SmartDelta

Automated Quality Assurance and Optimization in Incremental Industrial Software Systems Development

D2.4 – SmartDelta Methodology: Users and Developers Guidelines

Submission date of deliverable: Dec 31, 2024

Edited by: Benedikt Dornauer (University of Innsbruck, Austria), Andrea Pabón-Guerrero (Universidad Carlos III de Madrid, Spain), Mehrdad Saadatmand (RISE, Sweden), Hakan Kilinc (NetRD, Turkey), Nicolas Bonnotte (Akkodis, Germany), Andreas Dreschinski (Akkodis, Germany), Martin Heß (Software AG, Germany), Robin Gröpler (ifak, Germany), Muhammad Abbas (RISE, Sweden), Raluca Marinescu (Alstom, Sweden), Zulqarnain Haider (Alstom, Sweden), Akramul Azim (Ontario Tech University, Canada), Eduard Paul Enoiu (Mälardalen University, Sweden), and WP2 partners

Project start date	Dec 1, 2021
Project duration	36 months
Project coordinator	Dr. Mehrdad Saadatmand, RISE Research Institutes of Sweden
Project number & call	20023 - ITEA 3 Call 7
Project website	https://itea4.org/project/smartdelta.html & https://smartdelta.org/
Contributing partners	SmartDelta partners
Version number	1.0
Work package	WP2
Work package leader	Juan Miguel Gomez Berbis (UC3M)
Dissemination level	Public
Description	D2.4 describes the SmartDelta Methodology, addressing key aspects of delta management, providing solutions for different stages of the software engineering process. Its goal is to guide companies in managing software deltas, enabling tailored incremental development aligned with their strategic objectives.

Executive Summary

The SmartDelta consortium engages in collaborative research with a diverse range of industrial partners, including those in the railway, e-mobility, telecommunications, finance and banking, enterprise software, logistics, personal mobility, and cybersecurity sectors. Work Package 2 (WP2) offers a methodology for internal and external stakeholders, providing essential guidance for interacting with SmartDelta's solutions and workflow. One of the key objectives of these materials is to empower SmartDelta stakeholders to easily implement specified and developed procedures and solutions and understand features and capabilities of offered functions and tools and adopt them in their own contexts.

The SmartDelta Methodology offers a systematic guideline for managing software quality in incremental software development contexts, helping to adapt seamlessly to evolving requirements. By addressing key challenges in delta management, it delivers tailored solutions aligned with the various phases of the software engineering lifecycle. With a focus on empowering companies to efficiently handle software deltas, the SmartDelta Methodology facilitates the iterative development of systems while aligning with specific business goals and strategies.

Building upon industrial evaluations and incorporating valuable lessons learned, this document presents the final version of the SmartDelta Methodology developed by the project consortium. It reflects the thoughtful evolution of our approach.

Table of Contents

1. Introduction of SmartDelta Methodology

SmartDelta builds automated solutions for quality assessment of product deltas in a continuous engineering environment. It provides intelligent analytics from development artifacts (e.g., source code, log files, requirement specifications, etc.) and system execution, offering insights into quality improvements or degradation of different product versions and providing recommendations for the next build. To understand the relationships between individual processes and solutions, we created a software delta management concept labeled as the *SmartDelta Methodology*.

The SmartDelta Methodology provides a structured approach for managing software quality in incremental software development that emphasizes adaptability to change. It enables companies to address essential aspects of delta management within the software engineering process, offering targeted solutions for various stages. In this project,

A delta is any change in a software product that results in a new product instance with different functionality and/or quality properties.

Examples of such changes might involve adding or removing features and components, applying necessary fixes and updates, reconfiguring the product for deployment in a different environment, or customizing it to satisfy the requirements of a new customer. This definition of Delta discussed here is often linked to the temporal aspects of product evolution and is commonly referred to as ***Version Delta***. A version represents a specific release of a software product, typically including updates, that have bug fixes, enhancements, or new features, all of which can be viewed as forms of deltas. A ***Variant Delta*** refers to distinct instances of software that have been modified or customized for specific purposes or applications. These variants may differ in functionality, features, or configurations while maintaining a common core or foundation derived from the original software. They may arise from multiple version updates.

It is observed that software quality can change or even decline over time with each version or variant, and maintaining quality with existing processes and methods demands a considerable investment of ***resources***. These updates may enhance certain quality aspects, while some other aspects can deteriorate. The main objective of the SmartDelta Methodology is to provide a comprehensive guideline to support companies in managing quality across software deltas, enabling incremental development of their software systems according to their unique requirements and strategic objectives. To achieve this, the SmartDelta Methodology enables improvements across several dimensions by targeting and meeting the following objectives:

Table 1: Goals of the SmartDelta Methodology

Objectives	Description
Improve version management	The SmartDelta Methodology allows for an examination of the product's evolution over time, elucidating the way its constituent components have changed and the probable evolution of these components in the future.
Improve variant management	The objective is to facilitate the management and differentiation of multiple variants of a product, with a particular emphasis on understanding and reusing software components that share similarities across variants.
Improve Resources	The solutions provided by SmartDelta result in a reduction of resource consumption through the automation of processes, the reuse of artifacts, and the application of artificial intelligence, while simultaneously enhancing the overall quality of the software. Here, "resources" encompass time savings, reduced computational power

Objectives	Description
	requirements, minimized human effort, and enhanced scalability, among other factors.
Improve overall quality	The SmartDelta Methodology is designed to facilitate quality improvement in CI/CD practices through the provision of comprehensive monitoring and visualization capabilities.

2. SmartDelta Methodology Stages

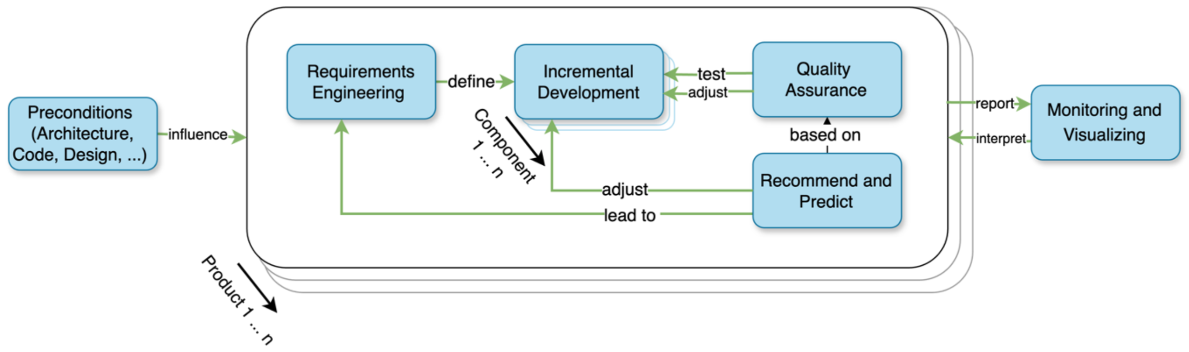

Figure 1: Methodology Stages Overview

By focusing on both ***Version Deltas*** (incremental updates within a product's lifecycle) and ***Variant Delta*** (customizations for specific applications), the SmartDelta Methodology is designed to address the evolving needs of complex software systems. It guides the development lifecycle through six key stages. Each stage incorporates delta-aware practices, ensuring that every modification meets the product's quality expectations. The *SmartDelta Methodology* contains several process stages that are part of a software development lifecycle:

Table 2: Methodology Stages Description Overview

Stage	Description
Preconditions (Architecture, Code, Design and other Artifacts)	**Influence factors** from stakeholders, entities, and other external sources (including architecture, management processes, code artifacts, and design principles) shape the product's direction, ensuring alignment with both external and internal requirements. Each new delta introduced at this "influential" stage brings adjustments that can significantly impact the development process.
Requirements Engineering	In this stage, **requirements engineering is a delta-focused, iterative process** that emphasizes capturing and managing changes in needs and specifications over time. Requirements are continuously identified and adapted from diverse sources, with a strong focus on tracking and analyzing each delta (i.e., any modification or adjustment needed to meet evolving goals). Quality standards are ensured through **delta-aware quality analysis**, where existing requirements are analyzed for reuse, adapting them

Stage	Description
	efficiently to new contexts and minimizing redundant efforts. **Model extraction** plays a crucial role in visualizing requirements and their deltas to understand the impact of each change on the system as a whole. **Verification and validation** processes confirm that these evolving requirements align with stakeholder needs and adhere to system constraints. By using delta-aware strategies, this stage ensures that functional and non-functional requirements across IT architecture, UX, and UI layers remain responsive and adaptable, adjusting changes throughout the product's lifecycle. [1].
Incremental Development	The incremental development process is focusing on **managing and leveraging deltas**. In the incremental development stage, SmartDelta enables analysis and management of each incremental update, providing a view of how different software versions and variants branch over time. This differentiation supports **code reuse** and enables the application of delta-aware tools, such as **automated model generation**, for **quality analysis** across the product line. Deltas can be assessed for potential reuse, enabling integration into newer versions or variants. Nevertheless, incremental updates often carry quality degradation. For example, adding new features to a software application can lead to increased cyclomatic complexity metrics, which measure the number of linearly independent paths through a program's source code. As more features are added, the control flow becomes more intricate, making the code harder to understand, maintain, and test, ultimately impacting the overall quality of the software. Thus, **quality checks** can be used to ensure software aligns with quality standards. Logs and issues from previous versions are resources for understanding potential risks in incremental development, allowing for preemptive adjustments. With some deltas representing a change in functionality or quality, **automated model generation** can provide an approach to evaluate the impact of these changes on the overall system. Through incremental development, SmartDelta is focusing on the reuse potential inherent in each delta, whether an enhancement, fix, or customization, aligns with the quality goals. Furthermore, by employing analyses across areas such as code reuse, quality, logs, and automated model generation, this phase can show feedback on frequent updates for continuous improvement [2].
Quality Assurance	This stage emphasizes **delta-focused quality assurance**, where each change or delta, whether a minor code adjustment or a major feature update, is tested and validated. **Delta-aware test generation** is a major technical area, explicitly targeting components and code changes to ensure that all modifications meet requirements and quality criteria. **Test amplification** further improves this process by expanding test cases to cover cases introduced by deltas. **Continuous monitoring and anomaly detection** provide real-time feedback into quality degradation, identifying any issues caused by new deltas.

Stage	Description
	Static analysis of deltas is also performed to assess changes against predefined static quality measures. By integrating these delta-aware techniques, this stage is used to find defects [3].
Recommend and Predict	This stage uses **delta-aware analysis to provide predictive recommendations** for the software's evolution. Using data from the quality assurance stage, it performs **quality analysis prediction** to predict the potential impact of new deltas on system quality, identifying areas that may require adjustments. This stage identifies recurring patterns or components that can be reused through **similarity analysis** and **reuse recommendation**, improving efficiency and consistency across the system. **Change impact analysis** assesses how each delta affects interconnected components, providing an understanding of potential effects. Based on these analyses, actionable recommendations are made to address challenges or improvements. This delta-focused approach ensures that each prediction and recommendation is backed by data-driven insights, allowing the system to adapt as it evolves. [5].
Monitoring and Visualizing	This **delta-aware stage of monitoring and visualization** provides a comprehensive view of metrics, states, dependencies, and interactions across all system components. At the **product line level**, monitoring and visualization track, log, and analyze resources, capturing states, events, and metrics. This level observes individual metrics and focuses on **variant delta interactions**, which are how changes or updates affect dependencies and relationships between system components. This overview allows teams to understand the broader impact of each delta across interconnected systems, providing insight into the evolving state of the entire infrastructure. At the version delta level, monitoring focuses on tracking different **metrics** throughout the product life cycle. **Visualization tools** can alert teams when values exceed acceptable ranges, enabling a detailed view of each delta's impact on specific components. This **in-depth analysis** helps detect issues and provides insights into how localized deltas may influence overall quality stability [4].

3. SmartDelta Stages and Technical Areas: Inputs, Outputs, Deltas, and Their Interrelationships

After introducing the stages, we aim to provide a detailed view of each stage. The following subsections introduce the relevant technical areas, illustrate the correlations between the stages, and define the inputs (i.e., what is required at each stage, e.g., documents) and the outputs (i.e., what is generated through the process, e.g., quality report).

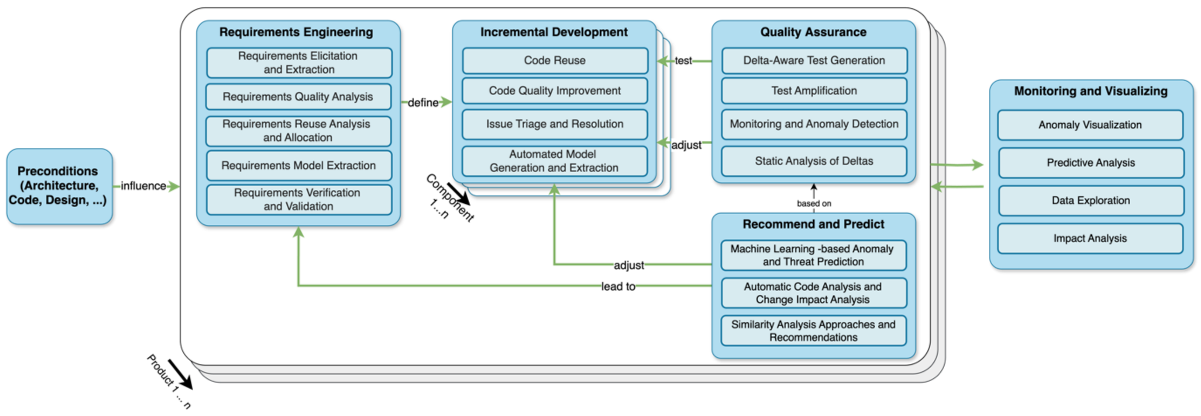
Preconditions (Architecture, Code, Design, ...)
influence
Requirements Engineering
Requirements Elicitation and Extraction
Requirements Quality Analysis
Requirements Reuse Analysis and Allocation
Requirements Model Extraction
Requirements Verification and Validation
define
Incremental Development
Code Reuse
Code Quality Improvement
Issue Triage and Resolution
Automated Model Generation and Extraction
Component 1...n
test
adjust
Quality Assurance
Delta-Aware Test Generation
Test Amplification
Monitoring and Anomaly Detection
Static Analysis of Deltas
based on
Recommend and Predict
Machine Learning -based Anomaly and Threat Prediction
Automatic Code Analysis and Change Impact Analysis
Similarity Analysis Approaches and Recommendations
adjust
lead to
Monitoring and Visualizing
Anomaly Visualization
Predictive Analysis
Data Exploration
Impact Analysis
Product 1 ... n

Figure 2: Overview of stages and their technical areas.

3.1. Preconditions Consideration

Business requirements define the scope of the solution and the company's objectives, whereas functional requirements address how the company will achieve its desired outcome. Furthermore, the characteristics of the solution, such as its architecture and languages, are considered, as these factors will influence the development of variants and versions. In the following discussion, we will focus on business requirements, as they play a key role in shaping and formulating software requirements.

An understanding of the functional requirements (FRs) is beneficial in determining the rationale behind the existence of the application in question. In other words, what is the fundamental business problem that the application is designed to address? It is also important to consider the original purpose for which the application was designed and the context in which it was created, including the technology stack, architecture, and security requirements. A focus on the way an application addresses its business problem will inevitably lead to an analysis of its functional requirements. A functional requirement represents the action an application must perform.

Non-functional requirements (NFRs), on the other hand, are not related to a specific function or behavior for the application to function. Still, they do define system attributes such as security, reliability, performance, maintainability, scalability, and usability. NFRs serve as constraints or restrictions on the system's design across different backlogs.

This precondition defines the baseline value (as-is situation) and the requirements to achieve the business objective in each environment.

Outputs	• Business Requirements, baseline values • Limitations for the other stages • Corpus of existing artifacts
Relationships	• Link business objectives to software requirements

Stakeholders and Roles	• Project Managers. • Product Owners. • Quality Assurance Engineers.

3.2. Requirements Engineering (RE)

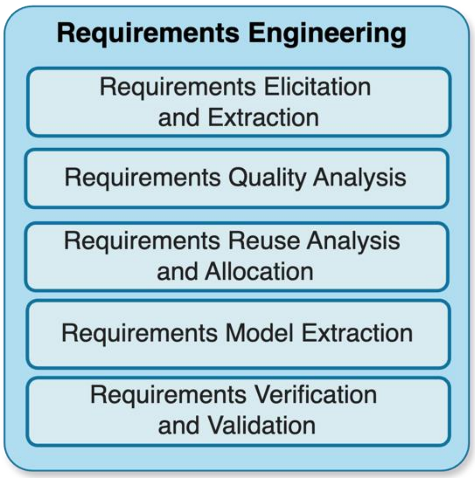
Requirements Engineering
Requirements Elicitation and Extraction
Requirements Quality Analysis
Requirements Reuse Analysis and Allocation
Requirements Model Extraction
Requirements Verification and Validation

Figure 3: Technical Areas Requirements Engineering

The Requirements Engineering stage, from the perspective of the SmartDelta project, involves the iterative process of defining, documenting, and managing both functional and non-functional requirements. This stage captures the evolving needs of stakeholders across product iterations, ensuring that the system under Development aligns with goals, regulatory requirements, and user expectations. **[Requirements elicitation and extraction]** To achieve this, the Requirements Engineering phase of the methodology focuses first on extracting and eliciting customer requirements from customer wishes (often tender documents and change requests, with tools such as ReqI), [**Requirements Quality Analysis**] which are then statically analyzed for potential quality issues and ambiguities (with tools like NALABS and RADICLE). [**Requirements Reuse Analysis and Allocation**] The requirements are improved, and the improved new requirements are then searched across the variants and versions of the existing projects and products to identify potential reuse opportunities (with tools like issue similarity and VARA+) for existing artifacts associated with the existing requirements. The aim is to reduce development and analysis waste by reusing verified and validated artifacts to realize the new requirements. During this process of reuse analysis, the identified potential artifacts for reuse may require additional modifications to be usable for the new requirements. Also, many of the requirements may necessitate the additional development of new artifacts. Therefore, the step also allows for the intelligent allocation of the new requirements as work items to various teams for implementation (with tools like REQA). [**Requirements Model Extraction**] To enable seamless incremental implementation of the final requirements, the methodology provides ways to extract behavioral models from the changing requirements continuously (with tools like REFORM). The extraction of the model from requirements allows users of the methodology to formalize the new requirements in standard formal languages such as UML state machines. For example, the Reform Tool from IFAK, applied in the Akkodis use case, facilitates this process. **[Requirements Verification and Validation]** Finally, the extracted models and the refined requirements are then used to verify and validate incremental development artifacts against the requirements (with tools like TIGER+, REFORM, and PyLC).

Note that all the sub-phases within requirements engineering can be instantiated independently. This stage feeds directly into incremental development and quality assurance efforts, with close feedback loops between requirements, development, and testing.

Inputs	• Development artefacts from other variants and versions ○ Architecture, code, design, requirements, issues etc. • Stakeholder needs, expectations and wishes ○ Change requests, tender documents, proposals, and issues etc.

Outputs	• Requirements documentation (quality checked) • Specifications for incremental development
Relationships	• Requirements Engineering leads to Incremental Development. • It adjusts based on feedback from Quality Assurance and Recommend and Predict stages.
Stakeholders and Roles	• Requirement Engineers, Development Team, Quality Assurance Engineers, Project Managers, Stakeholders

3.3. Incremental Development

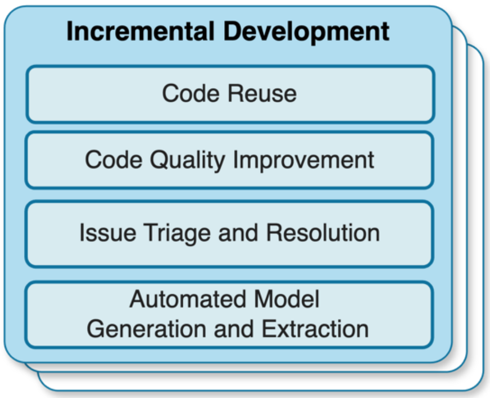
Incremental Development
Code Reuse
Code Quality Improvement
Issue Triage and Resolution
Automated Model Generation and Extraction

Figure 4: Technical Areas Incremental Development

In the Incremental Development stage, software systems are built using reusable and modular components. This approach allows the development team to integrate existing components. Components are iteratively developed, incorporating feedback from previous delta analysis and quality assurance efforts. Their modularity enables them to be adapted to functional and non-functional requirements. In this regard, the Incremental Development stage of the SmartDelta Methodology addresses activities that result in the creation of a new product instance and emergence of new product deltas.

Technical Areas and Focused Techniques. To enable incremental development, SmartDelta has provided innovations focusing on the following technical areas:

- **Code reuse:** Approaches for identifying reusable code and to avoid redundant development. This can be done based on the requirements analysis results, identifying similar requirements and software artifacts (e.g., components and test cases), and provision of reuse commendations.
- **Code Quality Improvement:** Improving the quality of the code for a new product instance to achieve certain characteristics such as reducing complexity, improving maintainability, optimizing the code, and enhancing performance of the product. To achieve this, the results of quality assurance techniques, and in particular SmartDelta solutions for delta-aware static analysis and testing, can be used iteratively and in a feedback loop to achieve quality improvements in the code.
- **Issue Triage and Resolution:** Using insights from customer reports and issue analysis results to patch and resolve the issues in the product or to update it with new features and functionality (e.g., based on customer requests and reports).
- **Automated Model Generation and Extraction:** Facilitating rapid prototyping and validation of product instances.

Overall, these techniques enable a data-driven approach, enhancing productivity and ensuring higher-quality outcomes in incremental development of industrial software systems.

Inputs	• Requirements from Requirements Engineering • Existing components for integration

Outputs	• Developed or integrated components • Metadata for Quality Assurance Assessment
Relationships	• Leads to Quality Assurance. • Receives feedback to adjust components from Quality Assurance and Recommend and Predict stages.
Stakeholders and Roles	Software Developers, System Architects, Project Managers, Product Owner, Product Line Manager

From a product line management perspective, the story of an Increment can be as follows. The first goal in product development is the Minimally Viable Product (MVP) level. The Increment does not have to be an MVP, but it should be usable, interactable with stakeholders, and improvable with feedback from the people who use it. After one Increment, MVP can also be achieved, or multiple Increments may be required. The Increment lives in a non-production environment until it reaches MVP. Once it reaches MVP, the Increment is distributed to production for actual use. Here, the MVP is just a milestone. It would be possible to model not only a minimal product but also the time and effort required for a build-measure-learn feedback loop. This cycle is iteratively repeated and improved until it incrementally reaches the desired point. Undesirable deviations and deltas are eliminated through learning and self-correction. In product line development, this process also proceeds in time and space, i.e., in the relationship between version and variant. The increment must be usable and auditable in any case. The given flow in the SmartDelta Methodology meets the Increment, and its reusability in developing new variants is valuable to consider in the build-measure-learn cycle.

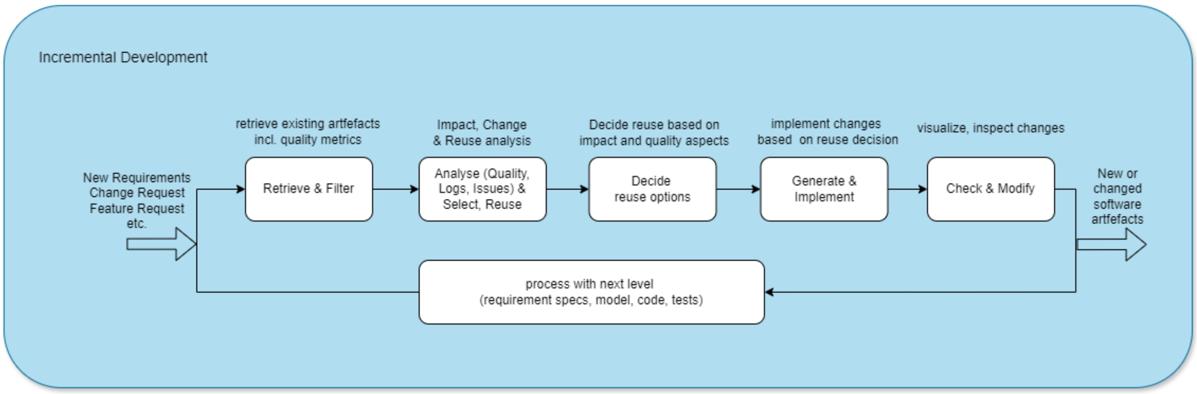
Incremental Development
retrieve existing artfefacts incl. quality metrics
Impact, Change & Reuse analysis
Decide reuse based on impact and quality aspects
implement changes based on reuse decision
visualize, inspect changes
New Requirements Change Request Feature Request etc.
Retrieve & Filter
Analyse (Quality, Logs, Issues) & Select, Reuse
Decide reuse options
Generate & Implement
Check & Modify
New or changed software artefacts
process with next level (requirement specs, model, code, tests)

Figure 5: General Incremental Software Development Approach

The diagram illustrates a systematic approach to incremental software development, highlighting the general steps from retrieving relevant artifacts to implementing and inspecting changes. This approach ensures a streamlined and efficient development process, leveraging automation and analytics to optimize decision-making and implementation.

3.4. Quality Assurance

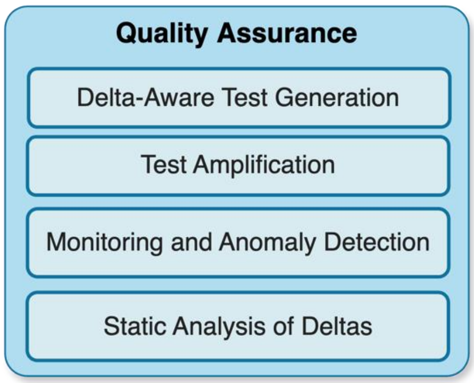
Quality Assurance
Delta-Aware Test Generation
Test Amplification
Monitoring and Anomaly Detection
Static Analysis of Deltas

Figure 6: Technical Areas Quality Assurance

The Quality Assurance stage uses inputs, including artifacts from incremental development cycles and data specific to incremental development, to shape a delta-oriented quality assurance process. These artifacts inform the delta-aware testing methodology by highlighting which code and test code areas are affected, enabling Quality Assurance Engineers and Test Engineers to focus their efforts on specific deltas.

Inputs and Processes. Incremental development input is used to tailor delta-aware testing to address specific new functionalities or adjustments. Delta-aware test generation and static analysis, such as that performed by SEAFOX, use this information to generate, select, and amplify test cases that thoroughly assess functional and extra-functional requirements specific to the evolved components. Additionally, static analysis performed by DRACONIS evaluates each delta against established quality criteria, allowing different stakeholders to detect potential issues early and provide actionable feedback for iterative adjustments.

Outputs and Relationships. The primary outputs of this stage include Quality Assurance Reports and Feedback directly applicable to ongoing incremental development. This feedback loop is important, as it allows Software Developers to address quality concerns early, thus minimizing the impact of defects on subsequent stages. Moreover, data generated during quality assurance activities is directed to visualization systems, supporting continuous monitoring and anomaly detection. This connection ensures that any quality degradation or unexpected behavior introduced by recent changes is addressed and visible through visualization tools.

Technical Areas and Focused Techniques. To ensure comprehensive quality control, the Quality Assurance stage employs various technical approaches:

- Delta-Aware Test Generation: This approach generates tests specific to deltas, ensuring that changes are considered when regenerating test cases.
- Test Amplification: Builds upon initial test cases by expanding them to cover additional scenarios and potential issues introduced by each delta.
- Monitoring and Anomaly Detection: Integrates feedback mechanisms that detect and alert QA teams to any anomalies resulting from incremental changes.
- Static Analysis of Deltas: Evaluates code changes against established quality criteria, identifying risks in functional and non-functional properties before full deployment.

Stakeholders and Roles. The Quality Assurance stage involves Quality Assurance Engineers and Test Engineers who directly manage and execute the delta-aware quality assurance process. Software developers are closely involved in implementing feedback for continuous improvement, while project managers oversee the alignment of quality goals with project timelines and quality standards.

Inputs	• Developed Artifacts from Incremental Development • Data from Incremental Development
Outputs	• Quality Assurance Reports • Feedback for Incremental Development Adjustments • Data for Monitoring and Visualising
Relationships	• Provides feedback to Incremental Development for Adjustments. • Sends data to Monitoring and Visualising.

Stakeholders and Roles	• Quality Assurance Engineers • Test Engineers • Software Developers • Project Managers

3.5. Recommend and Predict

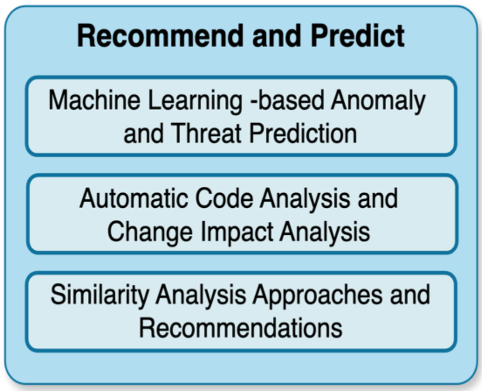
Recommend and Predict
Machine Learning -based Anomaly and Threat Prediction
Automatic Code Analysis and Change Impact Analysis
Similarity Analysis Approaches and Recommendations

Figure 7: Technical Areas Recommend and Predict

The Recommend and Predict stage focuses on applying machine learning (ML) techniques to enhance system predictability, anomaly detection, Offense (threat) prioritization, and code analysis. This phase integrates advanced algorithms for proactive error localization and anomaly detection, ensuring the system's efficient maintenance. **[Machine Learning -based Anomaly and Threat Prediction**] Our approach includes predictive models for locating anomalies and errors specifically within microservice architectures (NetRD), emphasizing ML-driven anomaly detection and prioritization for cybersecurity offenses (Ontario Tech, Glasshouse Systems). Additionally, telemetric data analysis helps identify patterns of potential anomalies, aiding rapid detection and response (Hoxhunt). **[Automatic Code Analysis and Change Impact Analysis]** This phase focuses on enhancing code quality and impact analysis through automation. It includes tools for sharing development insights by automatically gathering metrics from version control systems (ERSTE, DAKIK, Kuveyt Turk), predicting software faults within cloud-based architectures (Ontario Tech, Team Eagle), and simplifying software maintenance with automatic code analysis (University of Innsbruck, cc.com). Furthermore, tools for analyzing technical debt (Cape of Good Code, Vaadin) and assessing software quality trends assist in balancing quality improvement with feature development (FOKUS). **[Similarity Analysis Approaches and Recommendations]** By leveraging similarity analysis, this component utilizes graph-based techniques to offer recommendations (TWT, Software AG, Vaadin, Izertis, UC3M) and applies hierarchical modularization to check for model consistency (TWT, Akkodis). Additionally, ML-based approaches extract requirements from extensive data repositories and generate actionable recommendations (RISE, Alstom). Continuous ML-based learning further enhances software issue detection, promoting proactive system health (IFAK, Software AG).

Inputs	• Log source data • Code analysis data from version control • Architecture, code, and issue documentation • Software quality and technical debt metrics
Outputs	• ML-based predictions • ML-based recommendations • Automatic change impact and similarity analysis
Relationships	Utilizes data, visualization, and correlations from Quality Assurance and Requirements Engineering to adjust predictions and recommendations based on system feedback. This collaborative approach ensures insights that directly support Incremental Development, facilitating informed decision-making and continuous system improvement.
Stakeholders and Roles	Data Scientists, Quality Assurance Engineers, Development Team, Project Managers, Stakeholders

3.6. Monitoring and Visualizing

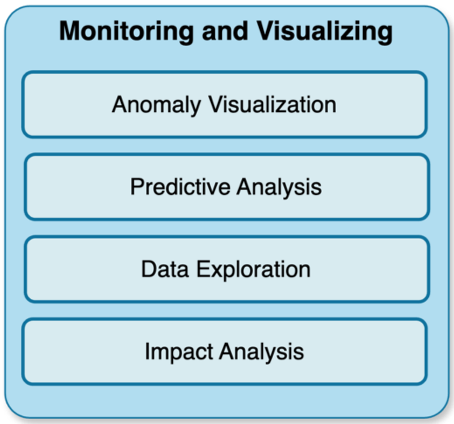

Figure 8: Technical Areas Monitoring and Visualizing

The Monitoring and Visualization stage utilizes inputs from quality assurance reports, static or dynamic analysis data, performance data, and component-level monitoring data. These pieces of data provide critical information that assists in crafting a visualization to effectively represent the system's health.

Inputs and Processes: Incoming data from Quality Assurance reports, static/dynamic analysis, performance metrics, and component monitoring get translated into an easily digestible visual format. These visual formats often come in the form of graphs, charts or interactive dashboards that make understanding and identifying patterns, trends, and anomalies more straightforward.

Outputs and Relationships: The primary outputs from this stage include delta analysis which highlights all changes made, Quality and performance trends that illustrate the system's behavior over time, correlation depiction between different metrics, and mapped dependencies and relationships between various components. This data visualization greatly contributes to the Recommendation and Prediction processes by providing clear, comprehensible and concise data correlations and patterns.

Techniques and Methods: The Monitoring and Visualization stage uses tools such as **anomaly detection** to highlight outliers or anomalies in the received data. By utilizing visualization tools or graphs, one can identify and highlight outliers or anomalies within a dataset. These significant points, which diverge from the established pattern, might indicate critical information or errors.
Graphs are used for **Predictive Analysis** to identify future trends or outcomes. Visualization allows us to project historical data into the future. It's a graphical representation to predict future trends or outcomes, making it easier to understand the prediction.
The use of data visualization can aid in understanding complex datasets. **Data Exploration** visualizes the complex data sets' hidden patterns, relationships, and insights that would be difficult to grasp in raw, unprocessed data.

Impact Analysis illustrates the effect or influence of a particular variable or action on different parameters. Displaying data visually can aid in understanding the impact or influence that a specific variable or action has on other factors. By manipulating these variables in a visualization, we can observe potential outcomes and trends.

Stakeholders and Roles: Quality Assurance Engineers, System Architects, Investors, Software Developers, and Project Managers all use the visualized data to make informed decisions and predict trends. These stakeholders can comprehend the application's state through the visualization and immediately identify problem areas that need their attention.

With visualization tools in place, all critical aspects of the data are highlighted and easily accessible to the relevant teams, promoting a more effective and efficient decision-making process.

Inputs	• Quality assurance reports • Static or dynamic analysis data • Performance data • Component-level monitoring data

Outputs	• Delta analysis • Quality and performance trends • Correlation of metrics • Dependencies and relationships of components to each other
Relationships	• Provides data, visualization and correlations to Recommend and Predict processes
Stakeholders and Roles	Quality Assurance Engineers, System Architects, Investors, Software Developers, and Project Managers

4. Mapping of the SmartDelta Tools to the Methodology

In the following table, we provide an overview of how the SmartDelta solutions are mapped to different stages of the methodology. These solutions can thus be used to implement and instantiate the methodology in different contexts and applications.

Tool Name	Owner (first position) and Partners (followed)	More information can be found in			Requirements Engineering					Incremental Development				Quality Assurance				Recommend and Predict			Monitoring and Visualizing			
		WP3	WP4	WP5	Requirements Elicitation and Extraction	Requirements Quality Analysis	Requirements Reuse Analysis and Allocation	Requirements Model Extraction	Requirements Verification	Code Reuse	Code Quality Improvement	Issue Triage and Resolution	Automated Model Generation and Extraction	Delta-Aware Test Generation	Test Amplification	Monitoring and Anomaly Detection	Static Analysis of Deltas	Machine Learning -based Anomaly and Threat Prediction	Automatic Code Analysis and Change Impact Analysis	Similarity Analysis Approaches and Recommendations	Anomaly Visualization	Predictive Analysis	Data Exploration	Impact Analysis
SoHist v2	UIBK, c.c.com	X	X	X							X						X		X					
Intelligent Issue Management Support Tool (INIMASU)	FOKUS		X	X		X		X				X						X	X			X	X	X
Call Graph Delta Analyzer	Özyeğin University (Subcontracted by Erste)	X	X	X							X		X			X	X		X	X	X			X
Mut4SLX	UAntwerpen	X												X	X	X								

Tool Name	Owner (first position) and Partners (followed)	More information can be found in			Requirements Engineering					Incremental Development				Quality Assurance				Recommend and Predict			Monitoring and Visualizing			
		WP3	WP4	WP5	Requirements Elicitation and Extraction	Requirements Quality Analysis	Requirements Reuse Analysis and Allocation	Requirements Model Extraction	Requirements Verification	Code Reuse	Code Quality Improvement	Issue Triage and Resolution	Automated Model Generation and Extraction	Delta-Aware Test Generation	Test Amplification	Monitoring and Anomaly Detection	Static Analysis of Deltas	Machine Learning -based Anomaly and Threat Prediction	Automatic Code Analysis and Change Impact Analysis	Similarity Analysis Approaches and Recommendations	Anomaly Visualization	Predictive Analysis	Data Exploration	Impact Analysis
Code Similarity Investigator	TWT		X							X										X				
Graph Similarity Recommender	TWT		X																	X				
SmartMetrics	Akkodis		X							X														
SmartTrace	Akkodis		X							X														
PyLC	MDU	X												X	X									
DRACONIS	MDU, ALSTOM	X		X					X	X	X						X							
SEAFOX	MDU	X		X				X						X	X									
Architecture Analysis and Visualization Tool	FOKUS			X								X					X							

Tool Name	Owner (first position) and Partners (followed)	More information can be found in			Requirements Engineering					Incremental Development				Quality Assurance				Recommend and Predict			Monitoring and Visualizing			
		WP3	WP4	WP5	Requirements Elicitation and Extraction	Requirements Quality Analysis	Requirements Reuse Analysis and Allocation	Requirements Model Extraction	Requirements Verification	Code Reuse	Code Quality Improvement	Issue Triage and Resolution	Automated Model Generation and Extraction	Delta-Aware Test Generation	Test Amplification	Monitoring and Anomaly Detection	Static Analysis of Deltas	Machine Learning -based Anomaly and Threat Prediction	Automatic Code Analysis and Change Impact Analysis	Similarity Analysis Approaches and Recommendations	Anomaly Visualization	Predictive Analysis	Data Exploration	Impact Analysis
Smellyzer	ARCELIK, Bilkent	X										X				X						X		
Relink	ARCELIK, Bilkent	X										X				X		X						
PieR	ARCELIK, Bilkent		X									X										X		
K2	FOKUS	X											X	X	X									
CBTS	FOKUS	X												X					X					
Jazure	ARCELIK, Bilkent	X					X		X															
Metric Dashboard	ARCELIK, Bilkent			X																	X		X	X
DIA4M	ORION/NETRD	X	X	X								X				X		X			X		X	
DETANGLE	Cape of Good Code, Dakik, TURK BANK	X	X	X							X	X				X	X				X			

Tool Name	Owner (first position) and Partners (followed)	More information can be found in			Requirements Engineering					Incremental Development				Quality Assurance				Recommend and Predict			Monitoring and Visualizing			
		WP3	WP4	WP5	Requirements Elicitation and Extraction	Requirements Quality Analysis	Requirements Reuse Analysis and Allocation	Requirements Model Extraction	Requirements Verification	Code Reuse	Code Quality Improvement	Issue Triage and Resolution	Automated Model Generation and Extraction	Delta-Aware Test Generation	Test Amplification	Monitoring and Anomaly Detection	Static Analysis of Deltas	Machine Learning -based Anomaly and Threat Prediction	Automatic Code Analysis and Change Impact Analysis	Similarity Analysis Approaches and Recommendations	Anomaly Visualization	Predictive Analysis	Data Exploration	Impact Analysis
GHS anomaly detection and Offense Prioritization tools	Glasshouse Systems and Ontario Tech University)		X													X		X			X	X		X
SONATA	Izertis, UC3M	X		X				X					X	X						X			X	
YATAP A Tool for Change Impact Analysis	Erste		X	X							X	X					X		X		X	X		
Modernization Toolkit	Vaadin		X	X						**X**			**X**						**X**	**X**				X
AirOPs	Ontario Tech University, Team Eagle		X								X	X							X			X		
NALABS	MDU, ALSTOM, ADDIVA	X		X		**X**			**X**															

Tool Name	**Owner (first position) and Partners (followed)**	**More information can be found in**			**Requirements Engineering**					**Incremental Development**				**Quality Assurance**				**Recommend and Predict**			**Monitoring and Visualizing**			
		WP3	**WP4**	**WP5**	**Requirements Elicitation and Extraction**	**Requirements Quality Analysis**	**Requirements Reuse Analysis and Allocation**	**Requirements Model Extraction**	**Requirements Verification**	**Code Reuse**	**Code Quality Improvement**	**Issue Triage and Resolution**	**Automated Model Generation and Extraction**	**Delta-Aware Test Generation**	**Test Amplification**	**Monitoring and Anomaly Detection**	**Static Analysis of Deltas**	**Machine Learning -based Anomaly and Threat Prediction**	**Automatic Code Analysis and Change Impact Analysis**	**Similarity Analysis Approaches and Recommendations**	**Anomaly Visualization**	**Predictive Analysis**	**Data Exploration**	**Impact Analysis**
GW2UPPAAL	MDU	X						X						X										
TIGER +	MDU	X						**X**						**X**	**X**									
ReForm	IFAK, Akkodis		X					X					X											
AISA	IFAK, Software AG		X				**X**					**X**								**X**				
AILA	IFAK, Software AG		X				X					X								X				
TCG	IFAK		X											**X**										
Telemetry Anomaly Analyzer	Hoxhunt		X	X														X			X		X	X

Additional details on these tools can be found in Deliverable ‘D2.5 - SmartDelta Methodology Implementation Toolset’ (https://itea4.org/project/workpackage/deliverable/document/download/432/SmartDelta%20D2.5%20-%20SmartDelta%20Methodology%20Implementation%20Toolset.pdf)

5. Industrial Application Examples

In the following subsections, we provide examples of how the SmartDelta Methodology have been applied and implemented in different industrial use cases and to solve their challenges.

5.1. Enterprise Software – Software AG

In the following sections, we describe Software AG's use case and its application of the SmartDelta Methodology to address key challenges in Software AG's development process for Enterprise Software.

5.1.1. Introduction and Background

Enterprises are highly dependent on the availability and reliability of their software in today’s world. Many functions become impossible or severely slow down when the software cannot be used or trusted. Software AG recognizes its important place in the industry as a supplier of world-class enterprise software and strives to provide its customers with high levels of security, reliability, and quality in the software it provides.

This led to the development of specific policies, controls, and procedures within the company that rely on design, development, and testing artifacts to ensure given characteristics of the products. This decade-long focus of the company naturally resulted in high quality, reliable and secure software for enterprises. However, the artifacts and accompanying controls are often static and may not well support the intent of the company to provide continuous improvement.

Software AG delivers a multitude of products both for on-premises and cloud use. These products are delivered by thousands of people, producing millions of artifacts. Every event in the lifecycle of a product is recorded and can be referenced. This daily increasing information could be used to automatically deliver important insights into the current trends in software quality and security supporting Software AG’s intention to continuously improve the quality of the software and set the best possible quality and security standards for the industry.

5.1.2. Challenges

Software AG faces several challenges in the production of enterprise software, especially related to the massive amount of code and other artifacts accumulated over time. We expect these challenges are universal for any company that develops a non-trivial amount of software.

1. **Repetition of design and code:** There is a large amount of possible reuse of code fragments and components that is not possible to tackle in a manual way due to the sheer size of the source base. For instance, there can be issues similar to something already being developed and developers could save the effort if they would be aware of this.
2. **Issue classification:** Security-related issues (feature requests, bug reports, etc.) are of utmost importance and must be prioritized as they have a major impact on the quality and reliability of the software. However, determining whether an issue is security-relevant or not is quite difficult and usually requires manual assessment by an expert which takes time. An automated approach for assigning security labels to issues could reduce the number of incorrect assignments and speed up the processing of issues.
3. **Code quality:** Maintaining high code quality is a key aspect for delivering secure, reliable and performant software. A common problem in this context is that the code produced could be sub-optimal and there could be "a better way" but the developers are not aware of this. One potential way to solve this problem is through training but an automatic system pointing

out changes leading to sub-optimal performance, e.g., based on "Lessons Learnt", could potentially improve the code even further.

5.1.3. Implementing the SmartDelta Methodology for enabling reuse and quality improvements

The SmartDelta Methodology provides guidance for tackling these challenges in a structured way. Software AG’s use case makes use of the stages Requirements Engineering (RE) (Section 3.2) and Incremental Development (Section 3.3) to enable reuse of existing software artifacts and automatic classification of issues, while stage Quality Assurance (Section 3.4) allows to utilize existing fixes to avoid repetition of errors. The corresponding SmartDelta tools enabling these features through artefact similarity analysis and machine learning are covered in step Recommend and Predict (Section 3.5). In addition, we use stage Monitoring and Visualizing (Section 3.6) to visualize information for the management that is related to the previous stages and steps.

Requirements Engineering - Requirements Reuse Analysis and Allocation
In this stage, we process new incoming issues such as new feature requests and bug reports to classify them for faster treatment by corresponding experts and to identify similar existing issues that may lead to potentially reusable software artefacts:

1. **Issue Classification:** Security-related issues are of utmost importance and must be prioritized as they have a major impact on the quality and reliability of the software. However, determining whether an issue is security-relevant or not is quite difficult and usually requires manual assessment by an expert which takes time. We address this problem by automatically assigning security labels using the Automatic Issue Labeling Tool (AILA) developed by IFAK. If the tool marks an issue as security-related, it will automatically be prioritized for faster processing by the corresponding experts. Still, all remaining incoming issues will continue to be analyzed by a security expert to ensure that no security-related issues are missed. Thanks to AILA's high detection rate, however, the processing of security-related issues can be significantly accelerated.
2. **Issue Similarity Analysis:** The new incoming issues are compared with the existing issues stored in our issue database to find similarities that could indicate potentially reusable artifacts. The similarity analysis is done by the Automatic Issue Similarity Analysis Tool (AILA) developed by IFAK. For each new issue, we select the 10 most similar issues reported by AILA for further processing. These results are subsequently visualized in a web application for further decision making. It shows the currently investigated new issue next to a similar issue reported by AILA selected by the user. Similar text elements can optionally be highlighted. If the user decides that the reported similarity is correct, this pair of new and existing issues is marked for reuse and will be processed further in the Incremental Development stage

Quality Assurance
As described above, another challenge in our use case is to use existing fixes and corresponding code changes to avoid repeating similar errors. For example, a newly implemented code in product A could be affected by the same problem that was observed and fixed in product B.

We tackle this challenge by combining the steps *Static Analysis of Deltas* and *Similarity Analysis Approaches and Recommendations* of the Quality Assurance and Recommend and Predict stages, respectively. For instance, we employ the SmartDelta tool Code Similarity Investigator developed by TWT to compare new code with code that has been fixed in the past. If the tool reports a high similarity, the new code could be affected by a problem similar to the one already fixed. This helps to avoid similar errors across different products, projects, and development teams. In addition, the identified fixes can provide hints for the implementation of better code.

Incremental Development
In this stage, we make use of the results obtained in the Requirement Engineering and Quality Assurance stages for enabling code reuse and to improve code quality.

1. **Code Reuse:** For each pair of new and similar existing issues identified in the Requirement Engineering stage, it is now determined whether the existing code and tests associated with the existing issue can be reused to implement or solve the new issue. This step is carried out manually by an expert, ideally by the developers who implemented the code and tests to be reused. They decide whether to use the existing artefacts as-is, modify them, or discard them in favor of a new implementation.
2. **Code Quality Improvement:** The identified code fragments similar to code that already have been fixed are used to provide developers with a starting point for improving the quality of their code based on existing knowledge and fixes. The developers can compare their code with the found similar code fragments to determine whether they should revise their code and if so, they can use the corresponding fixes as implementation basis.

Monitoring and Visualizing
In this last stage, we use a dashboard along with standard visualizations to provide information and insights about the previous stages and related KPIs to the management. This includes, e.g., a line chart visualizing the quality trend of selected products over time or a bar chart illustrating the number of already used and potentially reusable software artefacts.

5.1.4. Key Tools and Their Roles

- **AILA – Automatic Issue Labeling Tool**:
  AILA developed by IFAK automates issue labelling using fine-tuned BERT models. It classifies requirements or issues based on descriptions written in natural language, aiding in prioritization and team assignments. In Software AG's use case, it is used to automatically identify security-related issues for faster treatment.
- **AISA - Automatic Issue Similarity Analysis Tool**:
  AISA developed by IFAK automates issue similarity analysis using language models such as Sentence-BERT. This tool provides similar requirements or issues based on descriptions written in natural language, aiding in code and test reuse recommendations. In this use case, AISA is used to find existing issues that are similar to new issues that might point to potentially reusable design artifacts, code and tests.
- **CSI - Code Similarity Investigator**:
  The CSI tool developed by TWT allows to measure the similarity of two code sections (classes, methods). This is done by transforming code into Code Property Graphs (CPGs) and performing graph algorithms to calculate similarity scores. Here, the tool is used to compare new code with existing code that is known to be affected by some kind of problem, such as a bug or security vulnerability.

5.1.5. Benefits of the SmartDelta Methodology

Applying the SmartDelta methodology and the associated SmartDelta tools to our development process yield the following benefits:

- Reusability: Structured guidelines along with tools automatically detecting similar issues (feature requests, bug reports, etc.) enable reuse of existing artefacts such as design, code and tests. This would not be possible if done manually, considering an already extensive and still growing issue database. This allows us to reduce the amount of repetition of design and code, effectively saving time and costs.

- Quality Improvements: The guidelines provided by the SmartDelta Methodology in combination with associated automated tools help to identify error-prone code, e.g., through comparison with code that has been fixed in the past. This reduces the risk of repeating the same errors, which ultimately improves the overall quality of the software. Moreover, the automatic detection of security-related issues helps to speed up the treatment of critical issues.

5.1.6. Conclusion

Identifying and using reusable software artifacts to reduce repetition and leveraging existing fixes for future error prevention is still a challenge in modern software development, especially when working with massive amounts of software artifacts created for numerous projects and products by different development teams. Software AG's use case illustrates the practical application of the SmartDelta Methodology to tackle these challenges. By following the structured guidelines provided by the SmartDelta Methodology and using the associated SmartDelta tooling, it is possible to overcome these challenges in an automated way, resulting in a significant reduction in manual effort, costs, time-to-market and improved code quality.

5.2. Railway Domain - Alstom

5.2.1. Introduction and Background

Within the SmartDelta project, Alstom is a use-case provider and supports the evaluation of the proposed SmartDelta tools and processes to determine the performance of the individual solutions (i.e., as they were intended to be used) and performance on the overall software development process.

Three main challenges have been identified by Alstom to be tackled in SmartDelta and they are described in the next section as User Stories A, B, and C. In this section, we present how these User Stories are mapped on the overall SmartDelta methodology.

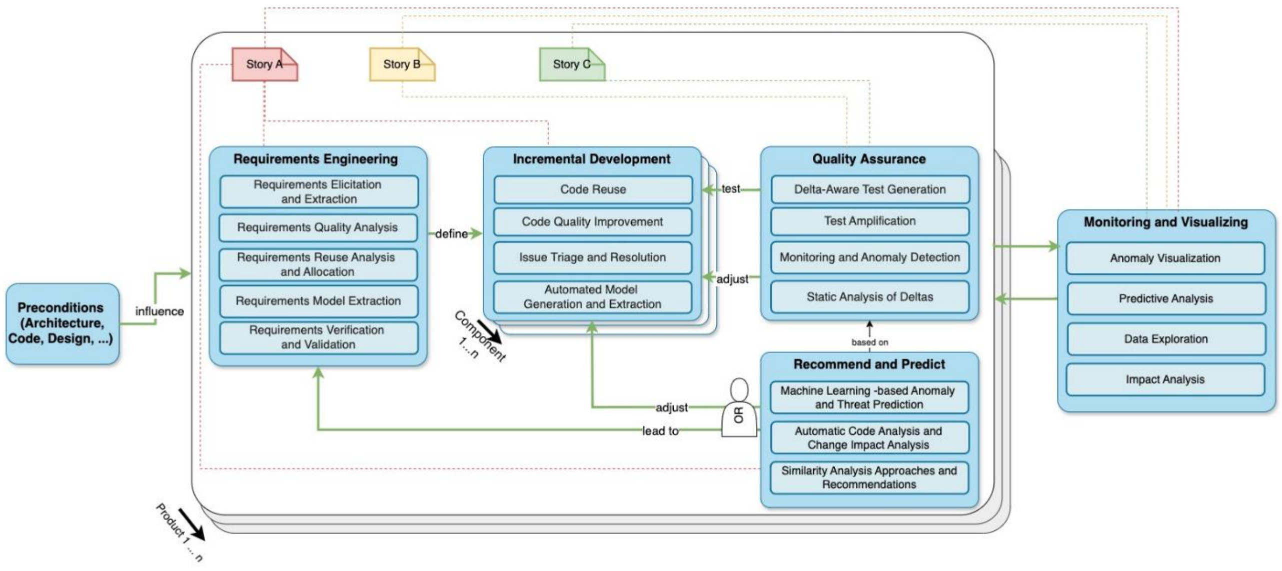
Story A
Story B
Story C
Requirements Engineering
Requirements Elicitation and Extraction
Requirements Quality Analysis
Requirements Reuse Analysis and Allocation
Requirements Model Extraction
Requirements Verification and Validation
define
Incremental Development
Code Reuse
Code Quality Improvement
Issue Triage and Resolution
Automated Model Generation and Extraction
test
adjust
Quality Assurance
Delta-Aware Test Generation
Test Amplification
Monitoring and Anomaly Detection
Static Analysis of Deltas
based on
Recommend and Predict
Machine Learning -based Anomaly and Threat Prediction
Automatic Code Analysis and Change Impact Analysis
Similarity Analysis Approaches and Recommendations
Monitoring and Visualizing
Anomaly Visualization
Predictive Analysis
Data Exploration
Impact Analysis
Preconditions (Architecture, Code, Design, ...)
influence
adjust
lead to
OR
Component 1...n
Product 1 ... n

Figure 9: Mapping of the SmartDelta Methodology on the Alstom use-case.

5.2.2. Challenges (before SmartDelta)

In this section, we describe the 3 User Stories included in Alstom use-case.

User Story A: Delta between product line features and customer needs.

In the railway industry, most projects often start with a customer publishing a call for tender. Due to the competitive nature of the industry, a quick response to the call with an overview of a technical solution is often a prerequisite to acquiring projects. Therefore, identifying technical specifications in the large tender documents enables a quick response to the call for tender. In addition, it also enables the feasibility analysis of the project as the novelty of the extracted technical specification could be computed by comparing it with existing projects.

Alstom meets new customer requirements through modifying previous project solutions or modifying standardized "product" hardware and software solutions known. Customer requirements are analyzed by advanced engineers for correlation and differences with our standard product and past projects, sometimes within a very short timeframe due to bid submission and development deadlines. During this process, the requirements manager has to manually annotate the technical specifications in the large tender documents. In addition, if the project is acquired successfully, the technical specifications also have to be allocated to various sub-system teams for implementation and testing. Finally, since the company develops similar products, a reuse analysis must also be conducted to identify reuse opportunities for existing software that could realize the allocated requirements. However, this process for requirements engineering is time-consuming, prone to human error, and is dependent on the experience of a few key engineers. Tools addressing this particular user story are needed to aid the bidding and implementation phases.

User Story B: Functional requirements quality and verifiability.

To ensure the customer requirements are verified, every customer requirement is broken down and linked to the relevant new and existing features. The features relevant to software control are broken down and linked to control requirements. All software requirements are linked to the relevant software implementation within the control software. To ensure the implemented software satisfies each requirement, at least one test is performed on the software, known as a functional test. The functional tests enable the verifier to demonstrate the software behaves as specified in the requirements. Testers must review the requirements and create relevant test cases which will sufficiently test the control software. Tools that automatically generate test cases from requirements and execute these tests to verify both the requirements and the technical artifacts (i.e., software) contribute significantly to the quality and validation of functional requirements.

User Story C: Code quality and Delta between manual and automatic test review, design review and code review

When designing software control solution with a block diagram approach, the designs should be developed according to a specific set of rules and standards to ensure a higher reliability of the code generated and readability of the design as well as adhering to the requirements from the railway standards such as EN50128 and EN50657 for compliance. These detailed quality-rules, such as maintaining cohesive naming conventions or following established coding guidelines are checked manually once the implementation of software is completed. Tools that automatically performing some design and code reviews performed manually today could improve the quality verification of visual block-based programming languages,

5.2.3. Implementing the SmartDelta Methodology

To address the existing tool limitations described in the Use Stories, the following solutions were proposed during the SmartDelta project. A description of each tool is included in the next section.

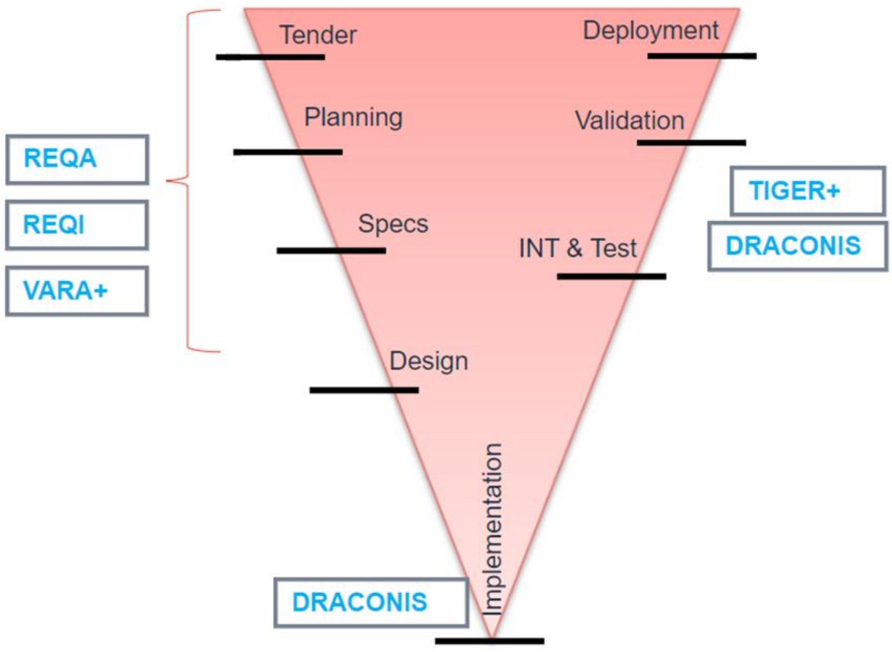

Figure 10: The SmartDelta Methodology for the Alstom use-case mapped to the V model

5.2.4. Key Tools and Their Roles

REQ-I – a tool for customer needs identification for tender documents
ReqIdentifier (REQ-I) formulate the requirement identification problem as a binary text classification problem. It uses various state-of-the-art classifiers based on traditional machine learning, deep learning, and few-shot learning for requirements identification from large tender documents. , the tool can process a text entry or a PDF document to extract text from it using Optical Character Recognition (OCR). Once the all the textual entries of the tender documents are available, a BERT language model-based classification pipeline is fine-tuned on the input text. In query mode, text or a PDF file can be given as input to the tool, and it outputs a PDF file with highlighted requirements.

REQA – a tool for allocation of requirements to teams for implementation
Once requirements are identified, they need to be allocated to various development and testing teams for implementation. The REQA tool combines traditional AI with deep learning to allocate requirements and generate supplementary information to support engineers in well-informed allocation. The REQA tool has two modules named Assigner and Augmenter:

- **The Assigner** module uses large language models with statistical classification to recommend the allocation of the requirements to various teams that are likely to accept the allocation,
- **The Augmenter** module uses lexical similarity-based clustering to generate cased based explanations to support the recommendations of Assigner and in turn, a well-informed allocation.

VARA+ – a tool for reuse identification
VARA is variability-aware requirements reuse analysis method which aims to automate product's assets reuse analysis and thus helps teams achieve quick and quality delivery of software systems. The tool uses state-of-the-art natural language processing and machine learning algorithms to

predict existing product's assets that can be reused to realize the new customer requirements. In addition, VARA also compute various metrics (readability index, complexity and subjectivity etc.) on the quality of requirements that help in writing better requirements.
VARA takes all existing requirements and their links to software components implementing them, as input to fit/train a content-based recommender--- driven by clustering. In query mode (steps can be followed with blue arrows), unseen customer requirements are used as input to recommend reuse based on similarity with neighboring existing requirements.

TIGER+ – a tool for test case generation and execution
TIGER uses the model-based testing concept to perform the concretization of abstract test cases and the generation of test scripts. It consists of three parts:

- **Abstract Test Case Generator**: GW (Graph Walker) takes as an input the model file in JSON/GraphML format and generates the abstract test cases by traversing through the model elements (i.e. states and transitions) based on a generator algorithm (such as random, quick_random, Astar, etc.) and a stopping condition (such as edge_coverage, vertex_coverage, etc.).
- **Abstract Test Case Generator**: The test case concretizer converts abstract test cases into concrete by mapping the logical signal names with their technical counterparts and corresponding values. Testers and developers use these logical names as initial names of the signals in the early phases of development. Later in the development, technical signal names became available that represent the actual signal names used by the SUT for its normal operations. Hence, the test case concretizer extracts the test data i.e. (variable names and their respective values) from the generated abstract test cases (available in a JSON file), extracts the required information about technical signal names from an XML file, and maps the logical signal names with technical signal names and their corresponding values based on defined mapping rules.
- **Test Script Generator:** Once the abstract test cases are converted into concrete test cases, the test script generator generates the test script in C# language using the implementation details of the SUT (i.e. script format, libraries, and methods to be executed on the target test execution platform, SIL & HIL). The generated test script contains two types of steps for each test case, forcing the input signals and verifying the expected output signals, to validate the expected behavior of the SUT.

DRACONIS – a framework for static analysis
DRACONIS is a static analysis framework. It is separated into three steps: Intermediate representation generation, analysis and reporting.

- **The intermediate representation** is generated either directly through model transformations from the source models, or by converting the models to JSON, which is then parsed by the framework .
- **The analysis** core supports analyses based on metrics or dataflow information. The tool supports design requirement checking by instantiating requirements as analysis rules (commonly called “checkers”) as a named combination of queries. Checkers are defined in configuration files. This allows multiple user configurations to be used, supporting for instance low cost checks that may be run on every change to more extensive configurations which are part of the final validation work. After the analysis is performed, the tool stores the model instance and the analysis report in a database. In cases where the model is then changed, a delta analysis is performed to recommend what analyses will need to be re-run.
- The tool can **produce reports** in multiple variants. This covers everything from a purely tekst based report that may be shown to the user to a structured report with a rendered image of the model. The usage of DRACONIS allows the automated generation of reports based on existing

design rules, which in turn will allow manual validation effort to focus on cases where a human is most needed.

5.2.5. Benefits of the SmartDelta Methodology

Each tool included in the Alstom use-case has been evaluated in SmartDelta. In this section, we provide a short overview of this evaluation.

REQ-I. Results from the evaluation on the Alstom use-case show that the tool could identify requirements in large documents with an average F1 score of 0.82%. Our results also confirm that few-shot classifiers can achieve comparable results with an average F1 score of 0.76 on significantly lower samples, i.e., only 20% of the data.

REQA. Results from the evaluation show that REQA can allocate the requirements to different teams with a 76% F1 score when considering requirements allocation to the most frequent teams. Information augmentation provides potentially useful indications in 76% of the cases.

VARA. Evaluation of VARA+ shows that the tool can recommend reuse with an average accuracy of around 82% and can reduce the lead time of the propulsion software system. In addition, the qualitative evaluation also shows that the recommendations produced by the tool are valuable and insightful.

TIGER. The evaluation within the SmartDelta methodology focused on optimizing model-based test suites using machine learning for delta-focused testing. In one study, TIGER+ was employed to reduce test suite sizes by over 80% while maintaining a high fault detection rate (87%-100%). This approach improved testing efficiency and maintained fault coverage comparable to manually created suites. Another study validated TIGER, showing its capability to generate fully executable test scripts with 100% requirements coverage and fault detection effectiveness through a software-in-the-loop platform. Both studies significantly improved industrial testing efficiency and fault detection for iterative updates within Alstom.

DRACONIS. DRACONIS supports configurable checks of different design rules, including naming conventions, metrics and dataflow analysis. Results are visualised in a textual form, with an optional rendering of the model for stand-alone report generation. The tool supports “edit-time” checking, allowing for rapid feedback and early discovery of potential issues during development. The primary use case is to alleviate the review process of systems in safety-aware contexts.

5.2.6. Conclusion

The SmartDelta methodology has shown results in addressing challenges faced by Alstom in requirements engineering, test case generation, and code quality assurance. Tools like REQ-I and REQA have been used to identify and allocate requirements, achieve good accuracy (F1 scores of 0.82 and 0.76, respectively), and improve efficiency in handling large tender documents and team assignments. VARA+ has improved reuse analysis, with an 82% accuracy in recommending reusable components, reducing lead times and improving delivery quality. TIGER has optimized test case generation, reducing test suite sizes by over 80% while maintaining fault detection rates of 87% to 100%, significantly improving testing efficiency for iterative updates. DRACONIS has introduced automated and configurable static code analysis, showing early error detection and helping the manual review burden in safety-critical contexts. Together, these tools and processes illustrate how SmartDelta's methodology integration into Alstom's workflow improves overall software development efficiency, quality assurance, and compliance with industry standards.

5.3. Smart Industry - Akkodis

Below, we present a summarized overview of the use case and its application of the SmartDelta Methodology to address key challenges faced by modern software development teams. A comprehensive description of the Akkodis use case is available in the public deliverable D1.6.

5.3.1. Background and General Assumptions

Today’s software development landscape is increasingly characterized by:

Rising Complexity: Software systems are becoming more intricate, incorporating numerous components, dependencies, and technologies.

- **Tight Time and Budget Constraints**: Projects often operate under pressure to deliver quickly, without compromising on cost.
- **Flexibility Requirements**: Development teams need to rapidly adapt to changing requirements, market conditions, or stakeholder expectations.
- **Need for Efficiency and Quality**: Achieving high-quality software within limited resources demands smarter processes and tools.

One promising avenue for meeting these challenges is the reuse of existing software components—whether at the project, functional, artefact, or library level. Reuse can drive both efficiency and quality by leveraging proven, tested solutions rather than reinventing the wheel for every new requirement.

5.3.2. Challenges in Software Reuse

However, while reuse offers significant benefits, it also introduces its own challenges:

- **Identification of Reusable Assets**: Development teams need effective methods to locate relevant components, which may be buried within large, scattered repositories or legacy systems.
- **Integration of Reuse into Daily Workflows**: A structured methodology is necessary to ensure that reuse becomes an integral part of the development lifecycle, rather than an ad-hoc effort.
- **Maintaining Quality During Reuse**: Reusing artefacts requires careful evaluation to ensure they meet current quality standards and are compatible with the new context.

5.3.3. Implementing SmartDelta for Incremental Development

The Akkodis use case tackles these challenges by implementing the SmartDelta Methodology within its development process. This methodology, as detailed in Section 3.3 on Incremental Development, introduces a structured, iterative approach that integrates reuse at multiple stages of the development lifecycle. Below is a brief outline of how Akkodis applies these principles:

1. Retrieving and Filtering Software Artefacts

The first step in the process is to identify and retrieve relevant software artefacts from various sources such as code repositories, documentation, and issue trackers. Akkodis has developed two specialized tools to support this phase:

- **SmartMetrics**: This tool scans and indexes Git repositories, storing all relevant artefact data in a structured database. This indexed data is later used for semantic search and to build a Retrieval-Augmented Generation (RAG) pipeline. SmartMetrics also calculates basic software quality metrics, such as code complexity and maintainability, to provide insights during the reuse decision-making process.
- **SmartTrace**: Operating on the SmartMetrics database, SmartTrace functions as a retriever. It efficiently finds and ranks software artefacts relevant to a given query,

helping developers quickly access the components most applicable to their current tasks.

These tools streamline the artefact retrieval process, ensuring that the development team has access to high-quality, relevant components. Detailed evaluations and limitations of these tools are discussed in deliverable D1.6.

2. Analysing Impact and Quality

Once artefacts are retrieved, the next step involves analyzing their impact and quality. Akkodis uses this phase to assess:

- **Impact on Existing Systems**: Understanding how new requirements or changes will affect current software components and system behavior.
- **Software Quality Metrics**: Incorporating metrics like code complexity, test coverage, and maintainability into the analysis to ensure long-term software health.

Akkodis has extended this analysis to operate on a model level:

- **Automatic Model Generation**: New requirements are translated into new or modified models (e.g., UML state machines) using ReForm, a tool developed by IFAK. This allows developers to explore the impact of changes on a high level without delving into implementation details prematurely.
- **Delta Analysis and Similarity Scoring**: The GSR tool by TWT compares newly generated models with existing ones, highlighting differences (deltas) and providing similarity scores for models and their branches. This analysis helps developers assess the scope and nature of changes.
- **Visualization of Models and Deltas**: Visual tools (e.g., provided by Fraunhofer FOKUS) present these changes clearly, enabling better understanding and communication of their impact.

This step helps teams balance implementation effort and quality, supporting decisions that optimize for long-term maintainability and minimize technical debt.

3. Deciding on Reuse Options

With a clear understanding of the impact and quality of potential changes, developers and architects can now decide how to proceed. This decision-making process involves evaluating various reuse and implementation options under constraints such as:

- **Budget and Timeline**: Ensuring that the chosen path aligns with project constraints.
- **Strategic Considerations**: Balancing immediate needs with long-term goals, such as maintaining architectural integrity or minimizing future maintenance costs.

The methodology provides structured support for these discussions by grounding decisions in data from the analysis phase.

4. Implementing Changes

Once decisions are made, the planned changes can be implemented. The SmartDelta Methodology is flexible and supports various workflows, depending on the process and toolchain used:

- **Automatic Code Generation**: Some changes can be generated automatically based on models.
- **Manual Implementation**: In cases where automated tools are insufficient, developers can implement changes manually.
- **Hybrid Approach**: Combining automation and manual work, allowing teams to optimize effort depending on the context.

This phase ensures that reused components are integrated seamlessly into the new system, whether changes are applied at the model level or directly in the codebase.

5. Inspecting and Validating Changes
The final phase focuses on validating the implemented changes and their impact:

- **Visual Dashboards and Reports**: Tools provide visual insights into changes, showing deltas between new and existing models or code, as well as their influence on quality metrics.
- **Quality Assurance**: This step takes a broader view, validating multiple changes against the original requirements. It also evaluates the evolution of software quality metrics over time, ensuring that the system maintains its intended performance and reliability.

The ability to continuously monitor and validate software quality metrics ensures alignment with project goals and supports continuous improvement. This feedback loop prepares the system for the next iteration, maintaining a balance between innovation and stability.

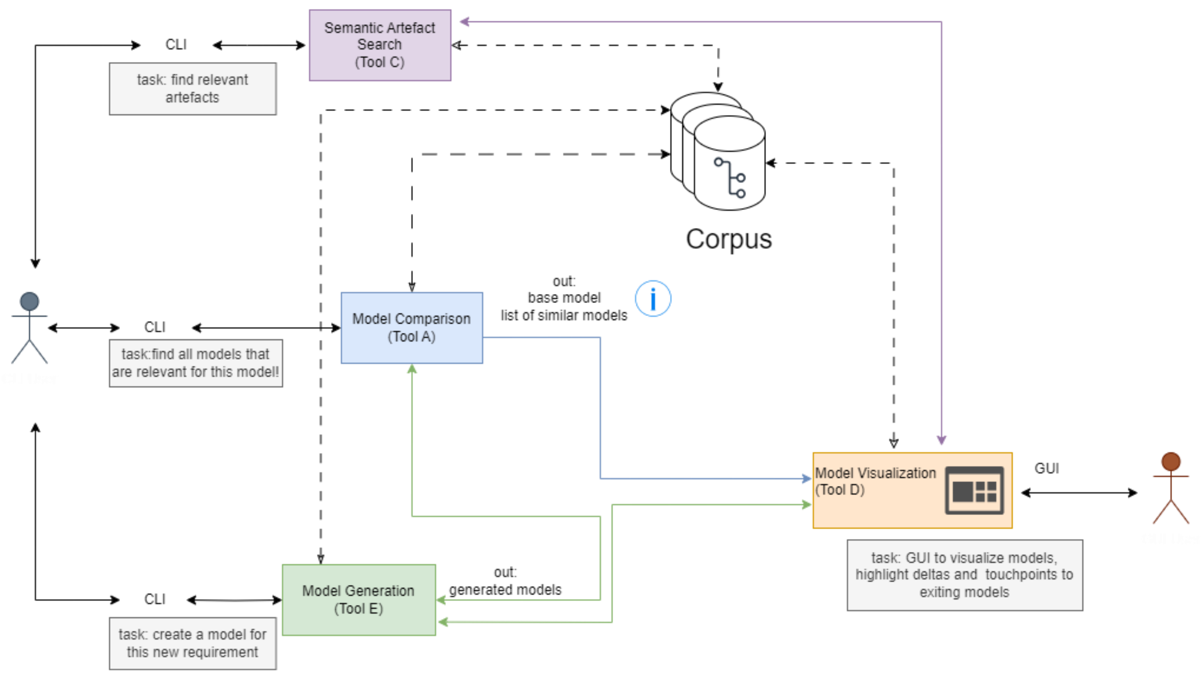
CLI
task: find relevant artefacts
Semantic Artefact Search (Tool C)
Corpus
CLI
task:find all models that are relevant for this model!
Model Comparison (Tool A)
out: base model list of similar models
Model Visualization (Tool D)
GUI
task: GUI to visualize models, highlight deltas and touchpoints to exiting models
CLI
task: create a model for this new requirement
Model Generation (Tool E)
out: generated models

Figure 10: Akkodis use case tools in SmartDelta Methodology

The diagram above illustrates the tools utilized in the Akkodis use case and their interconnections within the SmartDelta Methodology. This setup provides significant flexibility, allowing tools to be configured and applied in different sequences depending on the specific needs of the development task.

5.3.4. Key Tools and Their Roles

- Semantic Artefact Search (Tool C):
  This tool initiates the process by performing a semantic search across repositories to retrieve relevant software artefacts. It lays the groundwork for subsequent analysis by filtering potential candidates based on relevance.
- Model Generation (Tool E):
  Based on new requirements, this tool generates updated or entirely new models, such as

UML state machines, that reflect the intended system behavior. These models serve as a baseline for comparison and impact analysis.

- Model Comparison (Tool A):
  Primarily used to analyze differences (deltas) between the base models and newly generated models, this tool helps developers identify key changes. Importantly, it offers additional functionality and can be repurposed for other steps, such as re-ranking artefacts retrieved in the first phase.
- Model Visualization (Tool D):
  This tool enables stakeholders to visualize models and delta information. It highlights connections and touchpoints with existing models, providing clear insights through a graphical interface. This is crucial for understanding system-wide impacts and making informed decisions.

Flexibility and Workflow Variations
The methodology allows tools to be utilized in various combinations:

- Standard Workflow: Following the sequence of retrieving artefacts, generating models, and then comparing and visualizing them.
- Alternative Workflow: The model comparison tool can be integrated earlier in the process, enhancing artefact filtering by re-ranking results based on similarity to existing models.

By offering such adaptability, this approach ensures that development teams can tailor the process to optimize efficiency, maintain quality, and meet project-specific constraints.

Benefits of the SmartDelta Approach in Akkodis
By integrating SmartDelta into its workflow, Akkodis aims to achieve several key benefits:

- Enhanced Efficiency: Automated tools and structured processes reduce the time spent on locating, analyzing, and integrating reusable components.
- Improved Quality: Reusing proven components helps maintain high standards of software quality while reducing the risk of introducing defects.
- Greater Flexibility: The incremental development approach supports adaptability, allowing teams to respond quickly to changing requirements.
- Scalable Methodology: The SmartDelta Methodology is designed to scale across projects of varying complexity, ensuring consistent results in diverse development environments.

5.3.5. Conclusion

The Akkodis use case highlights the practical application of the SmartDelta Methodology in addressing the challenges of modern software development, such as rising complexity, constrained resources, and the need for flexibility. By embedding reuse and quality management into every phase of the development lifecycle, Akkodis ensures a sustainable model for delivering high-quality software efficiently. The combination of advanced tools like SmartMetrics, SmartTrace, and specialized modeling solutions demonstrates how automation and data-driven decision-making can optimize incremental development processes.

6. Evolution of the SmartDelta Methodology

The following figure illustrates how the SmartDelta Methodology has evolved over the period of the project to reach its final version as described in this deliverable.

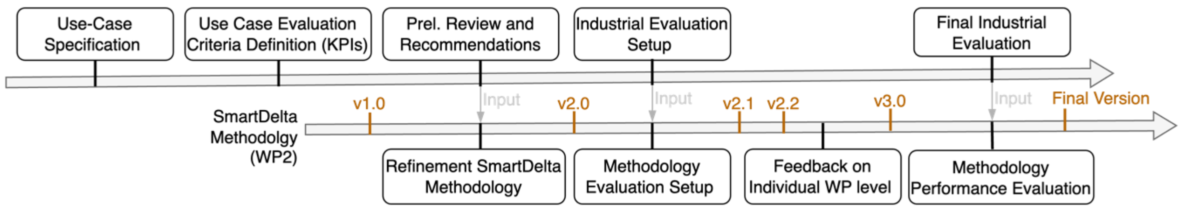
Use-Case Specification
Use Case Evaluation Criteria Definition (KPIs)
Prel. Review and Recommendations
Industrial Evaluation Setup
Final Industrial Evaluation
SmartDelta Methodolgy (WP2)
v1.0
Input
v2.0
Input
v2.1
v2.2
v3.0
Input
Final Version
Refinement SmartDelta Methodology
Methodology Evaluation Setup
Feedback on Individual WP level
Methodology Performance Evaluation

Figure 11: Evolution of SmartDelta Methodology

References

[1] D. Pandey, U. Suman, and A. K. Ramani, ‘An Effective Requirement Engineering Process Model for Software Development and Requirements Management’, in *2010 International Conference on Advances in Recent Technologies in Communication and Computing*, Kottayam, India: IEEE, Oct. 2010, pp. 287–291. doi: 10.1109/ARTCom.2010.24.

[2] M. Mohammad and V. Alagar, ‘A component-based development process for trustworthy systems’, *J. Softw. Evol. Process*, vol. 24, no. 7, pp. 815–835, Nov. 2012, doi: 10.1002/smr.472.

[3] J. Tian, *Software quality engineering: testing, quality assurance, and quantifiable improvement*. Hoboken, NJ: Wiley-Interscience, 2005.

[4] L. Bogaards, ‘The Different Levels of Monitoring: A Monitoring Maturity Model’, dzone.com. Accessed: May 01, 2024. [Online]. Available: https://dzone.com/articles/the-different-levels-of-monitoring

[5] M. Gasparic and A. Janes, ‘What recommendation systems for software engineering recommend: A systematic literature review’, *J. Syst. Softw.*, vol. 113, pp. 101–113, Mar. 2016, doi: 10.1016/j.jss.2015.11.036.