BACHELOR SOFTWARE ENGINEERING PROJECT

User Requirements Document

Team Waterfowl

June 16, 2021

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Eindhoven University of Technology

2IPE0 Software Engineering Project
QUARTILE IV - 2020-2021

User Requirements Document

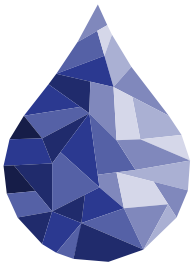
Team Waterfowl

Authors:
Adrien Castella *(1280880)*
Adrian Cucoş *(1327860)*
Cosmin Manea *(1298542)*
Noah van der Meer *(1116703)*
Lulof Pirée *(1363638)*
Mihail Ţifrea *(1317415)*
Tristan Trouwen *(1322591)*
Tudor Voicu *(1339532)*
Adrian Vrămuleț *(1284487)*
Yuqing Zeng *(1284835)*

Supervisor:
Gerard Zwaan, univ. lecturer
Customer:
Jim Portegies, asst. prof.

Version 0.7

June 16, 2021



Abstract

This document, the User Requirement Document (URD), describes the requirements for a robust Waterproof. Waterproof is a student-focused environment for writing mathematical proofs with a focus on real analysis. The current project aims to improve the existing system in three key areas: introducing an installer for the Windows platform (i), designing a robust AST-Parser (ii), and rewriting the tactics library to $L_{\text{tac}}2$ (iii).

This document complies with the ESA software standards (cf. [10]).

Contents

Document Status Sheet

Title	User Requirements Document
Authors	Adrien Castella, Adrian Cucoş, Cosmin Manea, Noah van der Meer, Lulof Pirée, Mihail Ţifrea, Tristan Trouwen, Tudor Voicu, Adrian Vrămuleţ, Yuqing Zeng
Version	0.7
Status	Final
Creation Date	April 25, 2021
Last Modified	June 16, 2021

Document History log

April 25th, 2021: Initial release.

April 28th, 2021: First revision based on client feedback.

May 2nd, 2021: Second revision based on client and supervisor feedback.

May 3rd, 2021: Third revision based on client feedback.

May 4th, 2021: Fourth revision based on client and supervisor feedback.

May 5th, 2021: Final revision based on customer feedback.

June 10th, 2021: Additional revision based on project development.

1 Introduction

1.1 Purpose

The User Requirements Document contains the general description, user requirements and use cases for the work done on improving the application Waterproof (cf. [7]). They are a collaboration between the customer, Jim Portegies, and the software development team.

The Requirements will be implemented according to their priorities using the MoSCoW model, cf. [2]. The **must have** requirements will have been implemented by the end of the project. All other requirements will be implemented based on their priority, should the resources allow it.

1.2 Scope

The project of improving Waterproof is being developed by a group of Computer Science students from Eindhoven University of Technology, as part of their Software Engineering Project. The goal of the project is to improve the user experience of the mentioned application, as well as to facilitate updating and performing maintenance on it.

Currently, since Waterproof is built on top of an Opam system, maintenance and installation are complex tasks to perform. The goal of this project is to improve the user experience and maintainability of Waterproof, a wrapper for Coq. In achieving this goal, an easy installer, whose purpose is to set up the underlying ecosystem, as well as an updater have to be created. Moreover, improvements to the Waterproof application have to be made. The most important ones are about the UI and its dependencies. To this end, syntax highlighting, auto-completion and a custom tactics library have to be implemented. A detailed description of each of those tasks can be found in Section 2.

1.3 List of definition

1.3.1 Definitions

Term	Definition
Automation	A feature of Coq to automatically synthesize simple pieces of a proof.
Automatic solving	Coq commands that automatically advance the proof state.
Coq	A formal proof management system that allows for expressing mathematical expressions and assertions (cf. [3]).
Coq-SerAPI	A library for machine-to-machine interaction with the Coq proof assistant.
Command	An instruction that can be used to modify the state of a Coq document, for instance by declaring a new object, or to get information about the current state.
Cygwin	A program allowing the user to run native Linux applications on Windows.
Digital signature	Cryptographic security primitive that can be used to verify the authenticity of data and software packages
Electron	A framework for creating cross-platform desktop applications using web technologies (JavaScript, HTML, and CSS).
Executable file	A program that can be run in the OS without explicitly needing to load it with another program.
Hash	The result of applying a cryptographic one-way ("hash") function over an input. They are often used in practice to check that a file was not altered.
Lemma	Proven statement used as a stepping stone for proving a larger result.
$L_{\text{tac}}1$	A tactic language for Coq.
$L_{\text{tac}}2$	A newer tactic language for Coq (successor of $L_{\text{tac}}1$).
Mechanization	Automating a process for more efficiently accomplishing a given task.
OCaml	A general-purpose typed programming language. OCaml is designed to enhance expressiveness and safety (cf. [9]).
Opam system	A source-based package manager for distributing OCaml programs and tools.
Portable executable	An application that embeds all its necessary dependencies and can be executed from a portable storage medium. Not related to the Windows *.pe format.
Proof assistants	"computer programs" specifically designed for mechanizing rigorous mathematical proofs on a computer.
Proof state	Keeps track of all statements used so far which are true, as well as the conclusion statement that needs to be proven and the remaining statements to prove.
Git	A version control system, a tool used to manage and track changes of software projects.
Pull request	A method to propose changes to the Git repository of a code project.

Querying a lemma	Coq-specific language, meaning returning the information of the respective lemma (such as parameters etc.).
Serialization	The process of converting an object into a stream of bytes to transmit it or store it in memory, a database, or a file.
Sentence	Instructions used by Coq that can either refer to commands or tactics.
Software dependency	A software component necessary for the operations of a different program.
Software package	A collection of applications or code modules that work together to meet various goals and objectives.
Software repository	A storage location for software packages.
Tactic	Mathematical statement that advances the proof state.
Tactic language	A set of tactics that together form the complete functionality of Coq.
Transport Layer Security	A cryptographic protocol for secure communication over a computer network.
S-expression	A symbolic notation for representing a (nested) tree-structured data.
CoqAST	A type of data structure returned by SerAPI which represents the code written by the user.
Syntax highlighting	The feature displays text, especially source code, in different colours and fonts according to the category of terms.
User	Person that uses the application.
Wrapper	Application built around another program in order to provide a different interface. For example, Waterproof is a wrapper around Coq that simplifies the proving language.
`.v` files	Coq source code files (also known as vernacular files).
`.wpn` / `.wpe` files	Files used by Waterproof. The extensions stand for "Waterproof notebook" and "Waterproof exercise sheet" respectively.

1.3.2 Abbreviations

API	Application Programming Interface
AST	Abstract Syntax tree
INRIA	The French National Institute for Research in Digital Science and Technology
CPU	Central processing unit
JS	JavaScript
GB	Gigabytes
Opam	OCaml package manager
OS	Operating System
PDF	Portable Format Document
RAM	Random-access memory
SEP	Software Engineering Project
UI	User Interface

1.4 List of References

References

[1] 14:00-17:00. *ISO/IEC 25010:2011*. en. URL: https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/03/57/35733.html (visited on 05/02/2021).

[2] DSDM Consortium. *The DSDM Agile Project Framework for Scrum*. 2012.

[3] Thierry Coquand, Gérard Huet, and Christine Paulin. *The Coq Proof Assistant*. Accessed: 2021-04-27. URL: https://coq.inria.fr/.

[4] *Electron | Build cross-platform desktop apps with JavaScript, HTML, and CSS*. en-US. URL: https://www.electronjs.org/ (visited on 05/04/2021).

[5] H. Geuvers. *Proof assistants: History, ideas and future*. 2009.

[6] D. Goodin. *Backdoor built in to widely used tax app seeded last week's NotPetya outbreak*. https://arstechnica.com/information-technology/2017/07/heavily-armed-police-raid-company-that-seeded-last-weeks-notpetya-outbreak/. Accessed: 2021-04-27. 2017.

[7] *impermeable/waterproof*. original-date: 2019-09-20T07:42:09Z. Mar. 2021. URL: https://github.com/impermeable/waterproof (visited on 05/02/2021).

[8] *Ltac2 — Coq 8.13.2 documentation*. URL: https://coq.inria.fr/refman/proof-engine/ltac2.html#compatibility-layer-with-ltac1 (visited on 05/02/2021).

[9] *OCaml*. Accessed: 2021-04-27. URL: https://ocaml.org/index.html.

[10] ESA Board for Software Standardisation and Control (BSSC). *ESA Software Engineering Standards: Issue 2*. 1991.

1.5 Overview

The remainder of the document is organised as follows. As already mentioned, Chapter 2 contains a more general description of what this project aims to achieve. The main subjects that shall be discussed are the product perspective, general capabilities, the constraints of the system, the user characteristics, the environment description, and the assumptions and dependencies. Chapter 3, the formal user capabilities and constraints requirements will be listed. Those consist of requirements regarding the project, as well as their priority level. Finally, a list of all use cases relevant to the product is presented in Appendix A.

2 General description

Proof assistants (cf. [5]) are "computer programs" specifically designed for mechanising rigorous mathematical proofs on a computer. A particular mathematical theory can be constructed, in which the usual logical reasoning can be applied. The proof assistant tracks what remains to be proven, verifies correctness of the proof and may automatically finish parts of the proof.

One such popular proof assistant is Coq. Developed primarily at INRIA, Coq allows for interactively creating and validating formal proofs. The proof assistant provides its own extensible scripting language for elaborating proofs in the form of vernacular (`.v`) files. Each vernacular contains a series of instructions (called 'Sentences' in the Coq terminology), which can either be commands or tactics.

2.1 Product perspective

Waterproof is a wrapper around the proof assistant Coq (i.e. it facilitates using the underlying proof assistant), designed to help students at the Eindhoven University of Technology in their learning process of analysis, which, for now, is done throughout the course 2WA30 Analysis 1.

Within these courses, students need to understand and come up with their own rigorous proofs, check their correctness, and find their mistakes. They should ideally achieve this before receiving their final grade for an assignment and without being provided with the solution.

The Waterproof application aims to fulfil this need. It offers an editor interface in which students can write formal proofs. The program provides the users with feedback in case of a mistake. Furthermore, Waterproof shows what remains to be proven while the student is working. Finally, teachers can use Waterproof to quickly generate exercise files, which can be shared with students.

Waterproof enables the user to write formal statements (according to a strict syntax) inside text boxes ('blocks'). These statements can be run to obtain feedback. In the case of the proof being complete, it can be empty. Otherwise, as mentioned before, it can contain the remaining statements to prove, but the feedback is not necessarily limited to that. These particular blocks of statements can be so-called *tactics* or can contain hints or comments.

At the moment, the complete installation process of Waterproof (including its back-end dependencies) is unreliable and complex and said dependencies also need to be constantly updated. Many dependencies need to be installed separately (such as Opam and Coq-SerAPI) through a tedious process that can be difficult for some students. Furthermore, users on all operating systems often encounter errors during installation caused either by the installers, the operating system, or by improper steps taken during installation.

Secondly, it is difficult for new Waterproof users to learn the functionality and syntax of the functions used within this software. It would thus be helpful to aid them in their learning process using syntax highlighting.

Finally, the tactics library does not leave many options for customisation, and it is difficult to maintain.

2.2 General capabilities

On a high level, this project should add the following capabilities to Waterproof:

2.2.1 Better Waterproof installer

To install Waterproof, the user needs to install several of its dependencies *manually*: Coq SerAPI, Opam, and Cygwin. Cygwin is designed to set up an environment similar to that of Linux for the application to run in, if this is not already available. Coq-SerAPI is installed using Opam and is used by Waterproof to communicate with Coq (this process is called *serialization*). Waterproof is currently installed with a separate installer.

The project's goal is to create a new installer that combines all of the above tasks. This way, all dependencies of Waterproof will be installed. The Waterproof application will still have it's own installer. The focus will be on streamlining this installation process and

making it simple to execute for as many people as possible with a great success rate. However, if the installer still encounters issues, then it should report this in an error log.

2.2.2 Waterproof Updater

The dependencies mentioned in Chapter 2.2.1, and the Waterproof software remain in ongoing development. Hence, to access the newest features, a user would need to update Waterproof regularly and its dependencies manually.

An updater shall be integrated into Waterproof, allowing the user to update all software packages from the application easily. In addition, the user will be notified when new updates are available. This notification comes with an opportunity to install the updates immediately or to postpone them.

2.2.3 AST parser

The current Waterproof software requires users to input mathematical proofs in the form of vernacular Sentences. These commands are sent to SerAPI, which translates them to ASTs in the form of an S-expression which Coq can interpret. Coq processes this expression and then sends back a new AST containing its feedback for Waterproof. This AST feedback is sent back to Waterproof through SerAPI. However, it is not properly parsed and remains in the form of an S-expression, which is not readable to the end users. Currently, only simple information is extracted from the returned S-expression, but no proper representation of the AST is performed.

Waterproof should include a parser that converts the S-expression into an internal datastructure in order to enable the implementation of new features which improve readability. Syntax highlighting is one of these features.

While every response coming from SerAPI is a valid S-expression, only a subset of S-expressions concern the usage of the editor, namely the ones encoding a CoqAST datastructure.

2.2.4 Auto-Completion

Users can query keywords based on documented keywords in Waterproof as they type. The program will automatically display options for the complete keywords. An example of these keywords would be a lemma known to Waterproof.

2.2.5 New tactics library

Waterproof uses its own library of custom tactics to simplify the syntax. However, the old library depends on the outdated tactic language called $L_{\text{tac}}1$, which lacks some customisation options. This project will create a new tactics library implemented using the newer tactics language $L_{\text{tac}}2$.

The new library should offer at least the same tactics, but it may also add new features if time allows it. This change does not pose compatibility issues since, according to the Coq documentation, $L_{\text{tac}}2$ is backwards compatible with $L_{\text{tac}}1$ (cf. [8]). Thus, the migration process can be done incrementally.

One of the main resulting features of this transition will be the improved customisability offered by $L_{\text{tac}}2$. A first example is that using $L_{\text{tac}}2$ will make it possible to configure at runtime which hint-database is loaded. A second example is that parts of the automation

would become configurable; for instance, the number of steps it can look ahead and the lemmas it can use. More concretely, the first example from above avoids the oversimplification of the exercises - some lemmas should be proven explicitly instead of taken for granted and used by default within automation. The second one makes it possible to choose a set of lemmas that corresponds precisely to the lecture material.

Optionally, the new library may also introduce new tactics. These new tactics should be designed in coordination with the client at a later stage, but only if time permits.

2.3 General constraints

The additions made to Waterproof ought to satisfy several constraints. A high-level overview is given below. Some of these constraints regard implementation details, performance requirements, maintainability, backward compatibility etc.

2.3.1 General

The newly delivered version of Waterproof should still support all built-in syntax of Coq (its standard library) within code blocks. The client requires this feature for technical reasons.

2.3.2 Installer and updater

The guiding principles for the creation of the installer are simplicity of use and robustness, such that virtually all students can correctly install Waterproof on the Windows OS. These come as a response to the friction caused by the current installation process, which hinders the adoption of Waterproof and excludes less tech-savvy users from using it.

Recently, threats have used these updater mechanisms to disperse malware (cf. [6]). Thus, the updater should be designed so that it prevents malicious parties from exploiting the software as a malware distribution platform. It should incorporate, at the very least, basic security measures, such as checking the digital signature of the executables and the hashes of the known files.

Moreover, the updater should not modify the software's settings of each particular user, but preserve them as much as possible. This way, the user can continue to work with the Waterproof software without noticing any disruptions.

Finally, due to the constraints imposed by the customer, the installer and the updater will only be available for the Windows 10 OS, targeting version 2004 or later.

2.3.3 AST

The AST parser should output interpretable and flexible code. This means that the parsed information should be usable for various applications, including syntax highlighting and any other requests made by the client.

Additionally, the AST's parsing algorithm should have a linear asymptotic time complexity. Fast parsing is required to make real-time use of the AST, such as syntax highlighting, possible.

Finally, the resulting internal representation of the AST should be modular and extendable (cf.[1]). As this will be used by the Waterproof client, it should provide an easy to use API, with useful functionality. For example, the objects returned by the AST-Parser

should offer a `flatten` procedure which, given the tree, returns a summary in the form of a list datastrucutre.

2.3.4 Tactics library

The new tactics library should be a direct replacement of the previous library. Therefore, it must provide at least all the functionality of the previous library. Furthermore, it must be written in the $L_{\text{tac}}2$ tactics language, at the client's request. It should be a well-structured modular collection of separate files, bundled as an Opam package. Improved maintainability should be one of the main improvements with respect to the previous library. This means that individual tactics should be decoupled where possible. Finally, hard-coding and duplicate code should be avoided.

2.4 User characteristics

The primary target users of the Waterproof application are university students. However, Waterproof is also available to the general public.

The notion of user hierarchy or user role does not exist within Waterproof: every user can perform the same operations.

The client assumes that most students will use the Windows 10 OS, and have little experience with manually installing dependencies. A foolproof and straightforward installation is in the interest of students.

It is also unlikely students start using Waterproof outside of the scope of their courses (where it is supported). Only within a course templates and hints are provided. While customisation of the setup might not be a priority due to the high user (students, teachers, developers) and configuration heterogeneity, options to control certain aspects of the installation process might be required.

2.5 Environment description

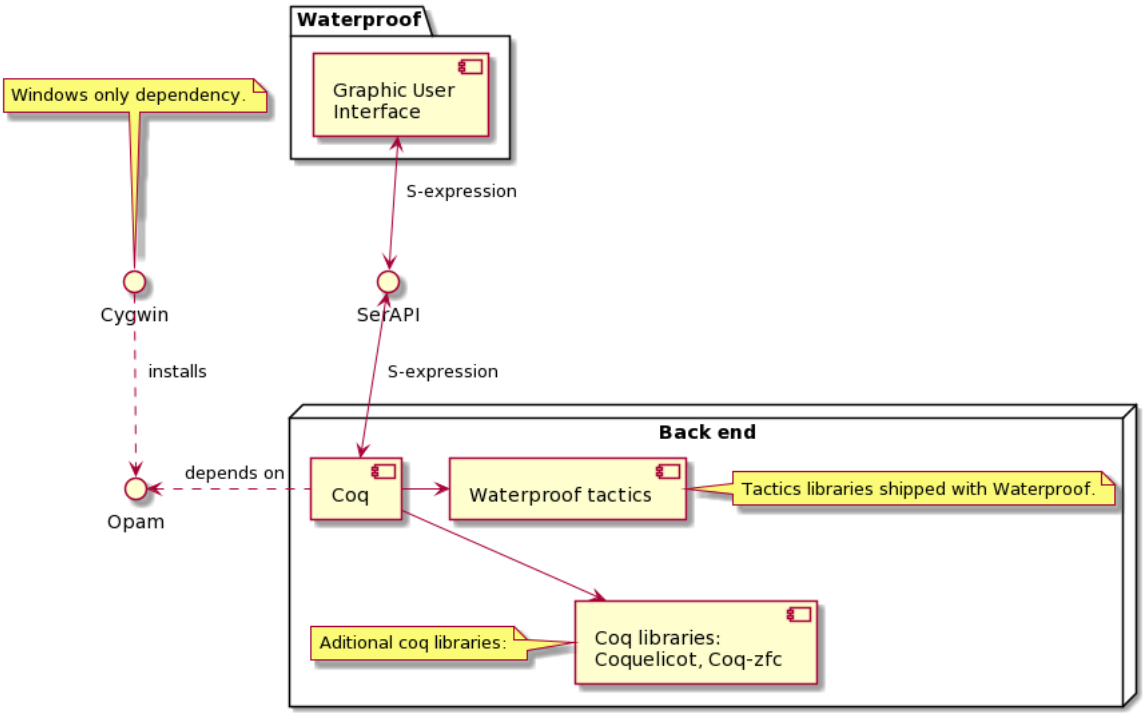


Figure 1: Environment diagram

Waterproof is a native application based on `Electron` (cf. [4]). The only changes made to Waterproof that will affect the environment structure are the transition to $L_{\text{tac}}2$, together with the modifications made to the tactic library. The current version of Waterproof contains an embedded library of tactics in the form of `.v` files. This will change due to the need for modularization since, as of now, a change in the tactics library of Waterproof requires modifications to the source code.

A more flexible environment that allows changes to the current tactic library is shown in Figure 1. Waterproof tactics will be in direct relation to Coq and will be compiled externally of Waterproof. Additional Coq libraries will also constitute an improvement to this environment since this will facilitate an extension to the functionality of Coq.

The Waterproof client communicates with the Coq back-end via SerAPI by means of S-expressions. The GUI of Waterproof serves the main interactions with the user. Libraries and tactics packages will be integrated within the Waterproof back-end so that Coq has direct access to them. Furthermore, Opam depends on Cygwin for its installation only on the Windows OS. Opam is a package manager that contains the OCaml package, which is required for running Coq. The provided diagram showcases the communication between the front-end and the back-end.

2.6 Assumptions and dependencies

This section describes the assumptions and dependencies upon which the project's requirements section are based (in Chapter 3).

2.6.1 Assumptions

The requirements on the Waterproof extension should be met under the following assumptions only:

1. The user uses the application as intended.
2. The user is using a functional up-to-date version of a supported OS.
3. The user possesses professional-level proficiency in English.
4. The features present in the Waterproof version prior to this SEP project (version 0.5.3) are working as intended and considered stable.
5. All used dependencies, such as OCaml, Opam, Coq, SerAPI and Cygwin, are functioning correctly.
6. The user is using a machine with at least 8GB of RAM and a CPU with at least 2 cores.

2.6.2 Dependencies

1. Waterproof, version 0.5.3 (at least)
2. Cygwin, version 3.2.0 (at least)
3. SerAPI, version 8.2 (at least)
4. Opam, version 2.0 (at least)
5. OCaml, version 4.0 (at least)
6. Coq, version 8.9.0 (at least)

3 Specific requirements

In this section, the description of the requirements for the items mentioned in Section 2 is presented. Each requirement has a priority to indicate its importance for the end product. This has been achieved via the MoSCoW model, which is a methodology that assigns to each requirement a priority using the following labels:

Priority	Description
Must have	The product must have this requirement to be called complete.
Should have	These requirements are not vital to the product, but are desirable to have.
Could have	These requirements would be nice to have, but can be left out.
Won't have	These requirements will not be added to the final product within the scope of this project.

3.1 Capability requirements

3.1.1 Installer

ID	Requirement	Priority
CAR-INST-1	The installer allows the user to choose which of the supported Coq libraries to install.	Must have
CAR-INST-2	The installer shall allow the developers to specify, in a configuration file, which of the supported Coq libraries are installed by default.	Must have
CAR-INST-3	The installer shall have options to include custom Coq libraries to the supported libraries.	Must have
CAR-INST-4	The supported Coq libraries shall include Coq-SerAPI.	Must have
CAR-INST-5	The supported Coq libraries shall include Coquelicot.	Must have
CAR-INST-6	The supported Coq libraries shall include Coq-zfc.	Should have
CAR-INST-7	The installer shall be compatible with the Waterproof front-end installed by the Waterproof installer.	Must have
CAR-INST-8	An uninstaller is installed which can uninstall Coq and its dependencies.	Must have
CAR-INST-9	During the uninstallation procedure, the user shall have the option on selecting which packages to uninstall.	Should have
CAR-INST-10	The installer shall show how much space is needed for the installation.	Could have
CAR-INST-11	The user can select in the installer whether it should associate `.wpn` and `.wpe` files with Waterproof.	Could have
CAR-INST-12	The installer shall collect system information about the user and transmits it to the software developers.	Won't have
CAR-INST-13	The installer can re-install software packages if these are already installed.	Won't have

3.1.2 Updater

ID	Requirement	Priority
CAR-UPDT-1	The updater shall be part of the Waterproof application.	Could have
CAR-UPDT-2	The Waterproof application shall have the option of turning on/off the automatic check for updates when the application is started.	Should have
CAR-UPDT-3	When the Waterproof application is started, the application shall check for updates if this option is enabled.	Should have
CAR-UPDT-4	When the user presses the "Check for updates" button, the Waterproof application shall check whether an update is available.	Could have
CAR-UPDT-5	When the Waterproof application is aware of an update, the application shall notify the user of this update.	Should have
CAR-UPDT-6	The updater shall have the option of updating Coqelicot.	Should have
CAR-UPDT-7	The updater shall update packages using their respective package managers (for instance, update Opam packages via Opam etc.)	Should have
CAR-UPDT-8	The user shall have the option of postponing updates.	Should have
CAR-UPDT-9	The user can update Waterproof with a single button click.	Should have

3.1.3 Abstract Syntax Tree Parser

ID	Requirement	Priority
CAR-AST-1	The AST-Parser shall create an internal data structure for the CoqASTs returned by SerAPI in Waterproof.	Must have
CAR-AST-2	The data-structure returned by the AST-Parser from a valid S-expression provides the Waterproof client with the necessary and sufficient information to perform syntax highlighting, in the form of a list of identifiers and their position in the Waterproof editor.	Must have
CAR-AST-3	The users shall have the option of auto-completing the names of lemmas by pressing a predefined key.	Should have
CAR-AST-4	The users shall have the option of querying lemmas based on documented lemmas in Waterproof as they type by using a predefined macro.	Should have
CAR-AST-5	The users shall have the option of auto-completing the template tactics by pressing a predefined key.	Should have
CAR-AST-6	The AST-parser has a feature to real-time translate the user input (i.e. while still typing) to AST and match against the AST syntax of Coq to mirror the syntax tree of Coq on Waterproof's side.	Won't have
CAR-AST-7	In case of errors, the AST-Parser should return human-readable errors.	Should have
CAR-AST-8	The GUI shall contain buttons that when pressed insert a specific well-formed template sentence at the cursor location.	Could have

3.1.4 Tactics library

ID	Requirement	Priority
CAR-TL-1	The new tactics library implements at least 80% of the tactics from the old Waterproof tactics library.	Must have
CAR-TL-2	The new tactics library shall implement the `Waterprove` automation tactic from the old tactics library.	Must have
CAR-TL-3	The new tactics library is bundled as an Opam package.	Must have
CAR-TL-4	A pull request to the Coq Git repository is made, containing the new tactics library.	Must have
CAR-TL-5	The new tactics library is released to the Coq Git repository.	Should have
CAR-TL-6	The hints database used by the new tactics library is configurable.	Must have
CAR-TL-7	The set of lemmas used by the automation in the tactics library is configurable.	Should have
CAR-TL-8	The amount of steps that the the automation in the tactics library is allowed to take is configurable.	Should have
CAR-TL-9	The tactics library shall encapsulate the error messages of the Coq software in more user-friendly and detailed messages.	Should have
CAR-TL-10	The tactics library shall contain commands that give feedback to the user.	Should have
CAR-TL-11	The tactics library shall introduce new tactics to Waterproof.	Could have

3.2 Constraint requirements

3.2.1 General

ID	Requirement	Priority
COR-GEN-1	The software artefacts produced as part of the SEP project shall be merged back with the original GitHub repository (cf. [7]).	Must have
COR-GEN-2	The updated version of Waterproof shall remain compatible with the Coq standard library.	Must have
COR-GEN-3	The software artefacts produced adhere to the ISO 25010 maintainability standard (cf. [1]).	Could have

3.2.2 Installer

ID	Requirement	Priority
COR-INST-1	The installer shall be a single executable file of size under 500MB.	Must have
COR-INST-2	The installer shall support Microsoft Windows 10, version 2004 or higher.	Must have
COR-INST-3	The user can install the software with at most 5 user interactions with the installer application.	Must have
COR-INST-4	The user can install the software with at most 3 user interactions with the installer application.	Could have
COR-INST-5	The installer shall work independently of the versions of the relevant software packages.	Should have
COR-INST-6	The installer shall comply with the license requirements of all the packages it installs.	Must have
COR-INST-7	The installer executable contains a digital signature of itself and the Waterproof software.	Could have
COR-INST-8	The installer should be a single executable file with a size under 50MB.	Could have
COR-INST-9	The installer is available in English.	Must have
COR-INST-10	The installer is available in Dutch.	Could have
COR-INST-11	The installer shall only perform manually reproducible operations (for instance, cloning a Git repository).	Should have
COR-INST-12	The installer shall catch raised exceptions and output appropriate feedback to an error log file placed on the user desktop.	Could have
COR-INST-13	The error logs produced by the installer shall be in plain text.	Should have

3.2.3 Updater

ID	Requirement	Priority
COR-UPDT-1	If the updater downloads the software updates, the updater shall download the software updates over a connection secured by Transport Layer Security version 1.2 or higher.	Must have
COR-UPDT-2	If the updater downloads the software updates, the updater shall verify the authenticity of updates.	Must have
COR-UPDT-3	When the updater updates the software, it shall perform only the minimal changes required in the user settings and user files.	Should have

3.2.4 Abstract Syntax Tree Parser

ID	Requirement	Priority
COR-AST-1	For an S-expression of 10000 characters or less, the AST-Parser can parse and output the AST data structure within 0.4 seconds.	Must have
COR-AST-2	For an S-expression of 10000 characters or less, the AST-Parser can parse and output the AST data structure within 0.1 seconds.	Should have
COR-AST-3	With n being the length of the AST in S-expression form, the AST-Parser creates an internal data structure from the AST within $O(n)$ – linear time.	Could have
COR-AST-4	Flatting the AST should match the sentence written in the Waterproof editor, ignoring the white spacing.	Should have
COR-AST-5	Each complete lemma query in Waterproof shall take less than 0.7 second to execute.	Should have

3.2.5 Tactics library

ID	Requirement	Priority
COR-TL-1	The tactics library shall use the $L_{\text{tac}}2$ tactic language where feasible.	Must have
COR-TL-2	The tactics library shall be compatible with the existing tactics languages in the sense that it, at least, offers the same functionality as the old tactics library (from a user's point of view), up to renaming of tactics.	Must have
COR-TL-3	The tactics library is a modular collection of `.v` files.	Must have
COR-TL-4	In the pull request to the Coq Git repository, the name of the new tactics library shall be "Waterproof".	Must have
COR-TL-5	The new tactics library shall be released to the Coq Git repository under the name "Waterproof".	Should have
COR-TL-6	The tactics library shall not contain dependencies between different tactics where realistically avoidable.	Should have
COR-TL-7	Parametric automatic solving in the tactics library is implemented only once.	Should have
COR-TL-8	The tactics library shall compile within 3 minutes.	Should have
COR-TL-9	The tactics library shall contain a template filling of set tactic phrases (to be defined by the customer).	Should have
COR-TL-10	The tactics library shall stimulate the students to use good coding/proving practices, such as specifying types of variables that they introduce, in such a way that the proof would seem full and complete from a mathematical point of view.	Could have
COR-TL-11	The tactics library should allow for synonyms in tactics. The actual phrasing decisions are left to the customer.	Could have

A Use cases

A.1 Syntax highlighting

Goal:	Highlighted S-expression
Pre-condition:	The Sentence entered by the user will elicit a valid response from Coq.
Post-condition:	The Sentence is syntax-highlighted in the Waterproof editor.
Actors:	The user.
Summary:	The user sends a Sentence to the system and the system parses and highlights the AST(in S-expression format) returned from Coq.
Priority:	Should have
Requirements:	CAR-AST-2, COR-AST-4
Actions taken by the actor and system :	(1) The user writes and sends a Sentence through Waterproof. (2) The system receives an AST in the form of an S-expression from SerAPI. (3) The system parses the AST and interprets the information to use for syntax highlighting. (4) Waterproof uses the information retrieved from the AST-Parser to highlight the current sentence.

A.2 Suggestion bar for template tactics

Goal:	Display the suggestion bar with the similar template tactics.
Pre-condition:	The user hold their mouse cursor over the selected keyword inside a code-block.
Post-condition:	A suggestion bar with similar template tactics is displayed beneath the keyword.
Actors:	The user.
Summary:	When a user presses the specified predefined key, the system queries a list of template tactics for similar tactics for the user and displays them as a suggestion bar.
Priority:	Should have.
Requirements:	CAR-AST-5
Actions taken by the actor and system :	(1) The user presses the predefined key. (2) The system interprets the text-box and queries for similar template tactics. (3) The system returns the list of similar template tactics and displays it as a suggestion bar for the user.

A.3 Suggestion bar for names of the lemmas

Goal:	Display the suggestion bar with the possible names of lemmas.
Pre-condition:	The user hold their mouse cursor over the lemma keyword inside a code-block.
Post-condition:	A suggestion bar with possible names of lemmas is displayed beneath the keyword.
Actors:	The user.
Summary:	When a user presses the specified predefined key, the system queries a list of possible names of lemmas for the user and displays them as a suggestion bar.
Priority:	Should have.
Requirements:	CAR-AST-3
Actions taken by the actor and system :	(1) The user presses the predefined key. (2) The system interprets the text-box and queries for possible names of lemmas. (3) The system returns the list of possible names of lemmas and displays it as a suggestion bar for the user.

A.4 Auto-completion

Goal:	Complete the keyword chosen by the user.
Pre-condition:	There is a suggestion bar for a selected incomplete keyword (see use cases A.3 and A.2) displayed.
Post-condition:	The keyword is completed.
Actors:	The user.
Summary:	The user selects a keyword in the suggestion bar and the system completes the partial text with the chosen keyword.
Priority:	Should have.
Requirements:	CAR-AST-3, CAR-AST-5
Actions taken by the actor and system :	(1) The user selects a keyword from the list in the suggestion bar. (2) The system completes the keyword in the text-box with the chosen option(s).

A.5 Querying a lemma

Goal:	The user queries a lemma.
Pre-condition:	The user has finished typing a known (i.e. previously proven) lemma and their mouse cursor hovers the name of the lemma.
Post-condition:	The user receives the information of the respective lemma.
Actors:	The user.
Summary:	The user queries a lemma and receives its particular information.
Priority:	Should have.
Requirements:	CAR-AST-4, COR-AST-5
Actions taken by the actor and system :	(1) The user presses the predefined key for lemma querying. (2) The system displays, in a separate window, the information possessed by the lemma (such as parameters etc.)

A.6 AST-parser error

Goal:	The developer receives an AST-parser error.
Pre-condition:	The developer called the AST-parser to parse a malformed S-expression.
Post-condition:	The AST-parser error is returned to the developer.
Actors:	The developer
Summary:	The AST-parser returns an error to the developer in case of poor input.
Priority:	Must have.
Requirements:	CAR-AST-7
Actions taken by the actor and system :	(1) The AST-parser tries to parse the malformed S-expression. (2) An error occurs during this parsing procedure. (3) The system returns a human-readable error message to the developer.

A.7 Install Waterproof dependencies

Goal:	Install the selected software packages.
Pre-condition:	-
Post-condition:	The selected software packages are installed on the user's machine.
Actors:	The user
Summary:	A user installs Waterproof dependencies.
Priority:	Must have.
Requirements:	CAR-INST-1, CAR-INST-4, CAR-INST-5, CAR-INST-6, CAR-INST-7
Actions taken by the actor and system :	(1) The user runs the installer. (2) The installer responds by displaying the user interface. (3) If the user wants a custom package list installed, then the use case A.8 starts. Otherwise, the configured default packages will be selected. (4) The user starts the installation procedure. (5) The system notifies the user about how much space is needed for the installation. (6) The system starts installing the selected dependencies. (7) The installer finishes and notifies the user.

A.8 Choose Waterproof dependencies

Goal:	Specify a selection of preferred packages.
Pre-condition:	The installer is running and at the package selection screen.
Post-condition:	The installer will install the selected packages upon its continuation.
Actors:	The user
Summary:	A user specifies a selection of packages which need to be installed.
Priority:	Must have.
Requirements:	CAR-INST-1, CAR-INST-4, CAR-INST-5, CAR-INST-6
Actions taken by the actor and system :	(1) The user specifies which software packages should be installed by clicking a checkbox in front of the package name.

A.9 Configure installer settings

Goal:	Configure default packages.
Pre-condition:	The developer can compile the installer.
Post-condition:	The packages selected by default in the installer are set by the developer.
Actors:	The developer
Summary:	A developer specifies the packages to be made available through the installer.
Priority:	Should have.
Requirements:	CAR-INST-2
Actions taken by the actor and system :	(1) The developer opens the configuration file. (2) The developer specifies the packages to be made available through the installer. (3) The developer recompiles the installer.

A.10 Creating the installer

Goal:	Compiling an installer.
Pre-condition:	The developer has has set up the additional configuration file as specified in A.9.
Post-condition:	An installer is created, and the packages included within the installer are available as configured by the developer.
Actors:	The developer
Summary:	A developer builds an installer, which installs Coq and other dependencies as configured in A.9.
Priority:	Must have.
Requirements:	CAR-INST-2
Actions taken by the actor and system :	(1) The developer performs the build instructions. (2) The scripts executed will create an installer.

A.11 Installation error log

Goal:	Check the installer error log.
Pre-condition:	The installer encountered an error during installation.
Post-condition:	The user has succeeded in opening the created error log file.
Actors:	The user
Summary:	A user checks the installer error log to get more details on the errors that occurred during installation.
Priority:	Must have.
Requirements:	-
Actions taken by the actor and system :	(1) The user navigates to the error log location. (2) The user opens the error log using a standard text editor. (3) The user can diagnose the problem, or potentially share the error log with the developers.

A.12 Uninstaller error log

Goal:	Check the uninstaller error log.
Pre-condition:	The uninstaller encountered an error during execution.
Post-condition:	The user has succeeded in opening the created error log file.
Actors:	The user
Summary:	A user checks the uninstaller error log to get more details on the errors that occurred during uninstallation.
Priority:	Could have.
Requirements:	-
Actions taken by the actor and system :	(1) The user navigates to the error log location. (2) The user opens the error log using a standard text editor. (3) The user can diagnose the problem, or potentially share the error log with the developers.

A.13 Uninstalling dependencies

Goal:	Uninstall Coq and the selected software packages.
Pre-condition:	The dependencies have been installed through the installer created at A.10 .
Post-condition:	Coq and the selected software packages are uninstalled from the user's machine.
Actors:	The user
Summary:	The user uninstalls the Waterproof dependencies.
Priority:	Must have.
Requirements:	CAR-INST-8
Actions taken by the actor and system :	(1) The user runs the uninstaller program. (2) The uninstaller prompts if the user really wants to uninstall. (3) The uninstaller shows the user a summary of the dependencies that are going to be removed from the user's machine. (4) The user confirms. (5) The uninstaller uninstalls the dependencies. (6) The uninstaller notifies the user that the uninstalling procedure has finished with success or with an error.

A.14 Check for updates

Goal:	Check whether new software updates are available.
Pre-condition:	Waterproof is installed on the user's machine, is functioning correctly and is currently running.
Post-condition:	The system has checked for updates.
Actors:	The user
Summary:	The system checks for new software updates at the user's behest.
Priority:	Should have.
Requirements:	CAR-UPDT-1, CAR-UPDT-2, CAR-UPDT-4, CAR-UPDT-9
Actions taken by the actor and system :	(1) The user presses the "Check for updates" button. (2) The system checks whether new updates of the software are available. If an update is available, the use case A.16 starts. If no updates are available, the system informs the user that they are running the latest version.

A.15 Automatic update settings

Goal:	Change the automatic update check settings (on/off).
Pre-condition:	Waterproof is installed on the user's machine, is functioning correctly and is currently running.
Post-condition:	Waterproof automatic update checks are configured as the user desires.
Actors:	The user
Summary:	The user enables/disables the automatic update check.
Priority:	Should have.
Requirements:	CAR-UPDT-1, CAR-UPDT-2, CAR-UPDT-3
Actions taken by the actor and system :	(1) The user enters the application settings. (2) The user navigates to the updater section. (3) The user enables/disables the automatic update check.

A.16 Updating

Goal:	Update the software packages used by Waterproof, and Waterproof itself.
Pre-condition:	Waterproof is installed on the user's machine, is functioning correctly and is currently running.
Post-condition:	Waterproof, and its dependencies and packages, are updated to a new version if this was possible.
Actors:	The user.
Summary:	A user updates Waterproof.
Priority:	Should have.
Requirements:	CAR-UPDT-5, CAR-UPDT-6, CAR-UPDT-7, CAR-UPDT-8, COR-UPDT-3
Actions taken by the actor and system :	(1) The system detects that a new update is available. (2) The system notifies the user about this, and polls the user whether or not they want to update now. (3) The user can then select if they want to update now, or postpone for later. If the user agreed with updating now: (a) The user selects their updating preferences (such as which packages to update). (b) The system starts the updating procedure. (c) Upon finishing, the system notifies the user that it has finished. (d) The system restarts Waterproof. If the user postpones, the notification is removed.

A.17 Configuring the tactics hints database

Goal:	Configure the hints database that will be loaded.
Pre-condition:	The developer has access to and is using the new tactics library.
Post-condition:	The system concerning the configurability of the hints database is changed to now load the desired hints database.
Actors:	The developer.
Summary:	The developer configures the hints database that they want to use.
Priority:	Must have.
Requirements:	CAR-TL-6
Actions taken by the actor and system :	(1) The developer changes specific function parameters concerning which hints database should be loaded.

A.18 Receiving feedback from the tactics library

Goal:	Feedback (incl. errors) is sent to the user for a selected code-block.
Pre-condition:	The user is currently working on a proof in a particular code-block in the Waterproof application.
Post-condition:	The feedback has reached the user.
Actors:	The user.
Summary:	The user received feedback about their proving code in a selected code-block.
Priority:	Should have.
Requirements:	CAR-TL-9, CAR-TL-10
Actions taken by the actor and system :	(1) The user compiles the selected code-block. (2) If relevant feedback is available, the system sends the feedback to the user.

B Signing Page

Hereby the customer and the supervisor agree upon the requirements stated in this document.

Customer
dr. Jim Portegies

Signature

2021-06-16

Date

Supervisor
dr. Gerard Zwaan

Signature

2021-06-18

Date