BACHELOR SOFTWARE ENGINEERING PROJECT

Acceptance Test Plan

Team Waterfowl
June 29, 2021

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Eindhoven University of Technology

2IPE0 Software Engineering Project
QUARTILE IV - 2020-2021

Acceptance Test Plan

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 0.5
June 29, 2021

Abstract

This document describes the Acceptance Test Plan (ATP) for Waterproof, an educational environment for writing mathematical proofs in interactive notebooks. The project aims to improve the existing Waterproof software in three key areas: introducing a robust installation procedure for the windows platform (i), designing and implementing a robust AST structure (ii), developing a new tactics library based on $L_{\text{tac}}2$ (iii). The tests in this document reflect the requirements in the corresponding User Requirements Document (URD). This document complies with the ESA software standards (cf. [2]).

Contents

Document Status Sheet

Title	Acceptance Test Plan
Authors	Adrien CASTELLA, Adrian CUCOŞ, Cosmin MANEA, Noah VAN DER MEER, Lulof PIRÉE, Mihail ŢIFREA, Tristan TROUWEN, Tudor VOICU, Adrian VRĂMULEŢ, Yuqing ZENG
Version	0.5
Status	Final
Creation Date	May 4, 2021
Last Modified	June 29, 2021

Document History log

May 4th, 2021:	Initial release (version 0.1).
June 16th, 2021:	First revision, for the first acceptance test phase (version 0.2).
June 24th, 2021:	Second revision, for the second acceptance test phase (version 0.3).
June 26th, 2021:	Third revision, after finishing the acceptance test phases (version 0.4).
June 29th, 2021:	Final revision (version 0.5)

1 Introduction

1.1 Purpose

The purpose of the Acceptance Test Plan (ATP) is to specify several test cases that need to be performed by the client in order to validate that the developed product, Robust Waterproof, created throughout the Waterfowl SEP project, is indeed acceptable and practically usable to an extent defined by the customer. These test cases correspond to the requirements as specified in the User Requirements Document (URD). Furthermore, this document describes the environment in which the tests are to be performed, the testing procedures, and the expected results.

1.2 Overview

Besides this one, this document consists of four more chapters/sections.

In Chapter 2, the features to be tested are presented, accompanied with a short overview of the testing process.

In Chapter 3, the specifications of each test case are mentioned. Each test case will consist of a unique identifier, a short description, the prerequisites for executing the respective test, the parameters of the respective test, and the requirements that are tested by the corresponding test case. The input and output specifications are then displayed (again, for each test case).

In Chapter 4, the test procedures are explained. All test cases defined in Chapter 3 will be executed in corresponding logical order. Exact instructions on performing the tests will also be provided.

Finally, in Chapter 5, the results of the corresponding tests from Chapter 3 are presented.

1.3 List of 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	An interactive proof assistant.
Coq-SerAPI	A library for machine-to-machine interaction with the Coq proof assistant.
Cygwin	A program allowing the user to run native Linux applications on Windows.
Executable file	A program that can be run in the OS without explicitly needing to load it with another program.
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 on a computer.
OCaml	A general-purpose typed programming language. OCaml is designed to enhance expressiveness and safety (cf. [1]).
Opam system	A source-based package manager for distributing OCaml programs and tools.
Proof assistants	"Computer programs" specifically designed for mechanizing rigorous mathematical proofs.
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.
GitHub	A provider of Internet hosting for software development, using Git.
Pull request	A method to propose changes to the Git repository of a code project.
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).

1.3.1 Abbreviations

API	Application Programming Interface
AST	Abstract Syntax tree
GB	Gigabytes
Opam	OCaml package manager
OS	Operating System
SEP	Software Engineering Project

1.4 List of references

References

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

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

2 Test plan

In this Section, the items and features that need to be tested are introduced. A general idea of the testing process is also given, consisting of the features that will be tested, the environmental needs for the respective tests, and also the criteria for passing/failing a test.

2.1 Test items

The software to be tested is comprised of the software artefacts designed throughout the Waterfowl project, about a Robust Waterproof. A description of the already existing Waterproof software can be found in the User Requirements Document of the Waterfowl project.

This project in split into three very different parts:

- Introducing a robust installation procedure for Waterproof and all of its dependencies for the Microsoft Windows platform.
- Designing and implementing a robust Abstract Syntax Tree structure internally for the representation of proofs.
- Developing a new tactics library based on $L_{\text{tac}}2$.

All features that are introduced in these three areas will be tested. For additional information, see the User Requirements Document (URD) and the Software Design Document (SDD).

2.2 Features to be tested

All features that have been implemented by the Waterfowl project will be tested. In the User Requirements Document (URD), the requirements corresponding to these features have been assigned a priority according to the MoSCoW model. All features marked with a *Must have* priority have been implemented and will therefore be tested. In addition, some features that have been assigned a *Should have* or *Could have* priority have also been implemented, and as such they are included in the test plan as well. No features corresponding to requirements marked with a priority *Won't have* have been implemented, and therefore no tests will be performed concerning these.

All of the acceptance tests specified in Section 3 were designed in close collaboration with the customer, Jim Portegies.

2.3 Test deliverables

The following items will be delivered before testing starts

- Chapters 1 through 4 of the Acceptance Test Plan (ATP) document.
- The User Requirements Document (URD).
- The software developed by the Waterfowl project.

After the tests have been completed successfully, the following items will be delivered:

- The complete, finalized Acceptance Test Plan (ATP) document.
- Acceptance test outcomes

- Problem reports (if any)

2.4 Testing tasks

Before executing the acceptance test procedure described in chapter 4, the following tasks need to be performed:

- Designing the acceptance tests.
- Link all acceptance tests to the corresponding requirements in the User Requirements Document.
- Ensuring that all environmental needs for the acceptance tests (described in Section 2.5) are satisfied.

2.5 Environmental needs

As explained before, the Waterfowl project is aimed at improving three key aspects of the Waterproof software. Due to the nature of the first aspect (the installation procedure), we will distinguish between the environments used for testing this first aspect and the other two aspects. The User Requirements Document (URD) contains a clear specification of the features and requirements related to each of these aspects.

For testing the features and requirements related to the installation procedure, the following resources will be needed:

- Virtual machines for the 3 supported versions of Microsoft Windows 10

More details on this can be found in Section 4.

For testing features and requirements related to the remaining two aspects (the implementation of the abstract syntax tree and development of the new tactics library), the following resources will be needed:

- A desktop or laptop computer running a version of the Linux operating system that supports the Coq proof assistant (such as Ubuntu 20.04).
- The Waterproof software, along with the improvements made by the Waterfowl project, is installed and running.
- The Coq proof assistant, which is a back-end dependency of the Waterproof software.

2.6 Test case pass/fail criteria

In Chapter 3, the acceptance test cases are specified in detail. A test case is passed if the received feedback matches the output specified for the test case, provided that the adequate parameters needed for the respective test are used (these parameters will be mentioned in Chapter 4). If the input matches the specification, but the provided feedback differs in any way from the expected output, the test case fails. If a test case fails, the overall acceptance test is considered to have failed.

3 Test case specifications

As described in Section 2.1, the Waterfowl project is divided into three parts. This chapter is hence divided into four parts: acceptance tests for general requirements, acceptance tests for the installer development part of the project, acceptance tests for the Abstract Syntax Tree development part of the project and acceptance tests for the new tactics library development part of the project.

ATP for general requirements

This section is devoted to the acceptance tests for the general requirements of the Waterfowl project.

ATP-1 New Waterproof still compatible with Coq

Preconditions:	The new Waterproof software is installed on the local machine
Parameters:	None
Requirements:	COR-GEN-2.

Input Specifications	Output Specifications

1. Start the Waterproof software.
2. Create a new, empty Waterproof notebook.
3. Run any sequence of supported Coq commands, such as
`Goal forall n : nat, n = n.`
`intro n.`
`trivial.`	

	4. Waterproof successfully compiles the sequence of commands

ATP-2 Software merged back to the original GitHub repository

Preconditions:	None
Parameters:	None
Requirements:	COR-GEN-1.

Input Specifications	Output Specifications
1. Access the website `https://github.com/impermeable/` and access the `coq-waterproof` project.	
2. Check that all the code for the new tactics library is present in this GitHub repository.	
	3. The new tactics library can indeed be found in this repository.
4. Access the website `https://github.com/impermeable/waterproof` and, upon loading, navigate to the ”Pull requests” tab.	
5. Check that a pull request with the title ”SEP Project - Robust Waterproof” exists.	
6. After opening the pull request, check that all the code for the AST parser is present in this pull request and can be merged into the main repo.	
	7. The AST parser code can be merged into the new repository.
8. Access the website `https://github.com/impermeable/` and access the `waterproof-dependencies-installer` project.	
9. Check that all the code for to create the dependencies installer is present in this GitHub repository.	
	10. The dependencies install code can indeed be found in this repository.

ATP for the installer development part of the project

This section is devoted to the acceptance tests for the installer development part of the Waterfowl project.

ATP-3 Installer file size

Preconditions:	None
Parameters:	None
Requirements:	COR-INST-1.

Input Specifications	Output Specifications
1. Open a file browser and locate the Coq platform installer	
2. Open the file properties	
	3. A file size smaller than 500MB is shown

ATP-4 User interactions and Language

Preconditions:	None
Parameters:	None
Requirements:	COR-INST-3, COR-INST-6, COR-INST-9

Input Specifications	Output Specifications
1. Start the Coq platform installer application	
	2. The Coq platform window appears
3. Press ”next”	
	4. A page showing the license agreement is shown
5. Press ”I agree”	
	6. A page in which packages can be selected appears
7. Press ”next”	
	8. A page in which the installation directory can be specified appears
9. Press ”install” to commence the installation	
	10. The installer window confirms that Coq has been installed

ATP-5 Advanced user settings

Preconditions:	None
Parameters:	None
Requirements:	CAR-INST-1

Input Specifications	Output Specifications
1. Start the Coq platform installer application	
	2. The Coq platform window appears
3. Press ”next”, and then accept the license agreement by pressing ”I agree”	
	4. A page in which packages can be selected appears
5. Select the packages to be installed and press ”next”	
	6. A page in which the installation directory can be specified appears

ATP-6 Installing dependencies using Coq platform

Preconditions:	None
Parameters:	None
Requirements:	CAR-INST-4, CAR-INST-5, CAR-INST-6, CAR-INST-7, CAR-INST-9, COR-INST-2.

Input Specifications	Output Specifications
1. Start the Coq platform installer application	
	2. The Coq platform window appears
3. Press ”next”, and then accept the license agreement by pressing ”I agree”	
	4. A page in which packages can be selected appears
5. Select the packages to be installed and press ”next”	
	6. A page in which the installation directory can be specified appears
7. Specify the installation directory 8. Commence the installation by pressing the “Install” button	
	9. The installer window displays ”Coq has been installed on your computer”, and Coq, SerAPI, Coquelicot, Coq-ZFC are installed on the computer
10. Press ”Finish” to complete the Coq platform installation 11. Install the Waterproof installer using the installer	
	10. The Waterproof software starts and successfully initializes

ATP-7 Uninstalling

Preconditions:	The Coq platform has been installed
Parameters:	None
Requirements:	CAR-INST-8

Input Specifications	Output Specifications
1. Open the Windows Control panel	
2. Navigate to the "Uninstall a program" page	
3. Select Coq from the list, and press "Uninstall"	
	4. The uninstaller window appears
5. Press "next"	
	6. A page in which the installation directory can be specified appears. A location has been filled in already
7. Commence removal by pressing the "Uninstall" button	

ATP-8 Updating

Preconditions:	An outdated version of Waterproof is installed
Parameters:	None
Requirements:	CAR-UPDT-1, CAR-UPDT-3, CAR-UPDT-5, CAR-UPDT-8, CAR-UPDT-9

Input Specifications	Output Specifications
1. Start the Waterproof software	
	2. A pop-up appears asking you whether you want to update
3. Press the button ”Yes, please”	
	4. The pop-up is closed.
5. Close the Waterproof software and wait 5 seconds	
6. Start the Waterproof software, and navigate to the ”about” menu	
	7. The latest version of Waterproof is displayed.

ATP-9 Updater Security

Preconditions:	An outdated version of Waterproof is installed
Parameters:	None
Requirements:	COR-UPDT-1, COR-UPDT-2

Input Specifications	Output Specifications
1. Open the Windows Command Prompt. 2. Navigate to the directory in which Waterproof has been installed. 3. Start the software by typing ”Waterproof.exe”.	
	4. A pop-up appears asking you whether you want to update
5. Press the button ”Yes, please”	
	6. The pop-up is closed. 7. In the Command Prompt, logging is provided showing that the update is being downloaded. The sources used for these downloads use HTTPS

ATP-10 Installer configurability

Preconditions:	Waterproof installed using default settings.
Parameters:	None
Requirements:	CAR-INST-2, CAR-INST-3

Input Specifications	Output Specifications
1. Open the configuration file.	
	2. The configuration file shows how to configure which Opam packages to install by default.
3. Open Waterproof. 4. Create a code cell. 5. Inside the code cell, type `Require Import Waterproof.AllTactics.` and press Alt+DownArrow.	
	6. Waterproof returns a checkmark.

ATP for the Abstract Syntax Tree development part

This section is devoted to the acceptance tests for the Abstract Syntax Tree development part of the Waterfowl project.

ATP-11 Syntax Highlighting

Preconditions:	The `ATP1.wpn` file exists in the specified folder and the newest version of waterproof is installed.
Parameters:	None
Requirements:	CAR-AST-2, COR-AST-4.

Input Specifications	Output Specifications
1. Open the Waterproof application. 2. Press the 'Open notebook' button in the center of the application. 3. Navigate to `waterproof/tests/unit/coqast/helpers`. 4. Open the `ATP1.wpn` notebook. 5. Open the Run tab in the top left corner of Waterproof. 6. Press on the 'Output All ASTs' button.	
	7. The syntax is highlighted according to the structure of the CoqAST.

ATP-12 Pretty-printer

Preconditions:	The `ATP1.wpn` file exists in the specified folder and the newest version of waterproof is installed.
Parameters:	None
Requirements:	CAR-AST-1.

Input Specifications	Output Specifications
1. Open the Waterproof application. 2. Press the 'Open notebook' button in the center of the application. 3. Navigate to `waterproof/tests/unit/coqast/helpers`. 4. Open the `ATP1.wpn` notebook. 5. Press `CTRL+SHIFT+I` to open the console in Waterproof. 6. Open the Run tab in the top left corner of Waterproof. 7. Press the 'Output All ASTs' button.	
	8. The corresponding parsed CoqASTs appear in the console log.

ATP-13 AST-flattening

Preconditions:	The `ATP1.wpn` file exists in the specified folder and the newest version of waterproof is installed.
Parameters:	None
Requirements:	CAR-AST-2, COR-AST-4.

Input Specifications	Output Specifications
1. Open the Waterproof application. 2. Press the 'Open notebook' button in the center of the application. 3. Navigate to `waterproof/tests/unit/coqast/helpers`. 4. Open the `ATP1.wpn` notebook. 5. Press `CTRL+SHIFT+I` to open the console in Waterproof. 6. Open the Run tab in the top left corner of Waterproof. 7. Press the 'Output All ASTs' button.	
	8. The flattened CoqAST is printed in the console for each line.

ATP-14 AST-parser error

Preconditions:	The newest git repository for Waterproof is cloned on the device.
Parameters:	None
Requirements:	CAR-AST-7.

Input Specifications	Output Specifications
1. Open the terminal in the Waterproof project folder. 2. Navigate to tests directory with the command `cd waterproof/tests/unit/coqast/helpers` 2. Run the command `npx ts-node shell.ts`. 3. Log the pre-loaded variable `badSExp` in the terminal shell.	
	4. The bad expression is printed in the terminal.
5. Run the code `toAST(badSExp)`.	
	6. The terminal outputs a `null` object.
7. Log the pre-loaded variable `goodSExp` in the terminal shell.	
	8. The good expression is printed in the terminal.
9. Run the code `toAST(goodExpr)`.	
	10. The terminal outputs the parsed data structure.
11. Possibility to repeat this procedure with CoqAST S-expressions made by customer.	

ATP-15 Unparsed type error

Preconditions:	The newest git repository for Waterproof is cloned on the device.
Parameters:	None
Requirements:	CAR-AST-1, CAR-AST-7.

Input Specifications	Output Specifications
1. In the in the folder `Waterproof/src/coq/serapi` access the js file `ASTProcessor.js` 2. Comment out one of the Coq types from the variable `constrDict`. 3. Open the Waterproof application. 4. Press the 'Open notebook' button in the center of the application. 5. Navigate to `waterproof/tests/unit/coqast/helpers` 6. Open the `ATP1.wpn` notebook. 7. Press `CTRL+SHIFT+I` to open the console in Waterproof. 8. Open the Run tab in the top left corner of Waterproof. 9. Press the 'Output All ASTs' button.	
	10. The terminal outputs a warning message with the name of the un-parsed (commented-out) Coq type.

ATP-16 Runtime of parsing

Preconditions:	The newest git repository for Waterproof is cloned on the device.
Parameters:	None
Requirements:	COR-AST-1, COR-AST-2.

Input Specifications	Output Specifications
1. Open the terminal in the Waterproof project folder. 2. Navigate to tests directory with the command `cd waterproof/tests/unit/coqast/helpers`. 2. Run the command `npx ts-node shell.ts`. 3. Log the pre-loaded variable `longSExp` in the terminal shell.	
	4. The long expression is printed in the terminal.
5. Run the code `const out = toASTwithTime(goodSExp)`. 6. Log the variable `out[0]` in the terminal shell.	
	7. The terminal outputs the time needed to parse the long expression.
8. Log the variable `out[1]` in the terminal shell.	
	9. The terminal outputs the parsed long expression.

ATP for the new tactics library development

The following tests will now be about the development of the new tactics library (cf.2.1). The prerequisite for most tests is that the user has cloned the Git repository of the branch

```
gitlab.tue.nl/sep-group-2-2020-2021-q4/waterproof/-/tree/new_tactics_lib
```

and saved it on their local machine, in a particular folder (this will also be mentioned in Section 4). We denote this folder as `branch`, and all of the folder paths that are specified in the coming tests will start in the folder

```
branch/waterproof/coq_tactics/waterproof/.
```

ATP-17 New tactics library compiles within three minutes

Preconditions:	The corresponding Git repository is cloned on the local machine.
Parameters:	None
Requirements:	COR-TL-8.

Input Specifications	Output Specifications
1. Open the Linux terminal and go to the folder `new_wplib`. 2. Run the command `make`. 3. Wait for the operation to finish.	
	4. Check that the above operation is ran without errors, and it is finished within three minutes after the time it was started.

ATP-18 LTAC2 Implemented number of the old tactics

Preconditions:	The corresponding Git repository is cloned on the local machine.
Parameters:	None
Requirements:	CAR-TL-1.

Input Specifications	Output Specifications
1. Open the old Waterproof tactics files `Tactics.v` and `TacticsContra.v` in the folder `wplib/Tactics`, and the new tactics library collection of tactic files in the folders `new_wplib/tactics` and `new_wplib/contradiction_tactics`.	
	2. The respective old tactics library files and new tactics library files open.
3. Manually check each file from the new tactics library and count the overall number of tactics that have been implemented in LTAC2.	
	4. At least 80% of the tactics from the old tactics library are implemented by the new tactics library.

ATP-19 LTAC2 Implemented customizable `waterprove`

Preconditions:	The corresponding Git repository is cloned on the local machine.
Parameters:	None
Requirements:	CAR-TL-2, CAR-TL-7, CAR-TL-8.

Input Specifications	Output Specifications
1. In the new tactics library collection of files, open the `waterprove.v` file in the `new_wplib/waterprove` folder.	
	2. The corresponding `waterprove.v` and the `test_load_database.v` files open.
3. Manually check that the `waterprove` automatic tactic is implemented in the `waterprove.v` file.	
	4. The `waterprove` automation tactic is fully implemented.
5. Manually check that the `waterprove` automation tactic is implemented to take a `lemmas` parameter, which is a list.	
	6. The `waterprove` automation tactic indeed takes a list parameter under the name of `lemmas`.
7. Manually check that the `waterprove` automation tactics is implemented to take a `search_depth` argument, which is an integer.	
	8. The `waterprove` automation tactic indeed takes an integer parameter under the name of `search_depth`.

ATP-20 Tactics library bundled as an OPAM package

Preconditions:	The OPAM software is installed on the local machine.
Parameters:	None
Requirements:	CAR-TL-3.

Input Specifications	Output Specifications
1. Access the website `https://github.com/impermeable`	
2. Access the project called `coq-waterproof`	
3. Check that the respective repository indeed contains the newly desired tactics library, by manually checking the file structure of the project.	
	4. The respective repository indeed contains the newly desired tactics library.
5. Check that a `Makefile` exists in the respective repository.	
	6. A `Makefile` exists in this particular repository, signifying that the library is bundled as an OPAM package.
7. Check that a `_CoqProject` exists in the respective repository. 8. Open the `_CoqProject` file and manually check that all the tactics from the new tactics library are written to be compiled.	
	9. The `_CoqProject` file indeed contains all the tactics from the new tactics library, and they are all written to be compiled.
10. Open the Linux terminal and run the command `opam repo add coq-released https://coq.inria.fr/opam/released`	
11. Run the command `opam install -v coq-waterproof`	
	12. The `coq-waterproof` package (containing the new tactics library) is successfully installed, without errors.

ATP-21 Pull request made under the name Waterproof

Preconditions:	None
Parameters:	None
Requirements:	CAR-TL-4, COR-TL-4.

Input Specifications	Output Specifications
1. Access the website `github.com/coq/opam-coq-archive/pull/1761#discussion_r657844383`	
2. Check that a pull request for the package `coq-waterproof.1.0.0`, containing the new tactics library, is made.	
	3. The respective pull request in indeed made and contains the new tactics library.
4. Access the website `https://github.com/impermeable`	
5. Open the file `_CoqProject`, and check that the first line in the respective file is `-R waterproof/ Waterproof`	
	6. The `_CoqProject` file indeed has `-R waterproof/ Waterproof` as its first line, signifying that the name of the library is `Waterproof` and that all tactics files will be found under the `waterproof` folder.

ATP-22 New tactics library released to the Coq Git repository, under the name Waterproof

Preconditions:	None
Parameters:	None
Requirements:	CAR-TL-5, COR-TL-5.

Input Specifications	Output Specifications
1. Access the website `github.com/coq/opam-coq-archive` 2. Access the folder `released`. 3. Access the folder `packaged`. 4. Check that the folder `coq-waterproof/coq-waterproof.1.0.0` exists in the current directory.	
	5. The respective folder indeed exists in the current directory, signifying that the new tactic library has been released to the Coq Git repository. The fact that the library was released under the name `Waterproof` is enforced by ATP-21.

ATP-23 Configurability of the hints databases

Preconditions:	The corresponding Git repository is cloned on the local machine.
Parameters:	None
Requirements:	CAR-TL-6.

Input Specifications	Output Specifications
1. Open the file `selected_database.v` in the forder `new_wplib` and the file `test_load_database.v` in the folder `new_wplib/test`.	
	2. The corresponding files open.
3. Manually check the `selected_database.v` file and check that the list of hints databases that can be used is configurable.	
	4. The list of hints databases that can be used in indeed configurable.
5. Run all the tests in the file `test_load_database.v`.	
	6. The corresponding tests are running without errors and are performing in the desired way, concluding that the configurability of the hints databases is working properly.

ATP-24 Coq error messages encapsulated in user-friendly messages

Preconditions:	The corresponding Git repository is cloned on the local machine.
Parameters:	None
Requirements:	CAR-TL-9.

Input Specifications	Output Specifications
1. Open the file `encapsulated_errors_atp_test.v` file from the new tactics library, in the folder `atp_v_files`.	
	2. The corresponding file opens.
3. Run all the tests from the `encapsulated_errors_atp_test.v` file and check that the caught `Coq` errors are replaced with custom, user-friendly error messages.	
	4. The caught `Coq` errors are indeed replaced by custom, user-friendly error messages.

ATP-25 Tactics library gives feedback to the user

Preconditions:	The corresponding Git repository is cloned on the local machine.
Parameters:	None
Requirements:	CAR-TL-10.

Input Specifications	Output Specifications
1. Open the file `user_messages_atp_test.v` file from the new tactics library, in the folder `atp_v_files`.	
	2. The corresponding file opens.
3. Run all the tests from the `user_messages_atp_test.v` file and check that the tactics used within the tests indeed give feedback to the user.	
	4. The tactics used indeed give feedback to the user, by means of printed messages.

ATP-26 Implemented the old tactics in LTAC2, allowing for synonyms in tactics

Preconditions:	The corresponding Git repository is cloned on the local machine.
Parameters:	None
Requirements:	COR-TL-1, COR-TL-11.

Input Specifications	Output Specifications
1. Open each `.v` file from the folders `new_wplib/tactics` and `new_wplib/contradiction_tactics` from the new tactics library collection of files.	
	2. The respective files from the new tactics library open.
3. Manually check that, in each file, LTAC2 was used in place of LTAC1, except for some of the tactics/files specifically discussed with the customer in advance.	
	4. Each tactic from each file (except the ones discussed with the customer in advance) uses indeed LTAC2 instead of LTAC1.
5. Manually check that, in each file, every tactic is implemented by calling a particular function, thereby allowing new, synonymous tactics to be defined by called the same function.	
	6. Each implemented tactic indeed functions by calling a particular function.
7. Manually check that, in the files `we_show_both_statements.v` and `choose_such_that.v`, the tactics in these two files are implemented using variables of type `opt`, signifying that they are optional variables.	
	8. The corresponding two files indeed implement their functionality using optinal variables (of type `opt`).

ATP-27 Compatibility of the new tactics library by means of good proof writing

Preconditions:	The corresponding Git repository is cloned on the local machine.
Parameters:	None
Requirements:	COR-TL-2, COR-TL-10.

Input Specifications	Output Specifications
1. In the folder `atp_v_files`, open the files `sets.v`, `sequences.v`, `series.v`, `subsequences.v`, `sup_and_inf.v`, `sup_and_inf_new_definitions.v`, `limsum_liminf_bolzano.v` and `sequential_accumulation_points.v`.	2. The respective files open.
3. For each file, manually check that whenever a tactic, other than the `Choose`, `Choose ... such that ...`, `Define`, `Rewrite inequality`, `Help`, `It suffices to show that`, `Write ... using`, `Write ... as` and the proof finishing tactics, is used, introducing a new claim/variable is done by specifically naming the respective variable/claim and specifying its type.	4. The tactics (except the ones specifically mentioned before) indeed adhere to firstly defining the variable/claim and then concretely specifying its type.
5. For each file, manually check (by holding `CTRL` and pressing, one step at a time, the down (↓) arrow) that each line from the respective file compiles without errors.	6. Each file compiles in full without any errors.

ATP-28 Modularity of the new tactics library without unneeded dependencies

Preconditions:	The corresponding Git repository is cloned on the local machine.
Parameters:	None
Requirements:	COR-TL-3, COR-TL-6.

Input Specifications	Output Specifications
1. In the folder `wplib`, open the `Tactics.v` file, and in the forlder `new_wplib`, open the `tactics` folder.	2. The respective folders and files open.
3. Manually check that the (file) structure of the new tactics library is a collection of many `.v` files, as opposed to the old tactics library, which is comprised of a single tactics file.	4. The new tactics library is indeed a collection of many `.v` files.
5. Manually check (by looking at the imported files) that almost all the tactics (except the files `write_using.v`, `write_as.v`, `rewrite_inequalities.v`) and `proof_finishing_tactics.v` depend only on auxiliary functions or the `waterprove` automation tactic, and not on each other. The auxiliary functions are the files `auxiliary.v`, `test_auxiliary.v`, `foward_reasoning_aux.v` and `decidability_db.v`.	6. The corresponding tactics files (except the specified ones) indeed do not depend on each other, but only on auxiliary functions and the `waterprove` automation tactic.
7. Manually check (by looking at the imported files) that the files `write_using.v`, `write_as.v` and `rewrite_inequalities.v` depend on the `rewrite_using.v` file, but only because they use the same functionality as the tactics from the `rewrite_using.v` file, but in a different context. Manually check (by looking at the imported files) that the file `proof_finishing_tactics.v` depends on the `we_conclude_that.v` file, but only because it uses the same functionality of the tactics from the `proof_finishing_tactics.v` file, but in a different context.	

	8. The last respective four files indeed depend on the respective mentioned other files, but only because they use the same functionality of their respective dependencies in a different context.

ATP-29 Automation implemented only once

Preconditions:	The corresponding Git repository is cloned on the local machine.
Parameters:	None
Requirements:	COR-TL-7.

Input Specifications	Output Specifications
1. In the folder `new_wplib/tactics/forward_reasoning`, open the `forward_reasoning_aux.v` file.	
	2. The respective file opens.
3. Manually check the that the functions `waterprove_with_hint` and `waterprove_without_hint` are used to wrap the `waterprove` automation tactic into a specific context with a single lemma.	
	4. The respective two functions indeed wrap the call of the `waterprove` automation tactic in the context of a single lemma.
5. Open the files `it_holds_that.v`, `it_suffices_to_show_that.v`, `proof_finishing_tactics.v`, `rewrite_using.v`, `we_conclude_that.v` and `rewrite_inequalities.v`.	
	6. The respective files open.
7. Manually check that the previously mentioned files use the function `waterprove_with_hint` or the function `waterprove_without_hint` in the definition of their respective tactics.	
	8. The previously mentioned files indeed use at least one of the functions `waterprove_with_hint` or `waterprove_without_hint`.

4 Test procedures

This chapter describes the procedures which should be followed to perform the acceptance test so that all test specifications are covered. Firstly, in Section 4.1, various preconditions are specified. Secondly, in Section 4.2, all of the test procedures are given. If all of the test procedures are executed and all test cases are successful, the acceptance test is complete.

4.1 Preconditions

For the test procedures Abstract Syntax Tree development part of the project, the notebook `ATP1.wpn` must exists in the file path `waterproof/tests/unit/coqast/helpers`. This is immediately true when cloning the most recent version of the Git repository. Additionally, the newest version of the Waterproof application must be installed on the device which is running the ATP tests. Finally, the newest Git repository itself must be cloned on the device in order to run shell codes in tests ATP-14 and ATP-16. The previously mentioned Git repository can be found at

`https://gitlab.tue.nl/sep-group-2-2020-2021-q4/waterproof/-/tree/develop`.

For the test procedure of the new tactics library development part of the project, as we previously mentioned in the corresponding testing section, the Git repository of the Waterproof software,

`gitlab.tue.nl/sep-group-2-2020-2021-q4/waterproof/-/tree/new_tactics_lib`,

must be locally cloned on a machine running a version of the Linux operating system that supports the `Coq` software (such as Ubuntu 20.04).

For the test procedure of the installer development part of the projet, virtual machines running the following versions of Microsoft Windows 10 should be available:

- Windows 10 20H1 amd64
- Windows 10 20H2 amd64
- Windows 10 21H1 amd64

Moreover, these should be set up according to the following configurations:

- A variant containing only the default Windows 10 installation
- A variant in which in addition to the default Windows 10 installation, also the Cygwin, Opam, Coq software are installed.

Using this approach, a total of 6 virtual machines are necessary. In order to efficiently manage and configure these virtual machines, a system running the Xen hypervisor is assumed to be running. As was requested by the customer, instructions for setting up a single virtual machine on a regular desktop computer are provided in the next section.

It is assumed that the files for the Coq platform installer and the latest Waterproof installer are available, as developed by the Waterfowl project. Moreover, in order to test the update functionality, a file for the installer of an older version of Waterproof should also be available, again as developed by the Waterfowl project.

4.1.1 Setting up a Virtual Machine

The following steps are intended to give some guidance on setting up a single Windows 10 virtual machine through VirtualBox. They are intentionally high-level, since for the actual acceptance test the Xen hypervisor is used to manage virtual machines.

- Download and install the Oracle VirtualBox software from the official website.
- Download an image (ISO) of the desired Windows 10 operating system from the Microsoft website (note: no links are provided since these are changed on a daily basis by Microsoft).
- In VirtualBox, create a new virtual machine through the guided setup tool. Select the "Windows 10" operating system type, and create a virtual hard disk with sufficient disk space (e.g. 50gb).
- After the virtual machine is created, open the virtual machine settings, and assign the previously downloaded Windows 10 image to the optical drive.
- Start the virtual machine, and proceed with the usual Windows 10 installation procedure.

4.2 Procedures

As our project is split into three parts, we divide the procedures into the corresponding three parts as well. The procedures should then be ran in this order: firstly, Section 4.2.1, secondly, Section 4.2.2 and lastly, Section 4.2.3.

4.2.1 Installer development part

1. Start a virtual machine running Windows 10 21H1.
2. Complete ATP-3 - Installer file size on this VM.
3. Complete ATP-4 - User interactions and Language on this VM.
4. Complete ATP-10 - Installer configurability on this VM.
5. Complete ATP-7 - Uninstalling on this VM.
6. Complete ATP-5 - Advanced user settings on this VM. Do not commence the actual installation.
7. Complete ATP-6 - Installing dependencies using Coq platform on this VM.
8. Start a virtual machine running Windows 10 20H1.
9. Complete ATP-6 - Installing dependencies using Coq platform on this VM.
10. Start a virtual machine running Windows 10 20H2.
11. Complete ATP-6 - Installing dependencies using Coq platform on this VM.
12. Start a virtual machine running variant 2 of Windows 10 21H1.
13. Complete ATP-6 - Installing dependencies using Coq platform on this VM.
14. Start a virtual machine running variant 2 of Windows 10 20H1.
15. Complete ATP-6 - Installing dependencies using Coq platform on this VM.

16. Start a virtual machine running variant 2 of Windows 10 20H2.
17. Complete ATP-6 - Installing dependencies using Coq platform on this VM.
18. Complete ATP-1.
19. Start any Windows virtual machine with internet access and install a version of Waterproof that is 1 version behind the latest version
20. Complete ATP-8 - Updating
21. Uninstall the Waterproof software, and install a version of Waterproof that is 1 version behind the latest version
22. Complete ATP-9 - Updater Security

4.2.2 Abstract Syntax Tree development part

1. Complete ATP-11 - Syntax Highlighting.
2. Complete ATP-12 - Pretty-printer.
3. Complete ATP-13 - AST-flattening.
4. Complete ATP-15 - Unparsed type error.
5. Complete ATP-14 - AST-parser error.
6. Complete ATP-16 - Runtime of parsing.

4.2.3 New tactics library development part

1. Complete ATP-20 - Tactics library bundled as an OPAM package.
2. Complete ATP-21 - Pull request made under the name Waterproof.
3. Complete ATP-22 - New tactics library released to the Coq Git repository, under the name Waterproof.
4. Complete ATP-17 - New tactics library compiles within three minutes.
5. Complete ATP-28 - Modularity of the new tactics library without unneeded dependencies.
6. Complete ATP-18 - LTAC2 Implemented number of the old tactics
7. Complete ATP-26 - Implemented the old tactics in LTAC2, allowing for synonyms in tactics.
8. Complete ATP-19 - LTAC2 Implemented customizable `waterprove`.
9. Complete ATP-29 - Automation implemented only once.
10. Complete ATP-23 - Configurability of the hints databases.
11. Complete ATP-27 - Compatibility of the new tactics library by means of good proof writing.
12. Complete ATP-24 - Coq error messages encapsulated in user-friendly messages.
13. Complete ATP-25 - Tactics library gives feedback to the user.
14. Complete ATP-2 - Software merged back to the original GitHub repository.

5 Test reports

The rest reports and the results of the Acceptance Tests can be found in Chapter 5 of the Waterfowl Software Transfer Document (STD).

6 Traceability matrix

URD Requirement	Test case	Priority
CAR-INST-1	ATP-5	Must have
CAR-INST-2	ATP-10	Must have
CAR-INST-3	ATP-10	Must have
CAR-INST-4	ATP-6	Must have
CAR-INST-5	ATP-6	Must have
CAR-INST-6	ATP-6	Should have
CAR-INST-7	ATP-6	Must have
CAR-INST-8	ATP-7	Must have
CAR-INST-9	Requirement not implemented	Should have
CAR-INST-10	ATP-6	Could have
CAR-INST-11	Requirement not implemented	Could have
CAR-INST-12	Requirement not implemented	Won't have
CAR-INST-13	Requirement not implemented	Won't have
CAR-UPDT-1	ATP-8	Could have
CAR-UPDT-2	Requirement not implemented	Should have
CAR-UPDT-3	ATP-8	Should have
CAR-UPDT-4	Requirement not implemented	Could have
CAR-UPDT-5	ATP-8	Should have
CAR-UPDT-6	Requirement not implemented	Should have
CAR-UPDT-7	Requirement not implemented	Should have
CAR-UPDT-8	ATP-8	Should have
CAR-UPDT-9	ATP-8	Should have
CAR-AST-1	ATP-12, ATP-15	Must have
CAR-AST-2	ATP-11, ATP-13	Must have
CAR-AST-3	Requirement not implemented	Should have
CAR-AST-4	Requirement not implemented	Should have
CAR-AST-5	Requirement not implemented	Should have
CAR-AST-6	Requirement not implemented	Won't have
CAR-AST-7	ATP-14, ATP-15	Should have
CAR-AST-8	Requirement not implemented	Could have
CAR-TL-1	ATP-18	Must have
CAR-TL-2	ATP-19	Must have
CAR-TL-3	ATP-20	Must have
CAR-TL-4	ATP-21	Must have
CAR-TL-5	ATP-22	Should have
CAR-TL-6	ATP-23	Must have
CAR-TL-7	ATP-19	Should have
CAR-TL-8	ATP-19	Should have
CAR-TL-9	ATP-24	Should have
CAR-TL-10	ATP-25	Should have
CAR-TL-11	Requirement not implemented	Could have

URD Requirement	Test case	Priority
COR-GEN-1	ATP-2	Must have
COR-GEN-2	ATP-1	Must have
COR-GEN-3	Requirement not implemented	Could have
COR-INST-1	ATP-3	Must have
COR-INST-2	ATP-6	Must have
COR-INST-3	ATP-4	Must have
COR-INST-4	Requirement not implemented	Could have
COR-INST-5	Requirement not implemented	Should have
COR-INST-6	ATP-4	Must have
COR-INST-7	Requirement not implemented	Could have
COR-INST-8	Requirement not implemented	Could have
COR-INST-9	ATP-4	Must have
COR-INST-10	Requirement not implemented	Could have
COR-INST-11	Requirement not implemented	Should have
COR-INST-12	Requirement not implemented	Could have
COR-INST-13	Requirement not implemented	Should have
COR-UPDT-1	ATP-9	Must have
COR-UPDT-2	ATP-9	Must have
COR-UPDT-3	Requirement not implemented	Should have
COR-AST-1	ATP-16	Must have
COR-AST-2	ATP-16	Should have
COR-AST-3	Requirement not implemented	Could have
COR-AST-4	ATP-11, ATP-13	Should have
COR-AST-5	Requirement not implemented	Should have
COR-TL-1	ATP-26	Must have
COR-TL-2	ATP-27	Must have
COR-TL-3	ATP-28	Must have
COR-TL-4	ATP-21	Must have
COR-TL-5	ATP-22	Should have
COR-TL-6	ATP-28	Should have
COR-TL-7	ATP-29	Should have
COR-TL-8	ATP-17	Should have
COR-TL-9	Requirement not implemented	Should have
COR-TL-10	ATP-27	Could have
COR-TL-11	ATP-26	Could have

A Signing Page

Hereby the customer confirms that all test cases and test procedures were formulated according to their wishes and have been completed successfully.

Customer
dr. Jim Portegies

[illegible]

....
Signature

Jul 1, 2021

....
Date