D5.6 Validation report

The DECODER project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 824231

Project Acronym	DECODER
Project Title	DEveloper COmpanion for Documented and annotatEd code Reference
Project Number	824231
Project Type	Research and Innovation Action (RIA)
Thematic Priority	ICT-16-2018 Software Technologies
Start Date of Project	January 1st, 2019
Duration of Project	36 months

Deliverable type	R
Deliverable ref. number	DS-05-824231 / D5.6 / V1.0
Work Package	**WP5**
Task	**T5.4**
Due Date	December 2021 – M36
Submission Date	January 27th, 2022
Dissemination Level	PU

Responsible organisation:	TREE	
Author(s)	Pablo de Castro	TREE
	Javier Gutiérrez	TREE
Reviewer(s)	Pierre-Yves Gibello	OW2

Version history

0.1	09-12-2021	Template	TREE
0.2	13-12-2021	Initial contributions	TEC
0.3	12-01-2022	UPV experiments	UPV
0.4	24-01-2022	Professional experiments	TREE
0.5	26-01-2022	First revision and document update	OW2, CEA, TREE
0.6	27-01-2022	Second revision and document update	OW2, TREE
1.0	27-01-2022	Final version	TREE

Table of Contents

List of Tables

List of Figures

Executive Summary

This deliverable D5.6 “Validation report” presents the activities carried out to validate the DECODER methodology defined under WP5 “Methodology” and consolidated and showcased in deliverables D5.4 “Updated methodology design report” and D5.5 “Methodology tool support”.

The activities to validate the DECODER methodology have focused on two complementary fields: (1) definition, testing and validation of a set of use cases involving the different roles (i.e., Administrator, Developer, Maintainer and Reviewer) within the software cycle and (2) guided experiments with two different groups (students and professionals), which have used diverse tools, explored the corresponding results and interacted with the Graphical User Interface (GUI). Both have contributed to the evaluation, but also to the identification of possible improvements of the DECODER Persistent Knowledge Monitor (PKM) and Process Engine (PE) and the improvement of the User Experience (UX) before the final release and demonstration.

Feedback from the two groups is presented in this report, showing that, although around 50% of them consider the DECODER environment to be useful, students think that improvements should be made in terms of usability and understandability.

Contributing tasks of this WP	T5.2, T5.3
Related deliverables of this WP	D5.4, D5.5
Input from other WP(s)	WP4 (GUI)
Output to other WP(s)	WP5

Keywords

DECODER, methodology, GUI, validation, use cases

Acronyms and Abbreviations

ASFM	Abstract Semi Formal Modelling
ER	Error
GUI	Graphical User Interface
NER	Named Entity Recognition
NLP	Natural Language Processing
PE	Process Engine
PKM	Persisting Knowledge Monitor
RI	Recommendation for Improvement

SP	Semantic Parsing
SRL	Semantic Role Labelling
UV	Universitat de València
UX	User eXperience

1 Introduction

As previously commented in deliverables D5.2 and D5.4, the DECODER consortium has been working on a methodology to assist the stakeholder with (1) an instantaneous access to project documentation, abstract models and verification processes, (2) an access to a virtual expert in order to produce code that is functionally correct, free of run-time errors, and with a control on memory space and execution time, (3) facilities that verify requirements, and (4) facilities that produce user documentation conformant to requirements.

Once completed, the work plan has already considered the validation of the DECODER methodology and the integrated tools in the context of different scenarios. This is the aim of task T5.4 “Methodology application and validation”, whose results can be found in this deliverable.

1.1 Document organisation

This report is structured as follows:

- Section 2 deals with the definition of the test cases and execution of the corresponding tests for validation and UX improvement.
- Section 3 describes the experiment conducted by a group of students from two different universities – i.e., Universitat Politècnica de València (UPV) and the Universitat de València (UV).
- Section 4 describes the experiment conducted by a group of professionals consisting of staff from the consortium partners.
- Section 5 assesses the results corresponding to the two types of experiments conducted by both students and professionals.

The last chapter presents the conclusions.

2 Methodology validation based on test cases

This section covers the test case definition for the DECODER software carried out by TEC in the context of task 5.4 of WP5. The validation focus was on the basic usability of the DECODER software and on the integration of the different tools via the DECODER web-frontend for artefact analysis. Additionally, possible obstacles, which a novice user might encounter were addressed in the software validation of the demonstrator. Therefore, some of the most relevant aspects included:

- Excerpt files from Reference Application MyThaiStar (java).
- Reference Application vector (c-code)
- Navigation through artefacts (code, annotations, comments, models, log files)
- How are the analysis tools applied to the different source code and specification artefacts?
- How are analysis results documented and may be tracked by the corresponding tools and processes?

2.1 Definition of test cases

The definition of test cases is split into three blocks to cover different roles: (i) Administrator, (ii) Developer and Maintainer, and (iii) Reviewer.

2.1.1 Administrator

This section holds the test cases for the role Administrator.

2.1.1.1 Create user

Summary	Creates new user.
Precondition	User is logged in as admin.

Steps	**Input Data**	**Result**
1) Open user page		List of current users is displayed
2) Select *new user* and enter new username, password and full name.	Username: testuserA Password: Full Name: Test User	User is added to the list of users
3) Login from another browser with user credentials from previous step	Name: testuserA Password:	Login with user credential is successful

2.1.1.2 Update of user information

Summary	Update user information by the Administrator.
Precondition	User is logged in as Administrator.

Steps	Input Data	Result
1) Open user page via user menu		User page is displayed
2) Update “Organisation”, email address	Email address Organisation	Email and organisation items are updated

2.1.1.3 Create project and assign user to team-member list

Summary	Create a project and give access rights to one or more user
Precondition	User is logged in as admin

Steps	Input Data	Result
1) Select “Home” page. The project page with all available projects is shown to the admin		List of all projects is displayed
2) Select Action “New Project”	Project name = testproject	Project will be created
3) Select project and switch to team-member page of the project and select “new member”		Additional entry in member list with empty name is created
4) Enter the user’s name which should have access to this project.	List of users which will get access to new project	
5) Select role of the user for this project (Developer, Maintainer, Reviewer, owner)	Role=Developer	User is assigned to project
6) Log out as admin and log in as one of the users who has access to the project		New project should show up in list of projects for the user

2.1.1.4 Remove user from team-member list of a project

Summary	Remove users from team member list of a project.
Precondition	User is logged in as admin.

Steps	Input Data	Result
1) Open project page of selected project		
2) Go to team member list of the project		
3) Select delete from team member to be removed	Team member username	Team member is removed from list

2.1.1.5 Delete project

Summary	Delete a project and verify that all related artefacts are also removed
Precondition	User is logged in as admin. Test-project is created

Steps	Input Data	Result
1) Go to project home		
2) Open project page for project which needs to be deleted	Test-Project	Project view is displayed
3) Select “delete project” on the project page		Project is deleted

2.1.2 Developer and Maintainer

This section holds the test cases for the Developer and Maintainer roles.

2.1.2.1 User login under Developer profile

Summary	User login as Developer. User has access to project and related tools.
Precondition	User has been created. User is part of the member list of the “Test-Project”.

Steps	Input Data	Result
1) Open Login dialog and enter user & password and push login	Username=”testuser”	Profile dialog opens up

	Password=test password	
2) Select Profile “Developer” and push connect	Profile=Developer	Project page shows up listing all projects where user is entered in team member list

2.1.2.2 Change user’s own password

Summary	User changes her own password.
Precondition	User is logged in as Developer under username “testuser”. “testuser” has been created.

Steps	**Input Data**	**Result**
1) Open user profile (*Username* under user icon)		User property pages open
2) Select *change password*		Dialog for password change opens
3) Enter new password and identical password under *password confirmation.* Push *change password*	Test-password	User page is displayed
4) Log out from DECODER and log in with new password. (see User Login)	Test-password	Login with new password possible

2.1.2.3 Import source documents, artefacts

Summary	Developer imports one or more artefacts (source code, documents).
Precondition	Test-Project is created. Test user is in members list of the project. User is logged in with profile Developer.

Steps	Input Data	Result
1) Select Project (home) page and select project where source needs to be uploaded		Project screen with list of accessible projects is shown For selected project upload page is provided
2) Browse for source file to be uploaded or drag and drop file onto upload page *Note: for java source the full path has to be added to the file name manually*	Source: file: *BookingmanagementRestService.java* on disk Full path: *java/mtsj/api/src/main/java/com/devonfw/application/mtsj/bookingmanagement/service/api/rest/*	Selected file is listed for upload
3) Push upload		File is uploaded. Pop-up message is displayed
4) Go to project navigation page and check if file was uploaded		Source file is listed under “Code” node

2.1.2.4 Delete an artefact

Summary	Developer removes an artefact.
Precondition	Test-Project is created. Test user is in member list of the project. User is logged in as Developer.

Steps	Input Data	Result
1) Go to the project		
2) Open an artefact		Artefact is displayed
3) Select “Delete”		Artefact is deleted
4) A dialog will show up with the name of the artefact to be deleted and the possibility to delete/cancel deletion	Delete	Artefacts will be deleted and display is updated

2.1.2.5 Apply JavaParser tools

Summary	Applies the JavaParser Tools to JAVA source code. The result will be a list of comments and a list of JML annotations extracted from the source artefact.
Precondition	Test-Project is created. Test user is in member list of the project. User is logged in as Developer. Source file (BookingmanagementRestService.java) is uploaded.

Steps	Input Data	Result
1) Select project process on Navigator page.		Available tools will show up
2) Under available tools select tool *Java parser.* Under Artefact selector select file to be parsed.	UsermanagementRestService.java	
3) Select ***Run tools***		Under tool result a return message from PKM tools is displayed once the processing is finished. Output: javaParser called with java/mtsj/api/src/main/java/com/devonfw/application/mtsj/usermanagement/service/api/rest/UsermanagementRestService.java -> {"answers":["","",""]}
4) Check log files entries		1: "AST EXTRACTOR success!" 2: "JML2JSON success!" 3: "Uploaded New RawSourceCode Normalized Version" 4: "Uploaded AST" 5: "Uploaded Comments" 6: "Uploaded Annotations"
5) Check Results: Go back to Navigation and under node “Comments” select file BookingmanagementRestService.java	file= BookingmanagementRestService.java	Display of all comments from the source file

6) Check Results: Go back to Navigation and under node Annotations select file	file= BookingmanagementRestService.java	Display of all annotations from source file

2.1.2.6 Apply openJML tool

Summary	Developer applies openJML tool to java source files. The tool carries out checks on Java language, JML assertions, JML assumptions. Results will be stored in the log collection.
Precondition	User is logged in as Developer. Source files to be checked are uploaded (BookingmanagementRestService.java). Note: Source needs full file path in file name in order that openJML will work correctly.

Steps	Input Data	Result
1) Select project process on Navigator page.		Available tools will show up
2) Under available tools select tool open*JML* and select file to be analysed in Artefact Selector.	source file: BookingmanagementRestService.java	
3) Select *Run tools*		In *Tool results* a return message from tools is displayed once the processing is completed. Output: OpenJML called with java/mtsj/api/src/main/java/com/devonfw/application/mtsj/bookingmanagement/service/api/rest/BookingmanagementRestService.java -> {"answer":"","numberOfErrors":0,"numberOfWarnings":0}
4) Check Results: Go back to Navigation	source file: BookingmanagementRestService.java	The output from openJML is reported in the log file. 1: "OpenJML Analysis Success!"

and under node Logs select file.		2: "OpenJML Results Parsed Success!"

2.1.2.7 Apply JMLGem tool

Summary	JMLGen takes as input a complete java project, and generates simple JML assertions in java source code.
Precondition	Project is created and user is in member list of the project. User is logged in as Developer. Source file (BookingmanagementRestService2.java) uploaded (source without annotation).

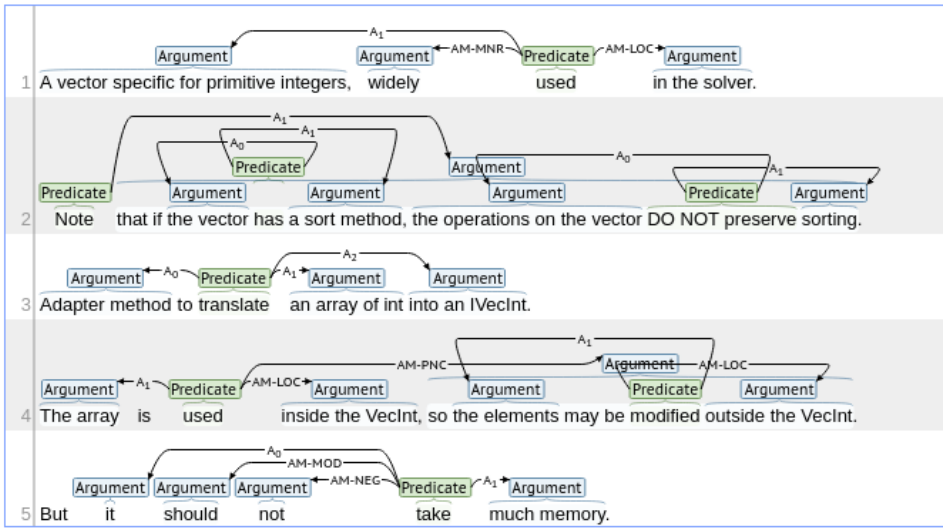
Steps	**Input Data**	**Result**
1) Select project process on Navigator page		Available tools will show up
2) Under available tools select tool *JMLGen. Note: file selection is not necessary. JMlGen will be applied to all java source files under node Code*		
3) Select *Run tools*		*Under Tool results* a return message from JMLGen tool is displayed once the processing is completed
4) Check Results: Go back to Navigation and under node *Code* select file	source file: BookingmanagementRestService2.java	JML annotations have been inserted into source file

2.1.2.8 Apply Code Summarisation tool (java)

Summary	Developer applies code summarisation to java source file. Creating an understandable description from source code.
Precondition	Project is created and user is in member list of the project. User is logged in as Developer. Source artefact (BookingmanagementRestService.java) uploaded.

Steps	Input Data	Result
1) Select project process on Navigator page.		Available tools will show up
2) Under available tools select tool *Code Summarisation* and select file to be analysed in the artefact selector.	Source file: BookingmanagementRestService.java	
3) Select *Run tools*		In *Tool results* a return message from tools is displayed once the processing is completed
4) Check Results: Go back to Navigation and under node *Code* select file.	file= BookingmanagementRestService.java	Summary has been added as comment at the beginning of each of the methods

2.1.2.9 Apply Semantic Role Labelling (SRL) tool

Summary	The SRL tool, when interacting with the PKM, takes as parameters the information necessary to find in the PKM a text segment. Its return value is only a pair of a string and a code indicating the success of the operation or explaining its failure. The actual result is an annotation added to the Annotations collection of the PKM. The illustration below shows a graphical representation of the kind of annotations that are added: 

Precondition	Project is created and user is in member list of the project User is logged in as Developer. Relevant Source file is uploaded

Steps	Input Data	Result
1) User selects artefacts to be analysed		
2) Under artefact selector user changes to annotation/comment artefacts. The related comments and annotations will be provided. User selects a comment or annotation as text segment.	Text segment	
3) User selects tools for SRL		*Under Tool results* a return message from SRL tool is displayed once the processing is completed
4) Open related log files and check results from tool		Log file will hold selected texts and list of predicates found

2.1.2.10 Apply Named Entity Recognition (NER) tool

Summary	The NER tool, when interacting with the PKM, takes as parameters the information necessary to find in the PKM a text segment. Its return value is only a pair of a string and a code indicating the success of the operation or explaining its failure (it will be later completed with the id of the artefact added to the PKM annotations collection). The actual result is an annotation added to the Annotations collection of the PKM. This annotation lists the occurrences of entities (persons, dates, numbers but also class names, identifiers, OS names, etc.) found in the targeted text.
Precondition	Project is created and user is in member list of the project. User is logged in as Developer. Relevant artefacts are uploaded.

Steps	Input Data	Result
1) Select artefacts to be analysed		

2) Under artefact selector user changes to tab “annotation/comment artefacts”. The related comments and annotation will be listed. User selects from comment or annotation a text segment.	Text segment “REST SERVICE METHODS”	Text segment is selected.
3) Select tools for NER		*Under Tool results* a return message from NER tool is displayed once the processing is completed
4) Change to relevant log file to check for result. (List of named entities in json format)		Results show up in the log file. "selected texts are: ['REST SERVICE METHODS',]" 4: "Analyzing REST SERVICE METHODS." 5: "NER completed True, {'entities': [{'text': 'METHODS', 'type': 'Identifier', 'pos…"

2.1.2.11 Apply Semantic Parsing (SP) tool

Summary	The SP tool, when interacting with the PKM, takes as parameters the information necessary to find in the PKM a text segment. Its return value is only a pair of a string and a code indicating the success of the operation or explaining its failure. It automatically generates source code according to the information provided by the annotations and comments.
Precondition	Test-project is created and user is in member list of the project. User is logged in as Developer.

Steps	**Input Data**	**Result**
1) Under artefact selector user changes to tab “annotation/comment artefacts”. The related comments and annotations will be listed. User selects from comment or annotation a text segment.	Text segment	
2) User selects tools for Semantic Parsing (SP)		
3) Change to relevant log file to check for result.		The comments of the functions are

		completed with the generated code

2.1.2.12 Apply FramaC tool

Summary	Apply tool Frama-C to C Source Code.
Precondition	User is logged in in as Developer. Source file is uploaded.

Steps	Input Data	Result
1) Select project process on Navigator page.		Available tools will show up
2) Under available tools select tool *FramaC* and select source file to be parsed in Artefact Selector.	source file: vector.c	
3) Select *Run tools*		In *Tool results* a return message is displayed once the processing is completed
4) Check Results: Go back to Navigation and under node Annotation, Logs and Comments select file.		List of annotations, comments, logs are shown under respective node annotation.

2.1.3 Reviewer

This chapter holds the test cases for Reviewer role.

2.1.3.1 Testar integration (View StateModel and TestResult)

Summary	TESTAR creates two Artefacts at the end of its execution: 'ArtefactTestResults' and 'ArtefactStateModel'. PKM will validate both Artefacts with the PKM-API schemas (https://gitlab.ow2.org/decoder/pkm-api/-/tree/master/schemas) and will return both Artefact-Ids if everything was successfully executed. This test case only covers the display of imported Testar result in DECODER.
Precondition	Results from Testar (TestResult, StateModel) are imported into PKM. User is logged in as Reviewer.

Steps	Input Data	Result
1) Open Navigation and open under node “Testar_State_model” Testar StateModel	Name of Testar State Model	list of available Models is displayed.
2) Select dropdown icon next to Model Id		List of tests for this model will be displayed
3) Double Click on view icon		The Results from Testar for this execution will be shown.
4)Select layer icon		Option Dialog to select layer will be shown
5) Select “Abstract Layer”	“Abstract Layer”	State model for the layer will be displayed

2.1.3.2 Jupyter integration (interface test)

Summary	Interface Jupyter to PKM Test the interface Jupyter GUI integration into the DECODER front, which allows utilising Jupyter via the DECODER front end and also allows Jupyter notebook to access PKM.
Precondition	Jupyter installed. Jupyter Notebook (PKM) available.

Steps	Input Data	Result
1) Select Jupyter icon in Tool bar.		Jupyter screen will show up embedded in decoder window
2) Import: Jupyter notebook mongotest.ipynb and pymongo_Conf.json	mongotest.ipynb Pymong_conf.json	Python script is executed without errors
3) Check read access to PKM by executing mongotest.py (1. Cell of notebook)		

4) Check write access to PKM by executing mongotest.py (3. Cell of notebook)		

2.1.3.3 Apply doc_to_asfm tool

Summary	It translates a structured Developer's documentation into a json document. This json document only contains information that the other tools can understand. Its syntax follows the Abstract Semi Formal Modelling (ASFM) language and can be sent directly to the Persistent Knowledge Monitor.
Precondition	User logged in as Reviewer. Word document already imported to PKM.

Description and Steps	Input Data & Precondition	Result
1) Select word document via artefact selection	Word doc artefact	
2) Select tool doc_to_asfm via tool selector		
3) Select *Run tool*		New json file will be created holding the word document structure in the ASFM language format

2.2 Test execution and test environment

Test cases were executed multiple times, whenever new, updated functionality or fixes were available.

The test environment was based on the following deployment:

- ***https://decoder-frontend.ow2.org/*** (Version from End of March 2021 frontend and backend).
- Test with Chrome Version 89.0.4389.90 (x86_64).
- A separate test project was created holding the artefacts to be processes/analysed.

2.3 Validation of results

2.3.1 Classification of findings

The test results were classified by execution status:

- PASS: Test was passed successfully.

- FAIL: Test failed, and an issue was created to resolve the error.

Finding during the test would be characterised as follows:

- (ER) – Error: Error was encountered during a test of this case
- (UX) – User Experience Improvement: Function basically works, test case was set to passed. Still a finding, documenting the shortcoming in the user experience (UX), has been created.
- (RI) – Recommendation for improvement: Function which was addressed by a test case was not available at point of time of the test. Or while testing/debugging the system a function was missing.

2.3.2 Errors

Over multiple test-runs few errors have been encountered and have been directly fed back to the respective development team. Fixes have been provided in the following deployments, which went through regression testing.

2.3.3 UX improvements

The UX Improvements in Table 1 were documented during the validation test. UX issues have been successfully resolved with the demonstrator version available at the end of the project.

Table 1: List of UX improvements.

ID	Description
UX-1	Navigation If a source file has a long name (including path). The name is not shown in the navigation page and also not in the tab header of the source code renderer. Name should be truncated either in the middle. Or whole name including path should be shown in a tool tip with mouseover.
UX-2	Navigation It is not possible to open related file/comments/annotations from list of related artefacts.
UX-3	Tool processing No progress indication is shown when a tool is started and when it is completed. Due to that the user might start the same tool twice.
UX-4	Tool processing When the tool is started “tool result” tab should open automatically to see the status feedback from the tool immediately once completed.
UX-5	Tool processing general *Run tool* button should be disabled in case no artefact or not tool is selected.

ID	Description
UX-6	Navigation - Display of Related Artefacts: Related artefacts tab should also list log files, which are related to the selected code artefact.
UX-7	Annotation's renderer If annotation only contains one line, Display the line number once in the heading of the annotation. E.g “Line. 3 – 3” should be “Line 3”
UX-8	When selecting a log file in navigation page also show the related artefacts in the related artefacts page.
UX-9	Navigation page: Search function does not apply to log folder. Function applies only to code, annotation, comment folder. Search should also consider entries in log folder.
UX-10	User Profile: Once user profile page is opened, there is no navigation back to the Project/Navigation page.
UX-11	In the artefact selection also entries for comment and annotations show up.
UX-12	When a source file is changed (e.g., by code Summarisation) the related comments and annotations are not updated and thus are incorrect. In this case the related comments and annotation should be deleted.
UX-13	View Testar results. User interface very different to other DECODER screens. It is not obvious: how to get list of test results How to view test results how to display state model
UX-14	Artefact Navigator: Artefacts are listed in order of the mongo-db-id. Listing should be by artefact name. This is also valid for the elements under nodes “Comments” and “Annotation”
UX-15	Artefact Navigator: If the number of artefacts (code, comment, annotation, logs) increases it is not possible to scroll down to the last entry of the tree.
UX-16	Artefact Navigation/Artefact rendering:

ID	Description
	The field for the artefact name in the heading of editor is too short. Full name should be shown in a tool tip with mouse over.

3 Experiments with students

3.1 Goal and context

DECODER has proposed a methodology for improving the productivity of the software development process for medium-criticality applications. The methodology has been complemented with a software tool that aims at helping software developers in the application of this methodology. The goal of the experiment was to evaluate the tool support for the DECODER methodology. Specifically, we performed user acceptance testing with students of software engineering as end-users. The user acceptance testing allows us to determine whether the DECODER tool can be accepted or not by software developers.

3.2 Environment

The experiment was performed in the Universitat Politècnica de València (UPV) and in the Universitat de València (UV) in December 2021. The participants were students enrolled in the course "User Centered Design" from the UPV and students enrolled in the course "Ingeniería del Software II" from the UV. The conductors of the experiment were a teacher from the UPV and a teacher from the UV, both participants in the DECODER project. The experiment was performed during a session of the mentioned courses.

3.3 Script and materials

The procedure and the materials involved in the experiment were the following:

1. Introduction: the conductors present the experiment.
2. Students answer the demographic questionnaire.
3. Students download the zip file "*bankexample*" from the platform indicated by the conductor. This zip contained several standard documents generated at the beginning of a software project. In this case, the project was related to bank accounts. The students must check that the zip contains:
   - 1.i a .docx file with the project requirements
   - 1.ii a .uml file with the class diagram specification
   - 1.iii four .java files (one of them with JML annotations)
4. Tool testing. Each student must:
   - 1.i access the tool: https://decoder-frontend.ow2.org/login
   - 1.ii perform the tasks specified in the next section
5. Students answer the user acceptance form

The *demographic questionnaire*[1] included the following eight questions:

[1] https://docs.google.com/forms/d/e/1FAIpQLSdtiDahd6zR2RrhAEK6Dv6_ECw9DAZlNMOGtbRMS0FujupM0w/viewform?usp=sf_link

Profile:

1 State your gender
   - Male
   - Female

2 State your age
   - 16-25
   - 16-40
   - 41-55
   - Above 55

3 Which description matches best your current work status?
   - Academic
   - Professional

4 Which is your highest academic degree?
   - Engineer
   - Master
   - PhD

Experience

5 How many software development projects have you been involved in?
   - Below 5
   - 5-10
   - 11-20
   - Above 20

6 How many years ago did you participate in your first project?
   - Below 3
   - 3-5
   - 6-10
   - Above 10

7 I am very familiar with software development methods such as RUP, XP, Scrum, V-Model, ...

1	2	3	4	5
Strongly agree				*Strongly disagree*

8 I am very familiar with CASE environments such as Power Designer, MetaEdit+, ...

1	2	3	4	5
Strongly agree				*Strongly disagree*

The *user acceptance form*[2] was composed of the following twelve questions that were answered using the Likert-scale (which allowed participants to express how much they agree or disagree with each particular statement):

Perceived Usefulness

1 The DECODER tool allows me to complete tasks more quickly
2 The DECODER tool improves my job performance
3 The DECODER tool improves my productivity
4 The DECODER tool enhances my effectiveness on the job
5 The DECODER tool makes it easier to do my job
6 Overall, I find the DECODER tool useful in my job

Perceived Ease of Use

1 I find the DECODER tool easy to learn
2 I do not need to consult the user manual when I use the DECODER tool
3 My interaction with the DECODER tool is easy for me to understand
4 The DECODER tool is flexible to interact with
5 I find it easy to become skilful when I use the DECODER tool
6 Overall, I find the DECODER tool easy to use

3.4 Evaluation

Once the students accessed the tool, they must perform the following tasks:

1 Log in. The conductors provided each student with a username and password.
2 Select the Developer profile.
3 Access the project (there was just one project created).
4 Upload the .docx file that contained the project requirements. Indicate that the file is of the type *document.* Check that the document has been uploaded.
5 Upload the .uml file and specify that it is of the type *diagram.* Check that the document has been uploaded.
6 Upload the .java files. To do so, for each document:
    6.a Drag the java files to the frame.
    6.b Add the path: src/main/java/core/ before the name of the java document in the rename file field.
    6.c Specify the type of document as *code.*

    Check that the document has been uploaded.

[2] https://docs.google.com/forms/d/e/1FAIpQLScaI9VQNzTMMHqZplL2atVQj8vh4PScl1aBzfpU4W0yPNnSKg/viewform?usp=sf_link

7 Visualise the UML class diagram. To do so:

7.a Transform the .uml file to a mermaid diagram specification using the *Class Model XMI To JSON transformer* tool.

7.b Open the diagram generated by the tool that is located in the UML diagram folder.

8 Generate the annotations for the source code artefacts that do not have them using the JMLGen tool.

9 Check the code to review the generated annotations.

10 Generate the AST, comments, and annotations files for the code using the JavaParser tool. Select the *all* option for the execution mode.

11 Check that the following artefacts have been generated:

11.a Comments (in the Comment folder).

11.b Annotations (JML) (in the Annotation folder).

12 Analyse the JML annotations for the four source code files using the OpenJML Analyzer. Select the *typechecking* option for the execution mode.

13 Check the results of the JML analysis. In the results tab, look for the OpenJML Analyzer invocation. Open it and go to the logs tab.

14 Modify the code of one of the java classes by adding a comment.

15 Generate again the comments file for the modified code using the JavaParser tool (similar to step 10).

16 Check the new comments file.

4 Experiments with professionals

4.1 Goal and context

This experiment pursues the same goal of methodology validation in the terms described in section 19, i.e., “evaluate the tool support for DECODER methodology”. However, this section focuses on experiments to evaluate user acceptance conducted by professionals recruited by the different consortium partners (i.e., TEC, CEA, CAPGEMINI, OW2, SYSGO, TREE, UPV) within their staff, bringing together different profiles that will help us to draw conclusions on the acceptance by the end users.

4.2 Environment

The experiment was carried out at each of the partner’s facilities and/or remotely due to the pandemic. To this end, TREE generated a set of five credentials (username and password) for each consortium partner and an associated project within the DECODER environment. The recruitment process was organised internally by the partners, who also distributed the credentials among the volunteers.

A period of two weeks was fixed to conduct the experiments – estimation of less than half an hour per experiment.

4.3 Script and materials

Professionals received the same documentation and materials as the students did (section 19). However, the script was extended with two additional tools (variable misuse and code summarisation) to also include them in the evaluation framework. Furthermore, additional details about some steps were provided to minimise the doubts that could be raised – especially since a person to provide support timely had not been appointed.

Volunteers should also respond to the same questionnaire generated by UPV for the experiment with the students, so the feedback can be compared between the two target groups.

4.4 Evaluation

The script shared with the volunteers was:

1 Log in using one of the usernames and passwords belonging to your organisation that has not been used already from the spreadsheet.

   1.a username: <check usernames in spreadsheet>

   1.b password: <check corresponding password in spreadsheet>

2 Select the Developer profile.

3 Access the project (there is just one project created per user).

4 Upload the .docx file that contains the project requirements. Indicate that the file is of the type *document*. Check that the document has been uploaded.

5 Upload the .uml file and specify that is of the type *diagram*. Check that the document has been uploaded.

6 Upload the .java files. To do so, for each document:

6.a Drag the java files to the frame.

6.b Add the path: src/main/java/core/ (without the spaces before and after) before the name of the java document in the rename file field.

6.c Specify the type of document as *code.*

7 Visualise the uml class diagram. To do so:

7.a Click on the name of the project under *Project versions.*

7.b Transform the .uml file to a mermaid diagram specification using the *Class Model XMI To JSON transformer* tool. For this, click on *Tools* on the left bar and then choose Low Level Design *(LLD) – Module Design.*

7.c Select the file *core.uml* and click on *Execute.*

7.d Go to *Navigation* and open the diagram generated by the tool that is located in the UML diagram folder.

8 Generate the annotations for the source code artifacts that do not have them using the JML Generator tool. For this, click on *Tools* on the left bar and then choose High Level Design *(HLD) – System Design*.

9 Go to *Navigation* and check the code to review the generated annotations.

10 Generate the AST, comments and annotations files for the code using the JavaParser tool. For this, click on *Tools* on the left bar and then choose Low Level Design *(LLD) – System Implementation*. Select the *all option* for the execution mode.

11 Go to *Navigation* and check that the following artefacts have been generated:

11.a Comments (in the *Comment* folder).

11.b Annotations (JML) (in the *Annotation* folder).

12 Analyse the JML annotations for the four source code files using the OpenJML Analyzer. For this, click on *Tools* on the left bar and then choose Low Level Verification *(LLV) – Unit testing*. Select the *typechecking* option for the execution mode.

13 Check the results of the JML analysis. In the *Results* tab on the left bar, look for the OpenJML Analyzer invocation. Open it (click on the eye on the right) and go to the *Logs* tab.

14 Modify the code of one of the java classes by adding a comment. For this, go to *Navigation*, select one Java source code file and edit it directly. Then, re-generate the

AST, comments and annotations files for the code using the JavaParser tool (same than step 10) and check the new comment in the *Comment* folder.

15 Run the code-summarisation tool for all the files of the project (*Code Summarization for Project*). For this, click on *Tools* on the left bar and then choose Low Level Design *(LLD) – System implementation.*

16 Go to *Navigation* and check whether the methods in the files now include a new comment with an automatically generated summary of the method (you can search for DECODER-CODE-SUMMARIZATION in the page). Otherwise, no bug has been found.

17 Re-generate the AST, comments and annotations files for the code using the JavaParser tool (same than step 10).

18 Run the *Variable Misuse for Project* tool. For this, click on *Tools* on the left bar and then choose Low Level Design *(LLD) – System implementation.*

19 Go to *Navigation* and look for some suggestion of possible variable misuse instance in the code (you can search for DECODER-VARIABLE_MISUSE in the page).

20 If desired, feel free to delete the project files and upload or write other Java, C or C++ to experiment with the tools that are included in the project. Note that the files and folder structure need to be complete to allow the use of the parsers (JavaParser, FramaC and FramaClang) and advanced tools and special compilation commands might need to be configured in complex cases.

5 Feedback assessment

5.1 Experiments with students

The user acceptance questionnaire (introduced in section 19) was used to determine whether the DECOCER tool can be accepted or not by software developers. As mentioned, the questionnaire consists of 12 questions, grouped in two sections: the perceived usefulness and the perceived ease of use.

Perceived ease of use refers to the degree to which an interaction designer believes that using our method would be effort-free.

Perceived usefulness refers to the degree to which an interaction designer believes that our method will achieve her/his intended objectives.

This questionnaire contains Likert-scale values ranging from 1 (strongly agree) to 5 (strongly disagree) to evaluate satisfaction with the entire process.

The user acceptance questionnaire revealed the following impressions.

5.1.1 Perceived usefulness

A high percentage of participants (50.9%) feel (Figure 1) that the DECODER tool allows them to complete tasks more quickly (strongly agree and agree). In the face of 23.6% of participants that do not agree with this perception (strongly disagree and disagree).

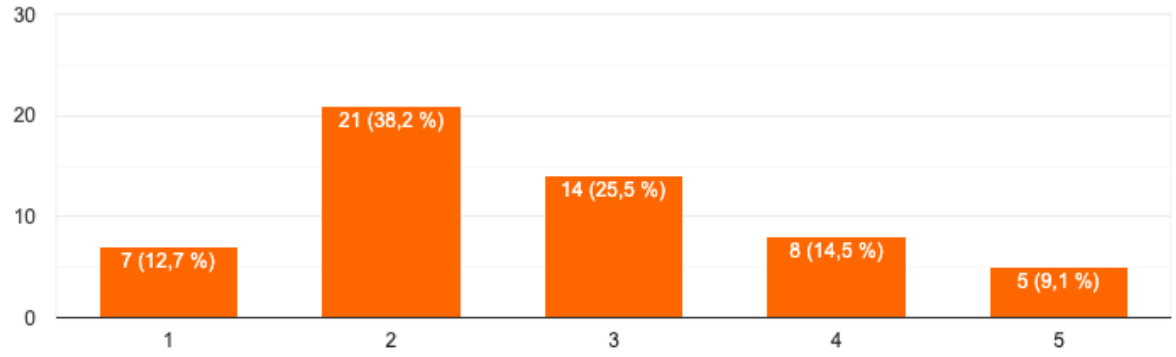


Figure 1: Students' perceived usefulness - Faster task completion

A high percentage of participants (48.2%) (Figure 2) feel that the DECODER tool improves their productivity (agree or strongly agree). In the face of 26% of participants that do not agree with this perception (disagree or strongly disagree).

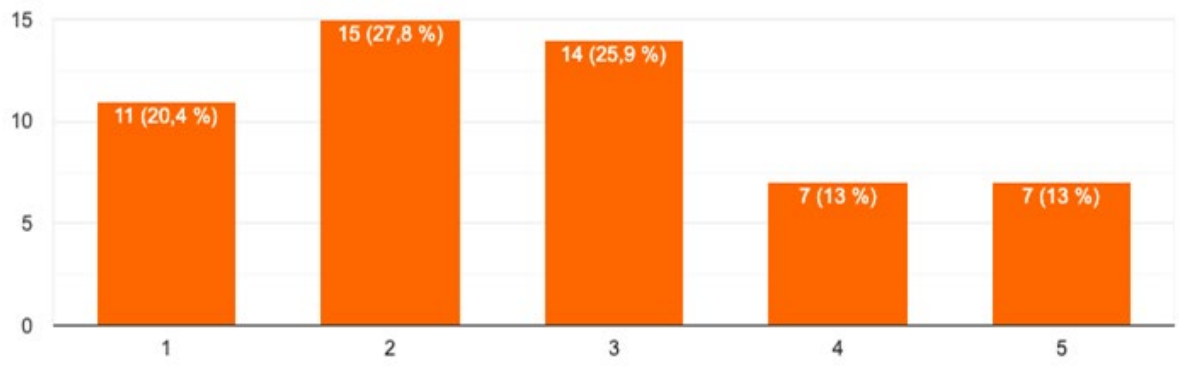


Figure 2: Students' perceived usefulness - Productivity improvement

Overall, a high percentage of participants (45.5%) find (Figure 3) the DECODER tool useful for their job (agree or strongly agree). In the face of 27.2% of participants that do not agree with this perception (disagree or strongly disagree).

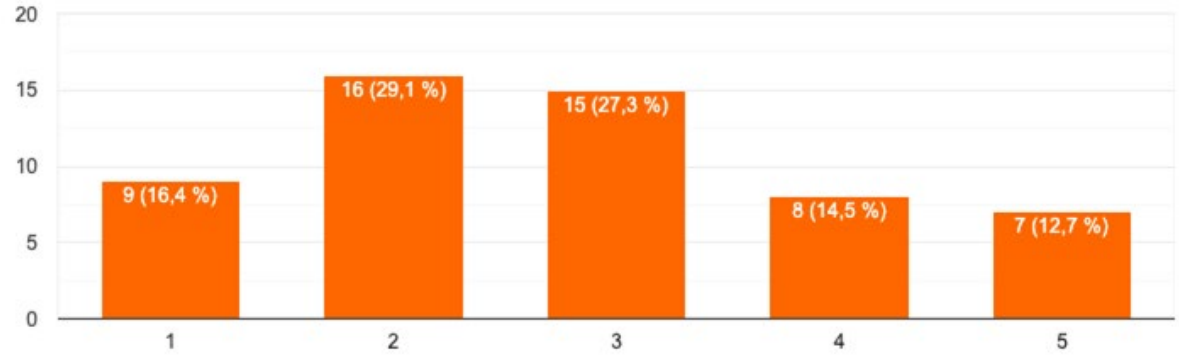


Figure 3: Students' perceived usefulness - Usefulness

5.1.2 Perceived ease of use

Most participants (47.2%) find (Figure 4) the DECODER tool not easy to learn (disagree or strongly disagree). In the face of 34.2% of participants that find it easy to learn.

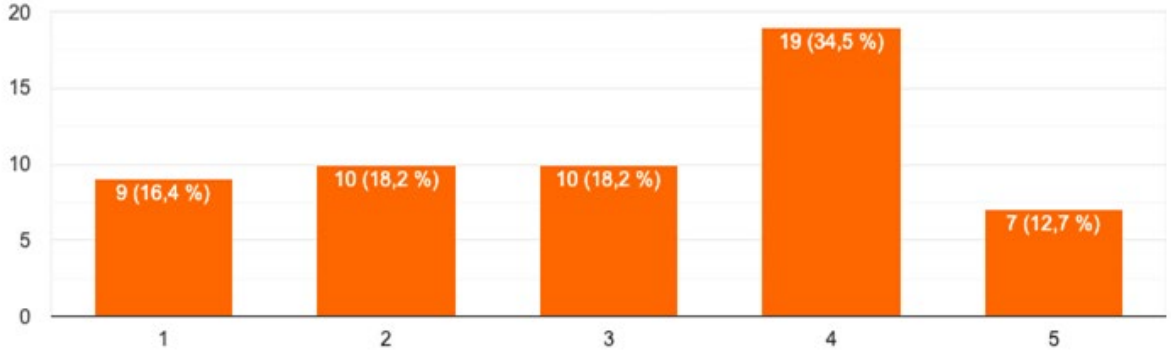


Figure 4: Students' perceived ease of use - Easy to learn

In Figure 5, most participants (40.8%) find the interaction with the DECODER tool not easy for them to understand (disagree or strongly disagree). In the face of 31.5% of participants that find it easy to understand (agree or strongly agree).

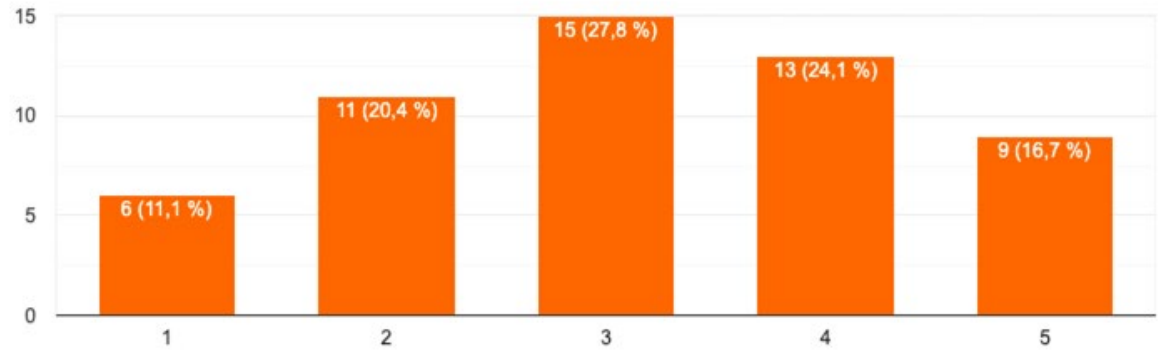


Figure 5: Students' perceived ease of use - Easy to understand

Overall, 32.7% of participants find the DECODER tool not easy to use (disagree or strongly disagree) and 36.4% of participants find it easy to use (agree or strongly agree) as shown in Figure 6.

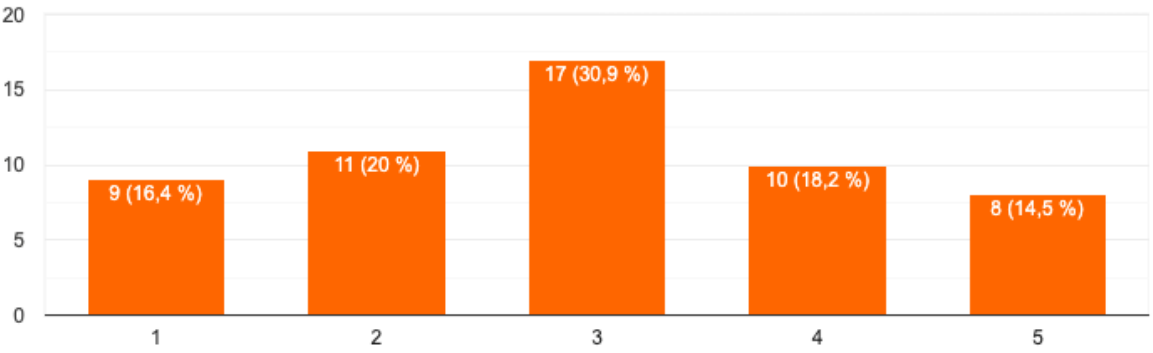


Figure 6: Students' perceived ease of use - Easy to use

Therefore, the experiment reveals that even though participants find DECODER tool a useful tool that would allow improving their jobs, it is not easy to learn and understand, and the interaction with the tool should be improved.

5.2 Experiments with professionals

As commented in section 23 the same questionnaires are used for the professionals to evaluate the DECODER environment and provide feedback.

A group of 24 professionals has taken part in the experiments with an age distribution that is presented in Figure 7(left). While this aspect may not be relevant for the group of students, we can see that the group made of professionals covers all the ranges, with 58.3% between 26 and 40 years.

Likewise, the highest academic degree in Figure 7(right) shows that half of them hold a PhD degree but only 4.1% have completed high school as highest degree. This is mainly due to the profile of the consortium partners and their staff. It would be good to have a more balanced distribution as DECODER is conceived for the roles of Developer, Maintainer and Reviewer, which can be adopted by different professional profiles.

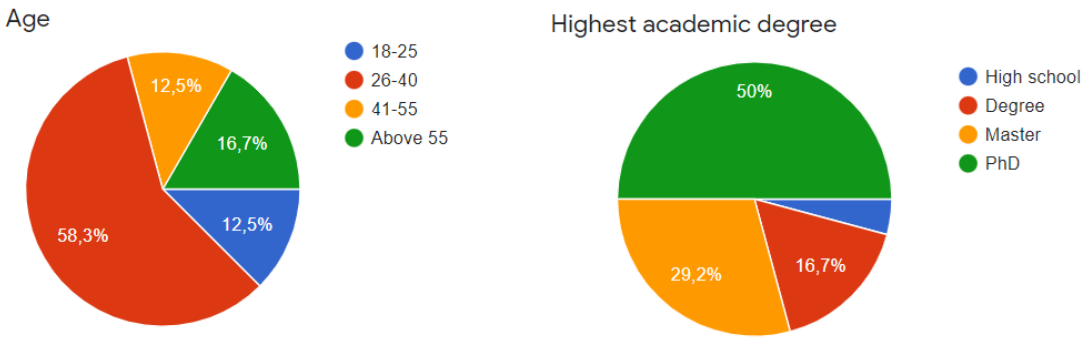


Figure 7: Age and academic degree of professionals that took part in the experiments

Finally, 83.3% of the volunteers are developers and 16.7% are reviewers. Unfortunately, the role of Maintainer is missing within the group.

5.2.1 Perceived usefulness

Although 24 professionals answered the demographic questionnaire, one of them did not complete the experiment (or forgot to fill in the second questionnaire). Therefore, only 23 are being considered in the following analysis.

As shown in Figure 8, almost 50% (47.8%) of the volunteers perceive DECODER (agree or strongly agree) as a solution that will allow a faster completion of the tasks. 26.1% are neutral and the remaining 26% disagree or strongly disagree.

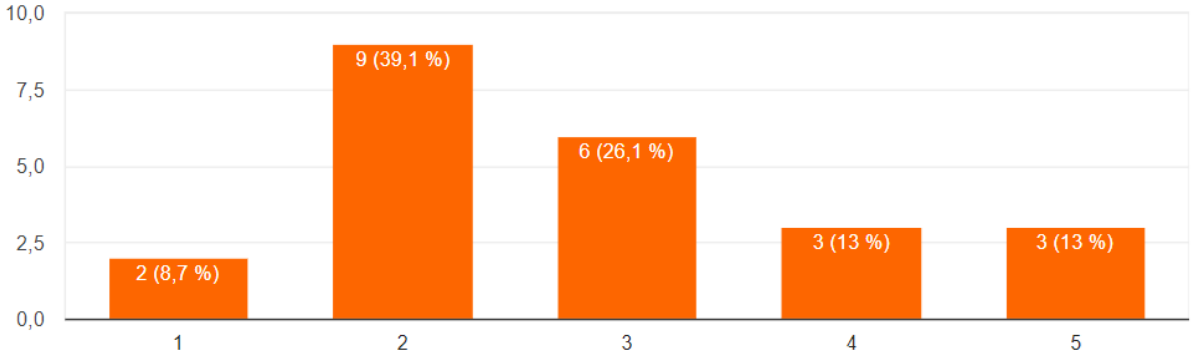


Figure 8: Professionals' perceived usefulness - Faster task completion

In terms of productivity improvement (Figure 9), 47.8% agree or strongly agree on the value of DECODER in their daily work. Besides, 8 of the 23 professionals do not find relevant what the environment can provide and only 17.6% disagree or strongly disagree with the assertion

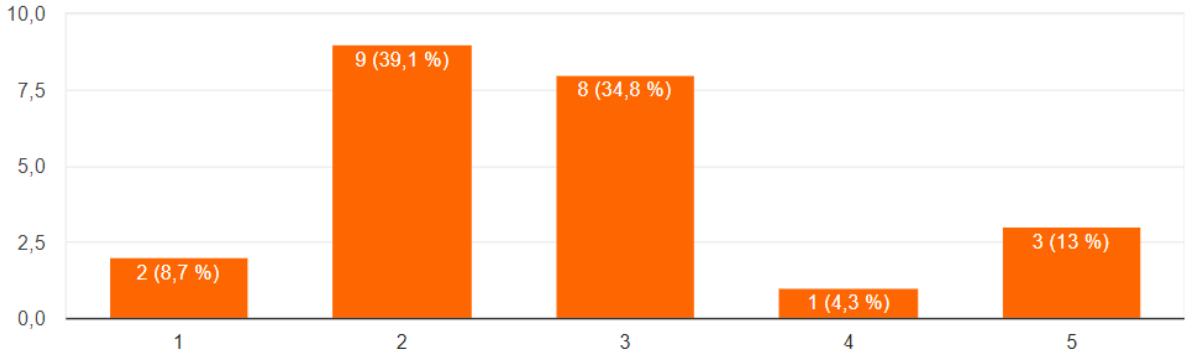


Figure 9: Professionals' perceived usefulness - Productivity improvement

Finally, in relation to the usefulness (Figure 10), 43.5% think that DECODER is useful (agree or strongly agree) for their daily work in comparison to 21.7%, who disagree or strongly disagree.

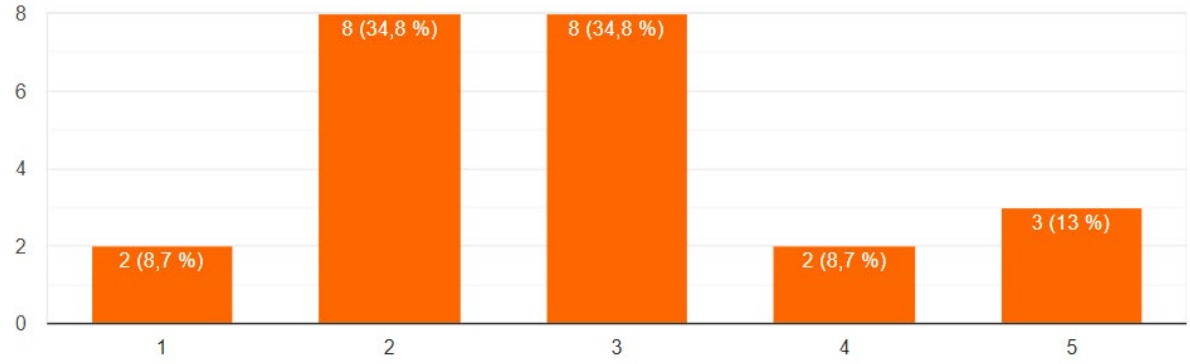


Figure 10: Professionals' perceived usefulness - Usefulness

5.2.2 Perceived ease of use

In terms of the perceived ease of use, the professionals that took part in the experiments may be conditioned by the fact that most of them were already familiar with the environment. Improvements in this field have been suggested in the biweekly calls addressing integration of the different tools and development of the process engine and graphical interface. This is the reason why the figures are not presented in this section and will stick to the feedback provided by the students in section 27.

6 Conclusions

This report has presented the approach to validate the DECODER methodology following a strategy consisting of an initial definition of test cases and lab validation and two experiments involving groups of both students and professionals.

The test cases covered four different roles of the user within the DECODER environment, i.e., Administrator, Developer, Maintainer and Reviewer. They also addressed the most relevant operations and the tools developed in other work packages. Errors were identified and fixed by the technical partners while recommendations for UX improvement were considered for the evolution of the GUI and PE, now integrated in their final versions.

The experiments were organised in two different blocks to get feedback from different profiles. Students from UPV and UV were recruited to test the DECODER methodology through the GUI and following a script that focused on activities that should enable assessing acceptance and usability. On the other hand, volunteers from the different consortium organisations participated in a similar (but extended) experiment with the same goal.

Feedback from the volunteers is quite similar for the two target groups in terms of usability. In brief, around 50% agree or strongly agree on the relevance of DECODER to help/support some tasks. On the other side, around 20-25% disagree or strongly disagree, while the remaining percentage of volunteers keep neutral.

In relation to how easy is to use DECODER and its tools, this seems to be one of the main drawbacks for the students, because 47.2% and 40.82% think that it is not “easy to learn” and “easy to understand” respectively. This should be considered in further steps, especially in terms of adding help tips and preparing a comprehensive manual.