Master Degree in
Computer Science

CADOCS
Conversational Agent for the Detection of Community Smells

Test Plan

https://github.com/gianwario/csDetector

Team Members

Gianmario Voria
g.voria6@studenti.unisa.it

Antonio Della Porta
a.dellaporta26@studenti.unisa.it

Reviewer

Stefano Lambiase
slambiase@unisa.it

18th July 2022

Contents

1 Introduction

To provide managers with novel knowledge about community smells, we want to give practitioners a tool able to detect them in a really fast and easy way. Such a tool will be available in environments which most of the practitioners are already familiar with, keeping the development of an intuitive and user-friendly software our main focus. In our opinion, this will help project managers take more informed decisions during all the phases of their work. To achieve this goal, we are creating a **Conversational Agent**, named **CADOCS**, that will be invoked with natural language. This agent will rely on an already existing tool—csDetector—that is able to detect community smells on a given GitHub repository using Machine Learning. For this reason, a part of our test plan will be about the detection of possible introduced faults in the previously existing tool.

2 Testing Approach

Since we are both going to evolve and maintain an existing tool, we will have more than one approach for the testing. In particular, while maintaining, we have to ensure ourselves that new portions of code introduced have not compromised the regular execution of the tool. For this reason, we plan to execute **regression testing** to be sure that CSDETECTOR still works as expected even after our changes.

In order to do so, we will also execute **system testing** by checking if the new version of the tool works as the previous. The reasoning behind this decision is that the developer of the tool did not provide any test suite that could be used to perform regression testing. In the scope of system testing, we will perform **functional testing** for the functional requirement regarding the main responsibility of the tool, i.e. the detection of community smells.

We also plan to execute **unit testing** for the majority of the functions written during the maintenance process.

In order to define the test cases we will use the **Category Partition** technique.

2.1 Regression Testing

The original tool we are maintaining and evolving—csDetector—can detect community smells on a given repository using machine learning. It consists of a series of analysis on the source code from GitHub, computing socio-technical metrics that are used to do predictions on the ML model. It is important to note that Almarimi et al. did not provide any test suite that we can execute, and testing this kind of software from scratch would be really difficult. This problem led us to the decision of considering the regression test suite the normal execution of the tool on a certain set of repository.

For this reason, we will find a set of GitHub repositories with which we can detect every single smell of the ten that the tool is able to detect. After each pull request from a CR to the main branch—and so after we complete every single change request—we will run the tool on the previously-defined set of repositories with our additions to see if the results are still valid.

2.2 System Testing

For which concerns the original tool, while studying the system in order to analyse the impact of our changes, we found out that there were not many functional requirements, and that the tool only has one big purpose, i.e., detecting community smells, which we identified as the first functional requirement. For the detection of the previously-mentioned smells, it calculates a lot of metrics, which we may consider as another functional requirement, alongside each of the step the tool executes with the final purpose of performing inference on the ML model. But this is still not something the user interacts with, and everything is based on the input given at the first functional requirements, i.e. detecting smells. For this reason, testing the single functionality of detecting smells can be seen as testing the whole system.

Since the CR_3 involves the implementation of new system, we will perform functional testing for the most important functional requirements.

2.3 Unit testing

We will only perform unit testing on the new functionalities written in the context of the CR_1, CR_2, since the third CR is the actual implementation of a whole new tool—a conversational agent—and we had limited budget in terms of time.

2.4 Testing CR_3: Conversational Agent

As stated in the previous section, we had no time perform unit testing activities for the CR_3, which is the implementation of a conversational agent available on the Slack platform that allows the execution of csDetector through natural language. But in order to validate it in its most important improvements with respect to csDetector, i.e. the user interface and the natural language interaction, we performed **usability testing** and **input testing**.

Of course, just like any of the other contributions, we will perform **functional testing** on each of the functionalities of the chatbot. Since it is based on NLU, and the actual execution results can't be predicted due to the probabilistic behavior of the ML model, we will relax the oracles of the tests.

2.4.1 Usability testing

The first evaluation concerned the usability of the bot. In this case, we applied **iterative usability testing**. This strategy is based on an iterative process in which, at each step, users give feedback about the tool's user interface—after executing a series of tasks—and developers modify the tool accordingly. We recruited four graduated students who attended and achieved a Human Computer Interaction course during their degree. We asked them to conduct three tasks during each process iteration—the tasks are detailed in Table 15.

Table 1 Iterative Usability Testing

Q1	Add the conversational agent to your *Slack* workspace using the provided guidelines.
Q2	Ask the conversational agent to provide the social metrics of one of your repositories.
Q3	Report the identified metrics at Q2 and the mitigation strategies.

After each iteration, we directly interviewed participants to measure the bot's usability in terms of well-known metrics—e.g., *learnability*, *efficiency*, and *satisfaction*—and consequently improved the tool interface. We kept iterating the usability evaluation until reaching saturation. Five iterations were required to consider our tool usable.

2.4.2 Input testing

We assessed the robustness of the machine learning model responsible for the natural language understanding module of the agent. We performed **input testing**, i.e. an approach based on metamorphic testing that aims at identifying potential reasons for unsuccessful training in the used data. Specifically, we performed two steps: (1) we morphed parts of our training data—questions to be asked to the conversational agent—into paraphrases having a relation of evenness with the original sentence; (2) we defined our test oracle so that the test will pass if the confidence of the prediction for the paraphrase does not exceed of 0.2 the confidence for the original sentence. Results showed that our dataset was good enough to train a model able to predict users' intents consistently. Hence, we deemed our model as robust enough to handle the requests received by users.

3 Features to Test

In this section we will explain at an high level the features we will test. In the scope of each change request, we will identify the new introduced functional requirements.

Table 3 CR_1 Feature to test

CR_1: Extracting the core functionalities and isolating the I/O process
Test Item
Since the goal of the changes is to execute the tool in different ways, we will test if the new execution is equivalent to the previously-existing one. While doing so, the testing process will cover every new introduced function.
Tested Features
• **CR_1-CRS**: The system must be able to print the output in console in an improved way with respect to the original version; • **CR_1-ATE**: The system must be able to be integrated and executed through an Adapter; • **CR_1-SSC**: The system must be able to persist the results of the executions in a spreadsheet.

Table 4 CR_2 Feature to test

CR_2: Wrapping csDetector in a web service
Test Item
With the introduction of a web service, the actual execution won't be modified. For this reason the goal of testing the changes of this CR is to check whether the new way of taking input and giving output works.
Tested Features
• **CR_2-WSE**: The system must be able to handle JSON input parameters and return a JSON formatted response.

Table 5 CR_3 Feature to test

CR_3: Creating a conversational agent that will execute the tool
Test Item
This CR involves the creation of a whole new system. For this reason we will test the most important functional requirements through functional testing.
Tested Features
• **CR_3-I1**: The system must be able to execute CSDETECTOR on a given repository; • **CR_3-I2**: The system must be able to execute CSDETECTOR on a given repository starting from a specified date; • **CR_3-I3**: The system must be able to print a report of its last execution; • **CR_3-I4**: The system must be able to print the list of community smells that it can detect.

For the remaining CRs, namely *CR_4: Improving software's robustness* and *CR_5: Source code refactoring for readability concerns*, we will consider valid the **system testing** and the **regression testing**. The reason is that these changes won't modify or add any new functionality, and for this reason we will only need to check if it works in accord to the specification.

4 Pass and Fail Criteria

The test that we are going to execute has the goal of detecting faults inside the system that will eventually be corrected. Each test will have an oracle that is the expected output of a specified input. A test will be considered passed if, given a certain input of the oracle, the output provided by the system is compliant with the output described by the oracle. Otherwise, if with a given input of the oracle the output of the system does not respect the oracle, the test will be considered failed.

5 Category Partition

In this section, for each CR, we will define the test frames for the test item identified in the sections above.

5.1 CR_1

5.1.1 Test Frame for CR_1-SSC

Table 6 Category Partition CR_1-SSC.

Description	
The system must be able to persist the results of the executions in a spreadsheet	
Parameters	
config, starting_date, detected_smells	
Categories	
Category Name	**Category Value**
config	• CF1: config contains valid configuration of a repository, i.e. *url, owner* and *name* • CF2: config is not valid (passed as None type)
starting_date	• SD1: starting_date is present • SD2: starting_date is not valid (passed as None type)
detected_smells	• DS1: detected_smells is present • DS2: detected_smells is not valid (passed as None type)
Costraints	
Attribute	**Costraint**
config	• CF1: config contains valid configuration of a repository, i.e. *url, owner* and *name* **[property config_correct]** • CF2: config is not valid (passed as None type) **[error]**
starting_date	• SD1: starting_date is present **[if config_correct] [property starting_date]** • SD2: starting_date is not valid (passed as None type)
detected_smells	• DS1: detected_smells is present **[if config_correct] [property detected_smells]** • DS2: detected_smells is not valid (passed as None type) **[error]**

Table 7 Test Frame CR_1-SSC.

Test Case ID	Test Frame	Result
TC-SSC.1_1	CF1, SD1, DS1	Success: the smell is correctly inserted in the dataset
TC-SSC.1_2	CF1, SD1, DS2	Error: the smell is not inserted in the dataset and is asked to the user to insert the parameters correctly
TC-SSC.1_3	CF1, SD2, DS1	Success: the smell is inserted in the dataset without a starting date
TC-SSC.1_4	CF2, SD1, DS1	Error: the smell is not inserted in the dataset and is asked to the user to insert the parameters correctly

5.1.2 Test Frame for CR_1-ATE

Description	
The system must be able to be integrated and executed through an Adapter	
Parameters	
gitRepository, gitPAT, sentiFolder, outputFolder	
Categories	
Category Name	**Category Value**
gitRepository	• GR1: gitRepository is valid • GR2: gitRepository is present but not represent a valid GitHub repository (the string passed is not a valid URL or a malformed one) • GR3: gitRepository is not valid (passed as None type)
gitPAT	• GP1: gitPAT is present and valid • GP2: gitPAT is not present (passed as None type)

sentiFolder	• SF1: sentiFolder is present and contains the SentiStrength classes and data • SF2: sentiFolder is present but not contains the SentiStrength classes and data • SF3: sentiFolder is not present (passed as None type)
outputFolder	• OF1: outputFolder is present • OF2: outputFolder is present but not represent a valid folder in the file system • OF3: outputFolder is not present (passed as None type)

Costraints	
Attribute	**Costraint**
gitRepository	• GR1: gitRepository is valid **[property gitRepository]** • GR2: gitRepository is present but not represent a valid GitHub repository (the string passed is not a valid URL or a malformed one) **[error]** • GR3: gitRepository is not valid (passed as None type) **[error]**
gitPAT	• GP1: gitPAT is present and valid **[if gitRepository] [property gitPAT]** • GP2: gitPAT is not present (passed as None type) **[error]**
sentiFolder	• SF1: sentiFolder is present and contains the SentiStrength classes and data **[if gitPAT] [property sentiFolder]** • SF2: sentiFolder is present but not contains the SentiStrength classes and data **[error]** • SF3: sentiFolder is not present (passed as None type) **[error]**

outputFolder	• OF1: outputFolder is present and represent a valid location **[if gitPAT] [property outputFolder]** • OF2: outputFolder is present but not represent a valid folder in the file system **[error]** • OF3: outputFolder is not present (passed as None type) **[error]**

Table 9 Test Frame CR_1-ATE.

Test Case ID	Test Frame	Result
TC-ATE.1_1	GR1, GP1, SF1, OF1	Success: the adapter calls the tool and the repository is correctly analyzed
TC-ATE.1_2	GR1, GP1, SF1, OF2	Error: the system warns the user that the folder specified does not exist
TC-ATE.1_3	GR1, GP1, SF1, OF3	Error: the system inform the user that an output folder is needed
TC-ATE.1_4	GR1, GP1, SF2, OF1	Error: the system warns the user that the senti folder indicated does not contain the needed files
TC-ATE.1_5	GR1, GP1, SF4, OF1	Error: the system inform the user that the folder with the SentiStrength files is needed
TC-ATE.1_6	GR1, GP2, SF1, OF1	Error: the system inform the user that a GitHub PAT is needed to clone the repository
TC-ATE.1_7	GR2, GP1, SF1, OF1	Error: the system warns the user that the URL is invalid or malformed
TC-ATE.1_8	GR3, GP1, SF1, OF1	Error: the system inform the user that a GitHub repository URL is needed to start the tool

5.1.3 Test Frame for CR_1-CRS

The CR_1-CRS feature changes the way the system show the community smells detected to the user. So, this feature is only responsible for a different output format and does not add any new functionality to the system. The behavior of this functionality will be assessed during the execution of **system** level tests.

5.2 CR_2

5.2.1 Test Frame for CR_2-WSE

Table 10 Category Partition CR_2-WSE.

Description	
The system must be able to handle JSON input parameters and return a JSON formatted response	
Parameters	
repo, pat, user	
Categories	
Category Name	**Category Value**
repo	• GR1: repo parameter is present and represent a valid GitHub repository **[property repo]** • GR2: repo parameter is not present **[error]**
pat	• GP1: pat parameter is present and valid **[if repo] [property pat]** • GP2: pat parameter is not present **[error]**
user	• US1: user parameter is present and valid **[if pat] [property user]** • US2: user parameter is not present
Costraints	
Attribute	**Costraint**
repo	• GR1: repo parameter is present and represent a valid GitHub repository • GR2: repo parameter is not present
pat	• GP1: pat parameter is present and valid • GP2: pat parameter is not present
user	• US1: user parameter is present • US2: user parameter is not present

Table 11 Test Frame CR_2-WSE.

Test Case ID	Test Frame	Result
TC-WSE.1_1	GR1, GP1, US1	Success: the API call is executed with success and csDetector analyze the given repository
TC-WSE.1_2	GR1, GP1, US2	Success: the API call is executed with success and csDetector analyze the given repository but the result of the execution of csDetector are stored in the "default" folder rather than the user specific folder
TC-WSE.1_3	GR1, GP2, US1	Error: the API call fail reporting to the user that the *pat* parameter is missing
TC-WSE.1_4	GR2, GP1, US1	Error: the API call fail reporting to the user that the *repo* parameter is missing

5.3 Testing CR_3

The CR_3 involves the implementation of a natural language based conversational agent. For this reason, the execution of its functionalities have no specific input or parameters, but instead the execution is based on sentences.

In order to specify categories for this CR, we will relax the concepts of Category Partition.

5.3.1 Test Frame for CR_3-I1

Table 12 Category Partition CR_3-I1.

Description	
The system must be able to execute CSDETECTOR *on a given repository*	
Parameters	
repo, pat, user	
Categories	
Category Name	**Category Value**
repo	• R1: the sentence contains a valid repository URL • R2: the sentence does not contain a valid repository URL
Costraints	
Attribute	**Costraint**
repo	• R1: the sentence contains a valid repository URL • R2: the sentence does not contain a valid repository URL **[error]**

Table 13 Test Frame CR_3-I1.

Test Case ID	**Test Frame**	**Result**
TC-I1.1_1	R1	Success: the conversational agent is able to detect community smells and gives them as output
TC-I1.1_2	R2	Error: the conversational agent is not able to detect community smells and tells the user there has been an error

5.3.2 Test Frame for CR_3-I2

Table 14 Category Partition CR_3-I2.

Description	
The system must be able to execute CSDETECTOR *on a given repository*	
Parameters	
repo, pat, user	
Categories	
Category Name	**Category Value**
repo	• R1: the sentence contains a valid repository URL • R2: the sentence does not contain a valid repository URL
date	• D1: the sentence contains a valid date • D2: the sentence does not contain a valid date
Costraints	
Attribute	**Costraint**
repo	• R1: the sentence contains a valid repository URL **[property repo]** • R2: the sentence does not contain a valid repository URL **[error]**
date	• D1: the sentence contains a valid date **[if repo]** • D2: the sentence does not contain a valid date **[if repo] [error]**

Table 15 Test Frame CR_3-I2.

Test Case ID	Test Frame	Result
TC-I2.1_1	R1, D1	Success: the conversational agent is able to detect community smells starting from a specific date and gives them as output
TC-I2.1_2	R1, D2	Error: the conversational agent is not able to detect community smells starting from a specific date and tells the user there has been an error
TC-I2.1_3	R2	Error: the conversational agent is able to detect community smells but not starting from a specified date so the intent is not fulfilled