A.2 Requirements after Simplified Volere Template

This section contains the requirements that have been created during the RE phase. The requirements have been derived from the interview results in Appendix A.1. The requirement descriptions and structure are derived from the Volere Template [RR07], however, the schemata have been simplified. Further, instead of the customer satisfaction in numbers, the customer satisfaction for functionality and dysfunctionality of a functional requirement is represented from "- –" to "++", as described in the Kano model [KSTT84].

A.2.1 Project Drivers

Purpose

This project purposes allowing performance and chaos engineers to quickly set up suitable customizable complex experiments with the reference system MiSArch.

Stakeholders

The stakeholders for the tool that have an interest in its development are the **supervisors**, **researchers**, and **industry experts**.

Users

The performance and chaos engineering framework will be used by researchers and industry practitioners who want to conduct complex performance and chaos experiments (i) using a reference architecture or (ii) their system under test without overhead.

A.2.2 Constraints

R-C-001: Common and Open-Source Tooling

Description: The underlying tooling for the core functionalities, failure injection, load-generation, and reporting should be common, widely used tools.

Justification: Widely used tools have a higher acceptance rate in the research community and industry and, thus, are more likely to be adopted and used in the long term.

Origin: Researchers
History: Created on 26.03.2025

A.2.3 Functional Requirements

R-F-001: Steady State

Description: When running a performance experiment, the framework must ensure to warm up the architecture with load for some time before measuring. Further, before injecting failure the framework must measure the steady state of the system.

Justification: On start-up of the system, there might be side-effects that disappear after some warm-up time where the first load has been processed.

Origin: Researchers
Dysfunctional: + **Functional:** ++
History: Created on 26.03.2025

R-F-002: Realistic Workload Scenarios

Description: The framework shall provide predefined workload profiles that can be used to perform performance tests with MiSArch. These load profiles shall be as realistic as possible, e.g., by using public data on real-world load profiles. Further, at least the buy process, as described in the MiSArch architecture description, shall be modeled, including think-time.

Justification: Only with a realistic load scenario can the performance test results be taken as a blueprint.

Origin: Researchers, Supervisors
Dysfunctional: - **Functional:** ++
History: Created on 26.03.2025

R-F-003: Quality Attribute Workload Scenarios

Description: The framework shall provide predefined workload profiles that can be used to perform performance tests with MiSArch. These load profiles should contain concrete scenarios for testing specific non-functional attributes of the system under test, e.g., stress-testing, testing scalability, etc.

Justification: Performance and chaos engineers want to analyze specific quality attributes of the application. A fitting test setup to support them is required.

Origin: Researchers, Supervisors, Industry
Dysfunctional: - **Functional:** ++
History: Created on 26.03.2025

R-F-004: Configurable Load

Description: The framework shall provide the possibility to configure specific load parameters in the provided and newly created workload profiles.

Justification: Performance and chaos engineers want to analyze different requirements. A flexible test setup to support them is required.

Origin: Researchers, Supervisors, Industry
Dysfunctional: - - **Functional:** 0
History: Created on 26.03.2025

R-F-005: Scriptable and Versionable Test Setup

Description: The framework shall work with scriptable test setups. A test setup shall contain clear information in a machine-readable or machine-executable format that can be versioned and reproduced. The script shall be in a common language or format and versionable.

Justification: Scriptable test setups allow for reproducing the results and for conducting several experiments with minimal change, thus increasing usability.

Origin: Researchers, Supervisors
Dysfunctional: - - **Functional:** ++
History: Created on 26.03.2025

R-F-006: Test Goals in the Test Setup

Description: A Tester shall be able to configure certain test goals in the test setup, which the framework compares against. Those goals shall be measurable metrics, either from the user or system perspective.

Justification: A test always has an expected and an actual outcome, and they should be compared to each other.

Origin: Researchers, Supervisors
Dysfunctional: - - **Functional:** ++
History: Created on 26.03.2025

R-F-007: Configurable Observability Level

Description: The framework shall allow for different log and observability levels, which can be configured.

Justification: Different log levels allow for a tradeoff between observability data and performance impact.

Origin: Researchers
Dysfunctional: - **Functional:** +
History: Created on 26.03.2025

R-F-008: Configurable Failure States

Description: The framework shall allow for the configuration of failure states in the style of chaos engineering. The system must provide the ability to configure the following failure states for each service on each endpoint or topic: latency, network failure, and resource exhaustion. If time suffices, the system shall also allow for complete service failure and database crashes. Failure shall be applied on the infrastructure and container level first.

Justification: Detailed failure states at the application and endpoint level allow for realistic, fine-grained testing scenarios, which could also occur in real-world scenarios.

Origin: Researchers, Supervisors
Dysfunctional: - - **Functional:** +
History: Created on 26.03.2025

R-F-009: Dynamic and Static Failure States

Description: The framework shall allow for dynamic and static failure states. Static means the failure state does not change during the test execution, dynamic means the failure state can change during execution.

Justification: Dynamic and cascading failures are likely to occur in real-world failures and thus should be able to be tested. For simpler test scenarios, however, static failure does suffice as it is simpler.

Origin: Researchers, Supervisors
Dysfunctional: - - **Functional:** +
History: Created on 26.03.2025

R-F-010: Deterministic and Non-Deterministic Failure States

Description: The framework shall allow for deterministic and non-deterministic failure states.

Justification: Deterministic failure states are important for reproducible test results. Non-deterministic test results, however, allow for testing the general resilience of the system without knowing what will happen during the test upfront.

Origin: Researchers, Supervisors
Dysfunctional: 0 **Functional:** ++
History: Created on 26.03.2025

R-F-011: API and CLI integration

Description: The framework shall be able to consume the configuration files and start experiments via a CLI or an API integration.

Justification: A CLI interface allows for quick and simple test execution. An API allows for an extension of the framework by future researchers.

Origin: Researchers
Dysfunctional: - - **Functional:** +
History: Created on 26.03.2025

R-F-012: CI/CD integration

Description: The framework shall be able to consume the configuration files and start experiments inside a CI/CD pipeline.

Justification: A CI/CD pipeline integration allows for continuous testing in a DevOps environment, which helps in detecting failures in new releases.

Origin: Researchers, Industry
Dysfunctional: 0 **Functional:** +
History: Created on 06.04.2025

R-F-013: Experiment Configuration Generation

Description: The framework shall be able to support the creation of test configurations. Test configurations shall include work and load, failure states, goals of the experiment, and metrics to measure the goals. If applicable, the infrastructure can also be configured as code. The framework must be able to execute those configurations. The framework must offer an interface to receive required inputs, to create test configurations. Possible inputs are URLs to target with load, details on the desired work, and the desired failure states. The created test configurations should be standardized and contain this information. The test configurations must be machine-readable, human-readable, and versionable.

Justification: Generating executable configurations allows for quick prototyping of test-cases as a user and lowers the threshold to use the framework for technically inexperienced users.

Origin: Researchers, Industry, Supervisors
Dysfunctional: - **Functional:** ++
History: Created on 26.03.2025

R-F-014: Experiment Configuration Execution

Description: The framework shall be able to automatically execute a given test configuration. This includes executing the workload profile, combined with setting the correct failure states at the correct points in time and collecting and reporting the test results.

Justification: Executing the complex tests requires synchronization and is a repetitive task, which the user does not want to do.

Origin: Researchers, Industry, Supervisors
Dysfunctional: - - **Functional:** +
History: Created on 26.03.2025

R-F-015: Performance Metrics

Description: The framework shall monitor and export generic performance metrics for MiSArch. Those are response times, throughput, error rates, CPU and memory usage, the number of healthy pods, and application metrics, such as http requests. Those metrics should be measured on all components. Also, observability on all possible abstraction levels is required (e.g, container level, infrastructure level, application level).

Justification: Performance metrics allow understanding the performance of the system during the test.

Origin: Researchers, Supervisors, Industry
Dysfunctional: - - **Functional:** +
History: Created on 26.03.2025

R-F-016: Comparable Test Results

Description: The results of one test run should be in a comparable format, and the framework shall be capable of automatically comparing key metrics of two separate runs to spot differences.

Justification: To improve the understandability of the test results, a comparison feature can directly highlight the impact of a change.

Origin: Researchers, Industry
Dysfunctional: 0 **Functional:** ++
History: Created on 26.03.2025

R-F-017: Dashboards

Description: The results of one test run should be visualized in dashboards. There can be a differentiation between a user dashboard and a system dashboard, where the user dashboard covers the load driver perspective and the system dashboard monitors all components of the system.

Justification: To improve the understandability of the test results, a visualization helps massively.

Origin: Researchers, Industry
Dysfunctional: - - **Functional:** ++
History: Created on 26.03.2025

R-F-018: Automated Analysis using LLMs

Description: The framework shall be able to provide the collected metrics, traces, and logs to an LLM that extracts valuable information and key results and returns them to the tester in a concise format.

Justification: Analyzing the test results, especially for non-experts, is a time-consuming and complex task that can be supported by a generative AI, such as an LLM.

Origin: Supervisors, Researchers
Dysfunctional: + **Functional:** ++
History: Created on 26.03.2025

R-F-019: Excel-Export

Description: The framework shall provide an Excel export for the test results.

Justification: Excel is the backbone of most enterprises and can be understood by non-technical experts.

Origin: Industry
Dysfunctional: + **Functional:** +
History: Created on 26.03.2025

R-F-020: “Getting Started”-Example

Description: The framework shall provide a simple “Getting Started”-Example for a quick local setup that makes users understand how the tool works.

Justification: The conversion rate for a tool is much higher if an initial exploration setup is simple.

Origin: Researchers
Dysfunctional: - **Functional:** ++
History: Created on 26.03.2025

A.2.4 Non-Functional Requirements

Usability

R-U-001: Simple Usage

Description: The framework must be "easy-to-use". Running an experiment shall be straightforward. Some reproducible tests that can be used out-of-the-box shall be provided. A mini version of the framework and application deployment shall be there to help with quick onboarding.

Justification: To reach a broad user base and high adoption of the tool, it must be easy to quick-start and understand, otherwise, users will lose interest and choose another tool.

Origin: Researchers
History: Created on 26.03.2025

R-U-002: Self-Explainability

Description: The framework shall provide good explanation texts and descriptions that clearly show what configurations in the test will likely have what impact on the test and the system under test.

Justification: The framework should enable users to create test setups that provide results meeting their needs, which is why good explanations are required.

Origin: Researchers
History: Created on 26.03.2025

R-U-003: Reproducible Results

Description: The performance tests executed with the framework must be reproducible by other stakeholders with similar results.

Justification: Academic adoption is only possible if the results are valid and can be reproduced by other researchers.

Origin: Researchers
History: Created on 26.03.2025

R-U-004: Reproducible Framework

Description: The framework must be able to be applied in other contexts, providing a clear structure that enables users to conduct performance tests in other domains.

Justification: If the framework only works with the MiSArch reference architecture, only a small number of users can profit from the functionality.

Origin: Researchers
History: Created on 26.03.2025

Performance

R-P-001: No Performance Overheads

Description: The framework must ensure that there is no additional performance implication by the framework itself on the system under test. For example, load generation and observability could both have performance impacts, which must be kept to a minimum.

Justification: The performance results might be affected by the framework in a bad design. This must not happen.

Origin: Developer, Supervisors
History: Created on 26.03.2025

Operational and Environmental

R-OE-001: Documentation

Description: The framework must be well documented. The software architecture of the framework must be well documented. The interfaces of the framework must be clearly and comprehensively documented. Design decisions must be documented using Architectural Design Records (ADRs)[1][2][3].

Justification: Clear documentation and common formats are indispensable to ensure that others can reproduce the results and use the software.

Origin: Developer, Supervisors
History: Created on 26.03.2025

R-OE-002: Storage and Provisioning

Description: The source code of the framework must be open-sourced in GitHub using the MIT License. The source code of the framework, as well as other related software artifacts, must be publicly accessible.

Justification: Only with open-source software can the results be used by researchers and industry.

Origin: Supervisors, Developer, Researchers
History: Created on 26.03.2025

[1] https://adr.github.io
[2] https://github.com/joelparkerhenderson/architecture-decision-record
[3] https://adr.github.io/madr/