communication with stakeholders helped refine requirements and ensure they align with user's expectations.

Formalizing Requirements:
The specification was structured to include:

- **Non-Functional Requirements:** Constraints and quality attributes (e.g., coverage, performance).
- **Functional Requirements:** Clearly defined user stories that outline the desired behaviour.
  - **Acceptance Criteria:** Where applicable, further specification of the expected functionality.

Agile Workflow Integration:
Once defined, the user stories were included in the agile board, integrating them with the sprint-based development approach. This step highlights my ability to bridge planning with execution while ensuring visibility.

4.1.3 Software Architecture (local)

Learning Outcomes Addressed:

- Professional Duties
- Situation orientation
- Future-Oriented Organisation
- Investigative Problem Solving
- Targeted Interaction

Architectural Decisions and Documentation:
Throughout the project, I documented big architectural decisions using the MADR (Markdown Architectural Decision Records) template, as detailed in the ADRs. This approach ensured and a clear and defensible rational for each decision, exemplifying **Professional Duties** and **Investigative Problem Solving** by applying industry-recognized practices and conducting thorough evaluations of alternatives.

Use of Standardized Models:
The architecture was visualized using the C4 model, supported by sequence diagrams and additional diagrams where needed. This adherence to industry standards ensured that the design was clear, communicable, and easily understandable for stakeholders, reflecting **Professional Duties** and **Situation Orientation** by adapting to the audiences' expectations.

Guiding Principles:
The architecture design was guided by the project's non-functional requirements (e.g., IDE independence, seamless integration, real-time validation). These constraints were come from the User Requirements Specification and aligned with stakeholder needs, showcasing **Future-Oriented Organisation** by considering scalability and maintainability.

Iterative Updates:
The architecture evolved alongside the project, with incremental updates to reflect changes in scope and features. This continuous alignment with the project illustrates **Future-Oriented Organisation** ensuring that the architecture remained relevant and effective.

Stakeholder Collaboration:
I regularly presented architectural decisions and diagrams to stakeholders for feedback, adapting designs based on their input. The initial idea do add future extensions within the diagrams, came from them. This demonstrates **Situation Orientation** and **Targeted Interaction**, ensuring the architecture aligned with stakeholder expectations and projects' goals.

4.1.4 Survey Report (local)

Learning Outcomes Addressed:

- Professional Duties
- Targeted Interaction
- Investigative Problem Solving
- Personal Leadership

Objective and Approach:
The survey aimed to identify and prioritize problematic features in Azure Pipelines to guide the development of a diagnostic tool. To make the questions clearer, I added links to official documentation for each feature, ensuring accessibility and usability for participants. This demonstrates **Professional Duties** and **Targeted Interaction** by creating a thoughtful and professional survey.

When responses were slow, I adapted my approach by searching for additional relevant Slack channels and asking colleagues directly to fill out the survey. This flexibility and effort to engage participants reflect **Targeted Interaction** and **Personal Leadership**, as I focused on encouraging collaboration to achieve the project's goals.

Feedback and Analysis:
I sought feedback on the survey design and noted suggestions for future improvement, even if I didn't apply them immediately, showing **Professional Duties** by maintaining a continuous improvement mindset. After collecting responses, I used a weighted scoring method to rank features based on severity, frequency, and proficiency. Going beyond the rankings, I analysed the data to develop actionable recommendations, demonstrating **Investigative Problem Solving** by identifying root causes and proposing actionable solutions.

Collaboration:
Throughout the process, I incorporated feedback from stakeholders to ensure the report's relevance and clarity, showcasing **Targeted Interaction**.

4.2 Additional Activities

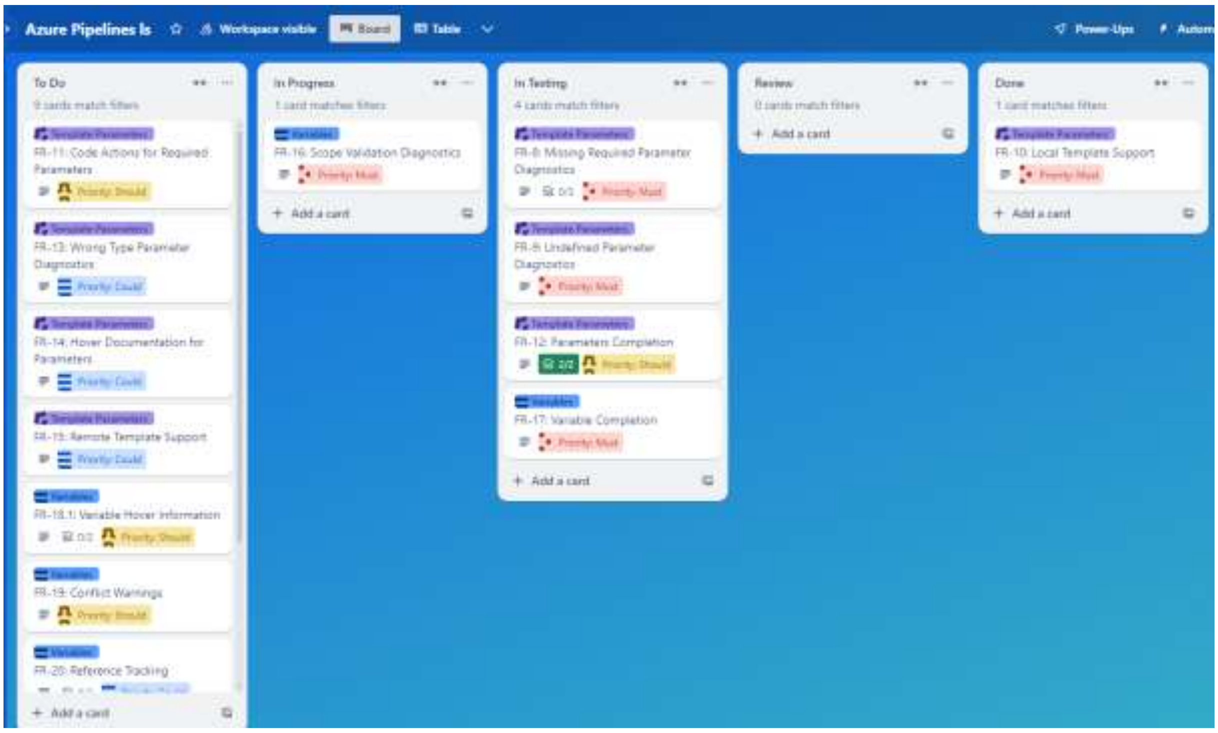
4.2.1 Kanban board

Learning Outcomes Addressed:

- Future-Oriented Organisation

To manage tasks throughout the project, I use a Trello Kanban board structured with columns: "To Do," "In Progress," "In Testing," "Review," and "Done," as outlined in the project plan. This setup allows me to track progress, prioritize tasks, and ensure that work flows efficiently through each phase.

Using this board helps me stay organized and adapt to changing priorities, demonstrating **Future-Oriented Organisation.**



Azure Pipelines ls — Workspace visible — Board — Table — Power-Ups — Autom

To Do — 9 cards match filters
- Template Parameters — FR-11: Code Actions for Required Parameters — Priority: Should
- Template Parameters — FR-13: Wrong Type Parameter Diagnostics — Priority: Could
- Template Parameters — FR-14: Hover Documentation for Parameters — Priority: Could
- Template Parameters — FR-15: Remote Template Support — Priority: Could
- Variables — FR-18.1: Variable Hover Information — 0/2 — Priority: Should
- Variables — FR-19: Conflict Warnings — Priority: Should
- Variables — FR-20: Reference Tracking
- Add a card

In Progress — 1 card matches filters
- Variables — FR-16: Scope Validation Diagnostics — Priority: Must
- Add a card

In Testing — 4 cards match filters
- Template Parameters — FR-8: Missing Required Parameter Diagnostics — 0/3 — Priority: Must
- Template Parameters — FR-9: Undefined Parameter Diagnostics — Priority: Must
- Template Parameters — FR-12: Parameters Completion — 2/2 — Priority: Should
- Variables — FR-17: Variable Completion — Priority: Must
- Add a card

Review — 0 cards match filters
- Add a card

Done — 1 card matches filters
- Template Parameters — FR-10: Local Template Support — Priority: Must
- Add a card

4.2.2 Meeting notes

Learning Outcomes Addressed:

- Targeted Interaction
- Personal Leadership

To clarify the project scope and align expectations, I documented the key points and outcomes of a meeting with my company mentor and Romijn M. This document served as a clear reference for decisions made during the discussion. By proactively organizing this meeting and ensuring its outcomes were captured, I demonstrated **Targeted Interaction** through effective communication and **Personal Leadership** by taking initiative to resolve uncertainties.

4.2.3 Progress Update Meetings

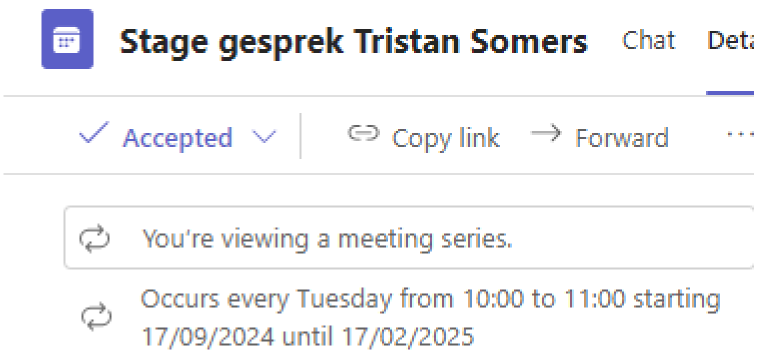
Learning Outcomes Addressed:

- Targeted Interaction
- Professional Duties

Throughout the project, I participated in regular progress update meetings, as outlined in the project plan. These included:

- **Weekly meetings with my first company mentor:** These provided consistent guidance and ensured alignment with the project's goals.
- **Bi-weekly meetings with my second company mentor:** These decreased in frequency as the project progressed and became less necessary, reflecting the project's steady development.
- **Weekly meetings with my first assessor:** These discussions focused on project progression, challenges, and ensuring alignment with Fontys expectations. These meetings also included another student, allowing us to compare progress, share advice, and support each other's development.

These meetings helped maintain clear communication with stakeholders, incorporate feedback into my work, and ensure the project would not go off track. The collaborative aspect of the assessor meetings shows **Targeted Interaction**, while consistent reporting and accountability demonstrate **Professional Duties**.



You're viewing a meeting series.

Occurs every Tuesday from 10:00 to 11:00 starting 17/09/2024 until 17/02/2025

4.2.4 Graduation requirements

Learning Outcomes Addressed:

- Targeted Interaction
- Personal Leadership

To ensure clarity and alignment regarding my graduation process, I created a detailed Graduation Requirements Document. This document outlines the key deliverables, deadlines, and important meetings required for the successful completion of my graduation project. It includes the six learning outcomes, deliverable expectations, and communication guidelines with assessors and the company.

By proactively organizing and sharing this document, I facilitated effective communication with all stakeholders, ensuring everyone was aligned on the process and requirements. This demonstrates **Targeted Interaction** and **Personal Leadership**.

4.2.5 Company Training

Learning Outcomes Addressed:

- Professional Duties
- Investigative Problem Solving
- Personal Leadership

At the start of my internship, I had no prior experience with Azure DevOps. To address this my company mentor suggested I follow training, thus I proactively searched for and enrolled in a two-day training course offered by Info Support: **Azure DevOps Engineer (AZ-400)**. This course provided a solid foundation in Azure DevOps concepts and practices, allowing me to better understand the context around the project..

Taking the initiative to identify and fill this knowledge gap demonstrates **Personal Leadership**, as I took responsibility for my professional growth. The training also reflects **Investigative Problem Solving**, as it directly addressed a challenge that could have impacted my ability to succeed in the project. Furthermore, it highlights **Professional Duties**, as I ensured I had the necessary skills to meet the expectations of my role.

4.3 Learning Outcomes

This section provides a mapping from learning outcome to the evidence where it is displayed.

Learning Outcome	**Evidence**
Professional duties	Project plan (local) User Requirements Specification (local) Software Architecture (local) Survey Report (local) Progress Update Meetings Company Training
Situation-Orientation	User Requirements Specification (local) Software Architecture (local)
Future-Oriented Organisation	Project plan (local) User Requirements Specification (local) Software Architecture (local) Kanban Board
Investigative Problem Solving	Project plan (local) Software Architecture (local) Survey Report (local) Company Training
Personal Leadership	Project plan (local) Survey Report (local) Company Training Graduation Requirements Meeting notes
Targeted Interaction	Survey Report (local) Software Architecture (local) Project plan (local) Progress Update Meetings Graduation Requirements Meeting notes

5. Evaluation and Reflection

Reflecting on this project, I see a clear split between the strong, structured beginning and the later stages where my workflow lost cohesion. Initially, I took a thorough approach to defining project requirements and actively sought opportunities to deepen my knowledge, such as enrolling in Azure DevOps training. This groundwork helped me gain clarity on the project's scope and build productive relationships with stakeholders.

However, as the project progressed and I encountered unexpected complexities; particularly around building a compiler-like tool for compile-time expressions; I began to deviate from my initial structured method of working. With the project running behind schedule, I neglected some of the formal processes and practices I had outlined at the start. This lapse resulted in multiple setbacks and the need for two new project plans. In hindsight, focusing on the most challenging feature first (compile-time expressions), while simultaneously learning how to implement a language server and parse YAML for the first time, was overly ambitious. A more incremental approach, starting with simpler features to build momentum, would have been a better strategy.

Despite these challenges, I believe I demonstrated resilience and willingness to seek help at crucial moments. When major roadblocks emerged, I turned to stakeholders; my mentors, assessors, and team members; for guidance, and their feedback proved instrumental in recalibrating the project. Eventually, I reintroduced structured methods such as daily stand-up meetings with my mentor, which significantly improved communication and progress tracking. Unfortunately, this structured approach came late in the process, leading to several rushed deliverables, including aspects of this project report.

Moreover, I received feedback that my communication needed to improve. Initially, I did not act on that feedback as proactively as I should have. However, by introducing stand-up meetings and more frequent status updates, I enhanced both the clarity and consistency of my communication. These late improvements highlighted the importance of ongoing self-reflection and the need to maintain professional practices throughout the project, not just at the start.

Overall, this experience taught me the critical value of sustained structure and clear communication. While I managed to deliver core functionality and eventually regain control of the project's direction, earlier adoption of a more incremental development approach would have prevented many of the complications I faced. Going forward, I will be more diligent in maintaining structured planning processes, seeking stakeholder input regularly, and prioritizing simpler deliverables first to steadily build momentum on complex projects.