Assignment 3 – Project Final Report and Artefact

Group 28

UNIVERSITY OF LINCOLN Team Software Engineering

Contents

Software Engineering

At first the software development method originally planned was the Waterfall Model. This is where sequential activities would occur in a linear structure of development. This strategy approach would have succeeded had the group had a solid foundation to build the project from. However, due to various issues discussed later, the team decided a different approach would be more appropriate. The group's new strategy was the software development approach of Scrum. The definition of scrum fit most well with the group's method of approaching tasks. After the group set a task, the members set a deadline and completed the task collectively or individually within the allotted time. The team needed to set priorities to the most important tasks. One example was to set a clear line of communication with the team supervisor regarding issues with the compatibility of the Nao robot.

The scrum master was Jackson. He had the primary role of communicating with the product owner/coordinator. This was done via email.

The product backlog was broken down into the following requirements: facial recognition, distance, arm angles, integrating navigate me, speech recognition.

Repeated inspection of working software was done by Dan, who primarily oversaw testing procedures and documentation of technical aspects.

The requirements for each sprint were produced in the product backlog, where individuals were set tasks. This is shown in the table below.

Product Backlog – To Do List

Story	Estimation	Priority
As a user I want to the robot to understand when I talk (Speech Recognition)	5	3
As a user I want the robot to look at me	4	5
As a user I want the robot to point in the direction of where I want to go	3	2
As a user I want the screen to display the map of where I want to go (NavMe)	3	4
As a user I want the robot to talk back to me (Robot Speech)	5	6
As a user I want the tablet to function	2	1
As an administrator I want to release the robot into INB.	3	7
As an administrator I want to program the robot. Change angles when the robot is set up. This is part of the maintenance portion of the software development cycle.	4	9
As an administrator I want any user feedback to come back to me so I can use that information to fix any issues that it has.	1	8
Total	30	-

The software was constructed in Choregraphe. This is discussed further on in the document.

Implementation

Planning

This section will explain how the group have programmed the robot and developed it. The group's approach of scrum was highly justified, due to the high trial and error the group faced with obtaining

usable software. The scrum strategy allowed the group to operate as fast and efficient as possible. The group had both the repeated inspection of working software and delivering important tasks in the shortest time possible.

The group methodically worked its way through combinations of operating systems, such as Ubuntu, and software suites, such as ROS, as well as differing robot types, such as the turtle bot, in order to obtain the ideal scenario with which to work with. This was chosen to be Choregraphe, which allowed for the importation of python code, giving a greater deal of customisability with the NaoBot, as well as a virtual copy giving a way to work without the physical robot. Each member would be given individual tasks on different parts of the robot. For example, there is speech recognition, arm movement and facial recognition. Additionally, the scrum master (Jackson) would maintain contact with Team Software Engineering staff and the program leader.

Dan's Task

Dan was tasked to take the string input from Speech Recognition, and generate a scan-able QR code to direct the end user to NavigateMe – with the destination room and current location already pre-filled in. This was done by importing the room string from the Google Speech-to-Text API Python script and handling the string to be parsable through the NavigateMe URL parameters. Once the URL for the destination has been generated, it is passed through the PyQRCode library (Found at: https://pythonhosted.org/) to generate a QR code, to be rendered using the PyPNG library (Found at: https://pypi.org/). This is displayed on the display below the NaoBot. The purpose of the displayed QR code is so that as the user leaves the robot, the directions that are given are accessible even without being next to the NaoBot.

Dan also created the logic behind the arm movements – taking the floor and zone number from his earlier code to implement a dictionary-based direction system, so Sindy can further animate on these directions and provide paralinguistic feedback from the robot.

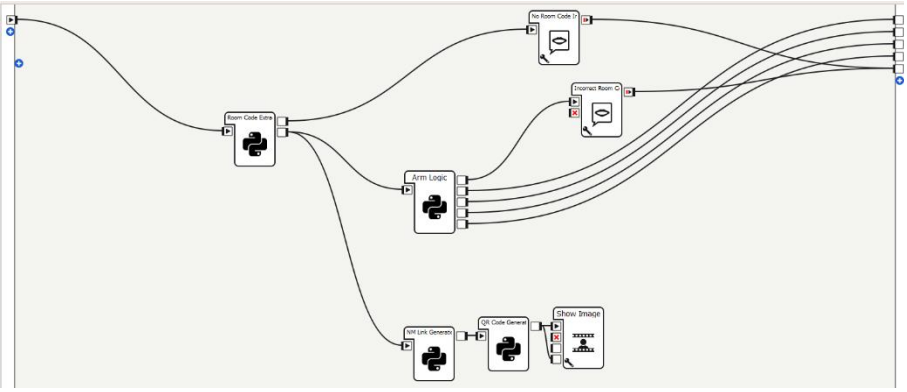

Figure 1 Image of Speech Recognition, NavigateMe URL Generation, and Robot Arm Logic on Choregraphe

Furthermore, Dan ensured the robot was reset to a standing rest with the screen cleared of any images upon program completion, so that the next end user would be able to interact with the robot.

Sindy's Task

Sindy's role was to manage the arm movement of the robot. Regarding Nao's arm movement, the goal was to make Nao point to the direction that the user requested as naturally as possible. Through using

Choregraphe, there are simple functions enabled on the physical robot on storing body movements which can be adjusted by simply moving Nao's joints. However, without having access to Nao, the movements had to be done through precise angles and using keyframes. By using Choregraphe, it was beneficial being able to see a virtual version of Nao which enables making movements as smooth as possible.

The first task was to gather the sense of awareness of where Nao would be positioned on the INB reception desk. This was planned out through sketches and measurements. An important factor to take into consideration was where Nao could respond best to users. After sketching out a rough idea of the placement for Nao, the next step was to familiarise with Nao's joints and the way they move. Nao has two moveable joints in the shoulder, two moveable joints in the elbow, one at the wrist and an additional hand grasping mechanism. As Nao has prehensible hands with three operable fingers that move simultaneously, it would result in Nao using the movement of opening and closing the hand to mimic pointing in a direction. After trialling this out, Nao's hand opening movement at the time of pointing does reflect the sense of pointing.

After being familiarised with Nao's joints, the next step was working on the different arm movements for different locations. Nao has now gained arm movements to the left (*figure 2*), right (*figure 3*), forwards (*figure 4*) and backwards (*figure 5*) direction. As Nao cannot twist in the hip, the most natural movement was to rotate Nao's head to make it seem like Nao is looking past the shoulder. Implementing Nao's arm movements consisted of making a timeline box (Timeline — NAO Software 1.14.5 documentation, 2020) and making keyframes by storing movements in the arms, head and fingers.

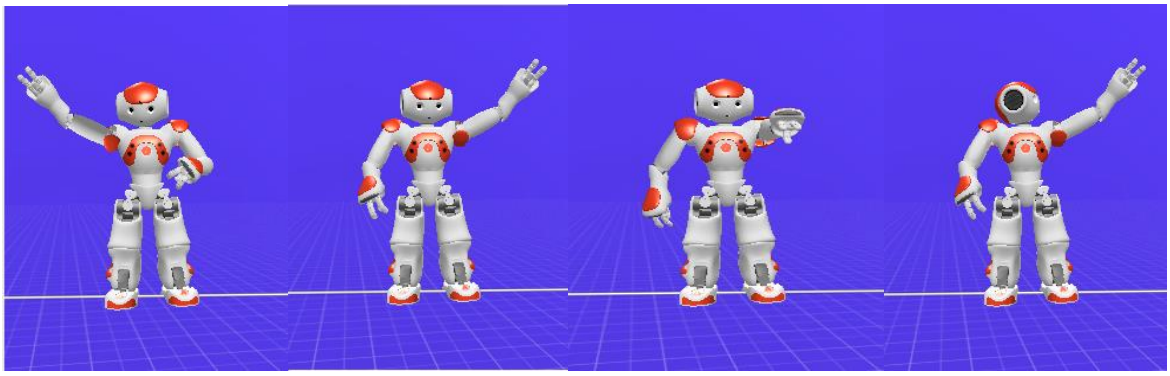

Figure 2. *Figure 3.* *Figure 4.* *Figure 5.*

The idea was that once having physical access with Nao, any movements that may not be as accurate can be stored using the store joints function on the robot to improve the pointing accuracy to certain locations. However, without access to Nao, it is impossible to understand the degree of accuracy of these movements.

Blake's Task

Blake's task consisted of discovering how the sonar worked as well as how Choregraphe passed on variables or changing the output depending on what event occurred. Blake knew how to structure the rest of the program, which made his task easier from that point. From discovering how the sonar worked, he helped fellow members of the team with working on sensors and showing how to make

loops from outputs. Initially, this took extensive guess work as not many tutorials touched on it or demonstrated a method to do it. He did however use 'Robo Phil' on YouTube for assistance with this task. When most of the team were ready, Blake helped Bradley bring together the pieces of work to form a near complete program. All that was left to implement was the voice recognition and the motor control.

Bradley's Task

Bradley started out by researching how the facial recognition functions on the Nao bots. Without the use of a physical bot, he needed to improvise with what the group had. Bradley started out by researching and looking at different forums pages on how to create the code. He used what information he found and what he already knew. When he had the code working for the robot, he knew it was working because it ran without any errors. Afterwards, he worked with Blake to implant his sonar code into the facial recognition software. By doing this, the group had successfully created a software that has both sonar and facial recognition. All that was left was to add the voice and motor code.

Jackson's Task

Jackson was tasked with implementing Speech Recognition that was able to understand the users request and respond accordingly. The implementation of Speech Recognition allows NAO to verbally guide users to desired rooms and points of interest within the Isaac Newton Building, also being aided by gestures from NAO pointing the user in the approximate direction they should go. Choregraphe allows simple speech recognition to be implemented, however this only works for specific phrases, and didn't feel natural enough for scope of the project, it also requires the user to touch a sensor on top of NAO's head to stop recording the users voice. Because of this API's were utilized to extend the capabilities of the speech recognition.

The API's used were Python Speech Recognition (Zhang, A. 2017) and Python Chatter Bot (Belanger, P. D. 2017). Python Speech Recognition allows sound files to be uploaded to the Google Cloud Speech-to-Text service, whilst the Python Chatter Bot, alongside Pandorabots allows NAO to search the internet for answers when certain information cannot be found.

Dylan and Joseph's Task

Both individuals were tasked with writing up the main document (this one). They would obtain a summary from each of the other members and write them up to the document in an appropriate format. They also took the role of directing to the other members what details needed to be included in the project. Therefore, they also had a communicative role to the project and each of its members.

Testing Strategy

End-User generated testing has not been possible with this project, as due to physical limitations, the code has not been able to be deployed onto the Nao Robot, and has not been able to be presented to the general public for use. Therefore, the project was primarily held to two testing standards during development: Runtime Verification, and Active Unit Testing.

Runtime Verification

Runtime Verification is the active testing of inputs and outputs during coding, using stack analysis, variable tables, and the output logs of the compiler. This is essential for any development of software, as

the programmer can test lines of code individually, with hard-coded and computed values, as well as view the output of a function (to compare to expected values or responses). This is time effective as one would not need to deploy the code to perform this testing. This method was navigated by reading the works of *Ezio Bartocci & Ylies Falcone* (2018).

Within the context of this project, debug lines of code were used throughout the python scripts; created to show the input and outputs of each computational line of code. This was done to ensure data was being calculated and output correctly. The following example of this can be found within the "Room Code Extraction", after the Speech to Text API has been implemented:

```
31
32          # print(z)
33          # print(z-3) DEBUG
34          # print(z+4)
35
36          s= s[y:z+4] # Extract the letters and numbers of the     room code, from the original input
37          s = s.strip() # Strip any preceeding or following spaces
38
39          # print(s) DEBUG
40
```

The above code shows the room code being printed ("z"), then the attempt of splitting the room code into 'building' ("z-3") and 'room number' ("z+4"). This is useful as it shows the values going into the lines of code on line 36 and 37, then follows with an output being printed to compare to the expected values.

This form of testing has occurred on all inputs and outputs on most functions during the development process, as this is the easiest and least intrusive testing procedure to the programmer. It provides real-time feedback and is easily editable to ensure the functions that are being tested produce the expected results.

Active Unit Testing

Active Unit Testing is a method of which the program is compartmentalised into its classes and functions, and each section is tested independently form each other prior to final release. The benefits of this method include a comprehensive understanding of the direct inputs and outputs of the code, as well as promoting professional code writing through utilising encapsulation. This method is much more thorough in error-finding however, it is not expected to discover many issues due to Runtime Verification happening prior to it.

The process of testing, as described by Paul Hamill (2004), begins by selecting a subroutine or class and taking note of its' primary uses. For this example, I will again be using a modified extract of the "Room Code Extraction" Python Script, that has the debug options (see above) taken out.

```
def onLoad(self, p):

        import re # Regular Expression Engine
        import sys # System Library

        s = p # Set s as input string

        x = re.compile('\d\d\d\d') # Set four digits (the floor, zone and room code), as the regular expression format.

        try: # Search for four digits in the input string, and input it into variable v
            v = re.search(x, s).group()
        except:
            self.outputNoRoomCode() # Submit error, room code not found

        z = s.find(v) # Find the position of v within s
        if z < 3: # If it is at the start of the original input string
            y = z-2
        else: # If it is in the middle of the original input string
            y = z-3

        s= s[y:z+4] # Extract the letters and numbers of the     room code, from the original input
        s = s.strip() # Strip any preceeding or following spaces

        self.outputRoomCode(s) # output room code

    pass
```

Below is a table with the key functions written in, and the lines of code the tests refer to. Each test that will be carried out is given a numerical ID, followed by ".1", as it is the first time it will be tested. Each specific part of code will also have its inputs and expected outputs listed, along with the output received from testing. If a test fails to get it's expected outcome, another test is performed, and the testing number is incremented.

#	Description of function	Code	Input	Expected Output	Received Output
1.1	Import of necessary libraries	Lines[3:4]	None	Successful Import	Failed to find library
1.2			None	Successful Import	Successful Import
2.1	Set variable "s" from the input string "p"	Line[6]	~p= "blah blah blah" ~p = ""	~s = "blah blah blah" ~s = ""	~s = "blah blah blah" ~s = ""
3.1	Compile a regular Expression format of four digits.	Line[8]	c('\d\d\d\d')	Successful Compile	Successful Compile
4.1	Set the regular expression format generated to variable "x"	Line[8]	Compiled RegEx Format ('\d\d\d\d')	~x = {format}	~x = {format}
5.1	Perform a RegEx search within string "s" for the format generated in "x", and return the four digits.	Line[11]	~s = "INB0114"	Return "0114"	Return "0114"
6.1	Store the digits of a found RegEx string format in variable "v".	Line[11]	~s = "INB0114" ~s = "MB0603"	~v = "0114" ~v = "0603"	~v = "0114" ~v = "0603"
7.1	Perform a RegEx search within string "s" for the format generated in "x", and if it cannot be found, then terminate the script and active the error output.	Lines[10:13]	~s= "0 114" ~s = "something" ~s = ""	NoRoomCode() NoRoomCode() NoRoomCode()	NoRoomCode() NoRoomCode() NoRoomCode()
8.1	Find and return the starting position of the four-digit string "v", within "s".	Line[15]	~v = "0114" ~s = "Go to INB0114" ~s = "INB0114"	Return 10 Return 3	Return 10 Return 3

9.1	Store the returned value of the starting position in variable “z”.	Line[15]	Returned 10 Returned 3	~z = 10 ~z = 3	~z = 10 ~z = 3
10.1	Check if the room code is at the start (<3 position) of the Speech-to-Text input string, “s”.	Line[16]	~z = 0 ~z = 1 ~z = 5	True True False	True True True
10.2	Check if the room code is at the start (<3 position) of the Speech-to-Text input string, “s”.	Line[16]	~z = 0 ~z = 1 ~z = 5	True True False	True True False
11.1	If #10 returns true, set the start position of the room code to “z-2”, as variable “y”	Line[17]	Successful	Successful	Successful
12.1	If #10 returns false, set the start position of the room code to “z-3”, as variable “y”	Line[18:19]	Successful	Successful	Successful

This method helps highlight errors and exceptions within the code, by using a variety of inputs, that can then be procedurally fixed as errors are found.

Release

Due to circumstance out of the group’s control there was no opportunity to release the code for the receptionist robot online but if the code was released, it would be on Nao where the code gets run from anyway. A benefit for releasing the code on Nao would be making it open source thus allowing for people to use it and make improvements. Another benefit with making it open source is that it does not cost anything to upload it. As a result, this will save on the cost of the project.

One of the drawbacks that came to the group before the code was released was not having access to the building layout. Due to COVID-19 the group plan of the project was compromised as they did not obtain the chance to get the layout of INB, without referring to the NavigateMe system (which was part of the project being implemented). Another drawback was the limited meetings and contact the group had with the supervisor while creating the code/robot as they could not get feedback on the code to improve it. The group never had the opportunity to interact with a robot, so they had to make do with using the virtual robot in cerograph to test out the code.

Hypothetically if the group were to release the code on to Nao, they would make sure all bugs are fixed in the code as well as any bugs discovered post-release. The group would achieve this via keeping the program regularly updated.

It also must be noted that assumptions have been made due to the restrictions placed on the project. Through experimenting with the Speech to Text API, and other voice activated ‘smart assistants’, it was agreed that the string input for a room code is more likely be input as one word (For example, “INB0114”, rather than “INB 0114”). It has also been assumed that the “Show image” box-object on Choregraphe, provided by the Nao SDK, is supported by the Nao robot and the setup provided by the supervisor. Without access to the proposed setup, these assumptions must be made, otherwise development of the code would have been strongly hindered.

Evaluation

Reviews

Due to equipment constraints such as no physical access to Nao, the group cannot release the artefact. As a result, the project lacks user reviews and responses of the robot's performance. The group had prepared a range of questions to ask the audience once they had used the robot such as the following:

- Would you say you were comfortable with your experience with the Nao? (Strongly Agree - Strongly Disagree) - Likert Scale Question

- Would you say it was easy to use? (Strongly Agree – Strongly Disagree) - Likert Scale Question

- How natural was it to use? (1-10, 10 being very natural) - Rating question

- Did it understand your destination? (Yes or No) - Closed-ended question.

- Did Nao provide directions to your destination? (Yes or No) - Closed-ended question.

- Are there any features/improvements you would wish to see implemented? - Open-ended question to engage with all users' ideas for improvements

These questions would have led to a better experience in the future for users by implementing their feedback. Based on the feedback, the robot would have been improved and better built for user's needs.

Problems

During the beginning stages of the project, the group was informed by the supervisor to use ROS Kinetic to program the Nao with the python SDK however, Gazebo and many simulators could not use ROS kinetic to work and required ROS indigo. This brought issues to when it came to programming and testing when it would come up with more errors than the team could work with. So, the group did a fresh install of just ROS indigo and tried to program with the python SDK, however this repeatedly produced errors the team did not understand. Furthermore, the team did not obtain any assistance from their supervisor. This led to looking for a different IDLE to program in, thus giving the team more useful feedback. So, the team turned to Choregraphe which seemed far better and easier to work with and error fix. This is what the current version of the program was done in and it also enabled simulations within the IDLE up to a certain degree. On reflection, the team worked very well together with solving the issue explained above. This was partially due to the adaptable use of scrum allowing the team to take immediate fast action upon any situation. However, most of the success can be credited to how well the team worked together, all having and attentive and strong work ethic to get the work done no matter what barrier.

The current problems that the group can see occurring are that the robot cannot give point of accuracy directions if the user room input is on a different floor to where it is set up. This will mean it would have to be reprogrammed to be set up in a new location. As most of these directions are hard coded from a certain place (the front desk), problems will arise if Nao was placed anywhere else. This is due to the inaccuracy when pointing. Another issue that could arise is the disruption of any loud noises that could put off the voice recognition. This is an issue as the reception desk is near the entrance to the Isaac Newton Building which has a high footfall rate. The details the group would focus on if they were to

continue would be the speech recognition and the directions when pointing. Additionally, there is the fact that the group cannot test the robot without the required hardware which was cut off during the epidemic lockdown.

Improvements

The group concluded that it would be better if more welcoming voice options were included. That way the robot can say different things that get randomly selected. Another improvement that could be made is directing the users to the nearest toilets/vending machine. This allows Nao to provide guidance to areas that is beyond just rooms within INB. Questions such as 'Where are the nearest toilets/lifts' are commonly asked questions at the INB reception desk. Therefore, implementing this feature would be beneficial to the receptionist staff in the future. There could also be other future improvements such as a feedback system that will be optional for the user to take part in. By having this feature implemented, there is the ability to deliver constant feedback and improve Nao to the best it can. Another potential improvement would be if the robot could show a highlighted map on the screen as well as a QR code in the corner as not all phones may have an in built QR code reader.

Improvements on a larger scale could include Nao being used in other departments and not just limited to INB. This will allow for larger engagement with Nao across the campus and can assist a wide range of users. For example, another Nao Robot could be placed at the Minerva building which assists with the directions of rooms, cafeteria, multiple lecture halls and nearest toilets.

Another potential improvement would to be add language options to the robot as there are different nationalities of students and lectures in the university. This would enable easier communication with the users and enable a seamless interaction with them this would also require the robot to speak in their designated language so that it's easier to understand the robot.

The last potential improvement would be a help button, for if instructions were unclear or it is unable to help them with their problem/ directions and it would either ask to request assistance first or if they press the designated button. Which could link to alerting a receptionist with a light or buzz to let them know that someone is requiring assistance at the front desk and the robot was unable to help them with their request.

Group Work Conclusion

Although there were numerous obstacles that the team experienced whilst working on the project, overall, the team believes that the group had worked the best they could together. The first obstacle of those was that whilst the group's coordinator had sent documentation to assist in learning, the documentation was about an IDE that was incompatible with the Nao robot. This led to the group being proactive and making important decisions on finding a new environment the work on.

To overcome this obstacle, the group had researched into about multiple different environments and concluded on using Choregraphe. With Choregraphe, the group was able to connect to a virtual version of the Nao robot. It was also simpler to grasp and get used to as it is a visual environment compared to ROS, which was entirely done on the command line for its programming. The group thought Choregraphe was the best option forward as it provided a virtual robot that the group could experiment with without the access of a physical robot. Whilst this meant that by using Choregraphe, the group would not have as much freedom as they would of using other environment, Choregraphe was still the

best option for the group as there was the ability to hardcode python to make things more flexible to the project. As a result, the project was successful and did not feel like there were any limitations on the flexibility of the code.

The group believes that the overall Scrum methodology that was adopted into this project worked well. Setting deadlines for tasks to split between each individual member meant that everyone worked as a team and no one was left out with nothing to do. Whilst each member had their own tasks, the group supported each other with suggestions and voiced their opinions as the project carried on.

On reflection of the first assessment's reflection, although there was minimal communication with the group coordinator, the group were actively more proactive to function as a team and made sure that the project was always on track in order to accomplish the project together. Although the group had proceeded to progress onwards with reduced support from a coordinator, this gave more drive to complete the project.

Artefact and Media Materials (Links)

Due to the code not being able to be deployed on the Nao robot, the code has been submitted to Blackboard support materials, as requested.

Our created media material for this project can be found at **https://youtu.be/u4AxkhHFSUs**.

Due to no physical access to Nao, the group came up with an animation in hopes to promote the groups artefact to its target audience.

References

Robo Phil (2015) *Python Programming Your Robot.* [online video] Available at: <https://www.youtube.com/user/robotphilip/videos> [Accessed 05/03/2020]

Doc.aldebaran.com. 2020. *Timeline — NAO Software 1.14.5 Documentation*. [online] Available at: <http://doc.aldebaran.com/1-14/software/choregraphe/panels/timeline_panel.html> [Accessed 10 March 2020].

Pythonhosted.org. 2020. PyQRCode Official Documentation [Digital Documentation] Available at: < https://pythonhosted.org/PyQRCode > [Accessed 19 March 2020]

Pypi.org. 2020. PyPNG Official Release [Website] Available at:<https://pypi.org/project/pypng/> [Accessed 20th March 2020]

David Belanger, P. (2017) *Chatter Bot API* [Software]. Montreal: David Belanger, P. Available at: https://github.com/pierredavidbelanger/chatter-bot-api [Accessed 27th March 2020].

Zhang, A. (2017) Python Speech Recognition [Software]. Ontario: Zhang, Anthony. Available at: https://github.com/Uberi/speech_recognition [Accessed 28th March 2020].

Paul Hamill (2004) Unit Test Frameworks: Tools for High-Quality Software Development. [Book]

Ezio Bartocci & Ylies Falcone (2018) Lectures on Runtime Verification, Introductory and Advanced Topics. [Book]