UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering,
Mathematics & Computer Science

# Duckbot
# A chatbot to assist students in programming tutorials

**Margot Rutgers**

**M.Sc. Thesis**
**August 2021**

**Supervisors:**
Dr. Ansgar Fehnker
Dr. ir. Randy Klaassen
Dr. Khiet Truong

Human Media Interaction Group
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
Drienerlolaan 5
7522 NB ENSCHEDE
The Netherlands

# Preface

The basis of this thesis was my many years of working as a teaching assistant in Creative Technology, close collaboration with the teaching staff and a personal interest in education. It was a natural next step when working on Atelier, a research project with a vision of improving education for both students and teaching staff. Many of our meetings were spent not talking about technology, but talking about our experiences. What made programming in Creative Technology so unique, and how could we make it even better?

The Atelier project was lead by Ansgar Fehnker, to whom I am very grateful for also guiding this project as my supervisor. Although Covid-19 eventually had us meeting online instead of at Starbucks, it was great working together. Also leading Atelier was Angelika Mader, my first programming teacher, who encouraged me to be involved as a teaching assistant and become a programmer. I would also like to thank those students I worked with during the Atelier project. And of course the TAs contributed to this research, especially Dennis and Jur, my TA-colleagues for years.

I would like to thank Randy Klaassen and Khiet Truong for their supervision during the later stages of the project. Their constructive criticism greatly elevated the quality of my thesis.

I have been lucky enough to have been involved with the development of my bachelor, Creative Technology, and master Interaction Technology, throughout my entire time at the University of Twente. I am grateful to the teaching staff and in particular programme director Alma Schaafstal, for giving me these opportunities.

Many, many thanks to my family and my friends - especially Joelle, Carmen, Job, Lieve, Emma, Tom, Joep. You got me through this project, and through lockdown!

Finally, always in my thoughts are to those who were there when I started this, but who now we miss. Dear Ad, thank you so much for your love, support and inspiration for everything I do. Opa Marinus and oma Annie, I wish you were here to see me graduate. I know you would be proud of me graduating in ’Something With Computers’.

# Abstract

Projects concerning the creation of an online platform to aid programming courses in the bachelor study programme Creative Technology are ongoing. A key principle is to help foster a Community of Practice, where members of the community - teaching assistants and students - can learn together. This research aims to identify opportunities where a new online tool can be of aid. Guiding this research is the main research question: **"How can a digital tool enhance the quality of help seeking and giving, between students and teaching staff, in Creative Technology programming tutorials?"**

Concerns raised by TAs during a brainstorm session and a questionnaire included that students have problems identifying the cause of programming problems, and as such have a hard time asking the right questions to TAs, sometimes leading to students not making use of the help available at all. TAs expressed a wish for a tool that focuses on making time between students and TAs more useful, instead of completely moving help to an online environment, and nudging students towards being proactive in their help seeking.

Literature supports that seeking help can be difficult for students because the process can feel threatening, and because they do not know how to effectively ask for help. A computer-based solution can assist in both making the process less threatening, and guiding the student in effective help seeking. Self-talk is a strategy that can be incorporated as well to support learning.

A conversational agent, in particular a chatbot, was identified as a promising technology. It can fully guide students step-by-step to ask quality questions, make the process of asking questions less threatening, allow aspects of explaining and self-talk and be built into digital platforms currently used.

Inspired by previous research and popular online programming Q&A strategies, a chatbot was designed. It is named Duckbot, after the self-tack strategy 'Rubber Ducking', where people learn more about their problems by elaborating about it to a human or inanimate partner. Duckbot will ask the student for specific Problem Statement Details, such as 'goal statement', 'actual and expected results', 'relevant error messages', 'relevant code' and 'steps taken'. It is accessed through TA-Help.me, an online platform that students currently use to queue up questions when requesting help from TAs.

A user test of Duckbot yielded favourable results, with students recognising it helping them communicate problems, it being user friendly, and helping to understand their own problems more. Opportunities for improvement included technical errors to fix and Duckbot not fully understanding all user utterances.

It is recommended to do further research with a larger group of participants, add categorisation of help request types, expand the possible dialogue of Duckbot, and make it part of a larger social platform.

# Contents

# Chapter 1

# Introduction

Within the bachelor programme Creative Technology at the University of Twente, programming is one of the main topics. As it is part of a broader spectrum of computer science, electrical engineering and design topics, programming is taught differently from other engineering programmes: students look at programming through a lens of design and creative thinking, and programming assignments are largely self-defined. For example, knowledge of physics and mathematics can be demonstrated by programming a landscape with grass and birds moving as in nature. As such, there is no answer sheet: teaching assistants (TAs) and teachers help students by diving into their specific challenges, face-to-face.

To support the teaching team, the research project *Atelier* was founded. It is an online platform that is still in development and offers a way for students and teaching staff to communicate (about) code. The supporting philosophy is that of the Community of Practice: students and teachers should be able to form a community where the interests of, and interactions between, members are of mutual benefit.

This thesis started out when working on the Atelier platform. Currently the platform primarily offers a way to upload and discuss code online. However, there are opportunities to improve the process of help seeking and giving. For this thesis, we investigated the concerns of TAs concerning this process. How can we address these using an digital tool?

Research in topics such as the Community of Practice and Persuasive System Design - can students be motivated to use this tool? - as well as the experiences of teachers and teaching assistants, became the context for creating a system to be designed in this thesis. A strong sentiment of the Atelier project, which was echoed by TAs, was that a digital tool should support, and not replace, quality interaction with students. The quality of the interaction increases when students are pro-active and have skills in question-asking.

From that thought eventually came the idea of a chatbot that would help students understand the problems they encountered better, which would in turn lead to a more fruitful help session with a TA if requested.

## 1.1 Background

In preparation for this thesis, we undertook a literature review linked to the Atelier project, the Community of Practice ideology, and the design of engaging and persuasive systems. This literature review yielded some interesting insights and suggestions for designing an online support tool for Atelier. The full literature review can be found in the report for the “research topics” project. Here, a summary of the report is given as a background for this thesis.

### 1.1.1 Programming in CreaTe

Creative Technology has its own view on programming education, differing from more traditional Computer Sciences study programmes. Its teaching method, *tinkering in informatics*, has been presented in literature by Mader et al. [1] They aim to support the IT related programme Creative Technology which, unlike programmes such as computer science, is focused on leveraging *various* types of technology, as opposed to focusing on one. Students aim for creative solutions to *self-defined* problems. The student population is diverse in terms of gender and nationality, and its students do not have the sole aim of learning to programming. Instead, they learn programming as one of their tools. This requires a different approach to teaching programming than more traditional computer science programmes [1].

Mirroring the overall working method of Creative Technology, tinkering is a method where students have self-directed programming challenges. They experiment with the tools available to them, to create original solutions.

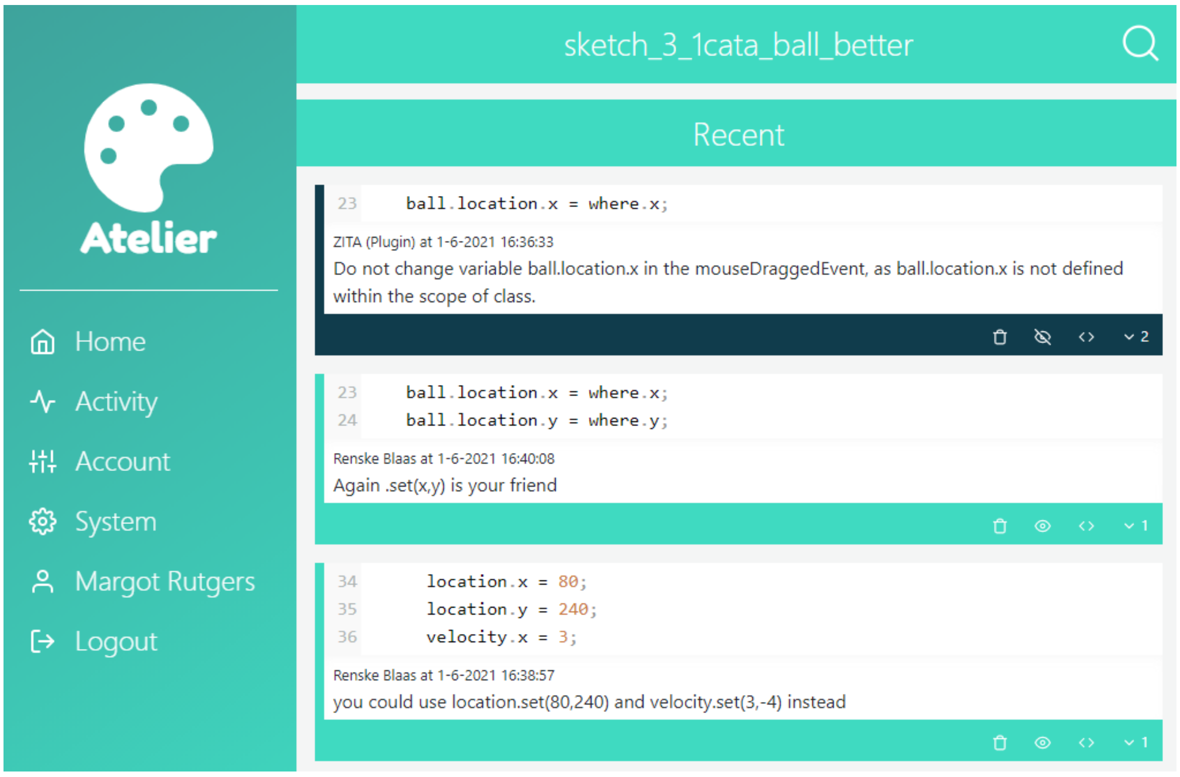
For most of their time, students learn to program in the programming language ’Processing’. It is syntactically equal to Java, but the accompanying programming environment makes it especially suitable for graphics and user interaction. Students learn various algorithms, often representing natural behaviour in graphics.

An example of an assignment could be: *Create a program where the physics of a mass-spring-damper system are showcased. Additionally, flocking behaviour should be involved. Include interaction between the user and the program.* One students could make a game where a flock of fishes will be steered through waving seaweed.

Another will allow the user to control a catapult that catches birds from a flock. Even completely abstract 'art pieces' are sometimes made.

### Atelier

As part of the Atelier project, the Atelier code platform was developed. When students ask a TA for help, they upload their code to Atelier first. That way the TAs can look at the code in preparation, and if desired, ask other TAs to look at it via the platform. When TAs help students face-to-face during the tutorial, they are advised to comment on the submitted code as well, so that the student can review these afterwards. Additionally, by adding tags, interesting comments and code snippets may be shared with TAs and fellow students. An example of comments on code is shown in Figure 1.1.



```
    ball.location.x = where.x;
```

ZITA (Plugin) at 1-6-2021 16:36:33
Do not change variable ball.location.x in the mouseDraggedEvent, as ball.location.x is not defined within the scope of class.

```
    ball.location.x = where.x;
    ball.location.y = where.y;
```

Renske Blaas at 1-6-2021 16:40:08
Again .set(x,y) is your friend

```
      location.x = 80;
      location.y = 240;
      velocity.x = 3;
```

Renske Blaas at 1-6-2021 16:38:57
you could use location.set(80,240) and velocity.set(3,-4) instead

**Figure 1.1:** TA's comments on a student's code submission

## 1.1.2 Atelier & the Community of Practice

The *Atelier* project was founded to make the tinkering teaching method more scalable and sustainable. It aims to foster a *Community of Practice* that has a programme-specific vision of good software design. As such, the class can be more self-reliant.

Wenger [2] defines a Community of Practice (CoP) as ”*...groups of people who share a concern or a passion for something they do and learn how to do it better as they interact regularly*”. Crucial characteristics are (1) a shared *domain* of interest, (2) a *community* of people that interact and learn together, and (3) a *practice*. A CoP has benefits internally (shared knowledge) and externally (produced results).

Individuals can use and support the community through various activities, such asking for assistance for a problem, requesting specific information, discussing developments, and reusing the the community's existing assets for new goals. These activities are applied differently depending on the organisation; for example, coders might share code with others, while teachers might benefit from each other's experience in certain courses [2]. The organisational benefits of CoP can be quantified by traditional measures such as time and financial cost analysis [3]. Employees that have a community to share and request information can avoid reinventing the wheel, saving time.

### CoP in education

Education is an interesting application domain of the Community of Practice. Unlike in other organisations, learning is not there to achieve a business end product - it is the end product itself. Changing the learning theory to incorporate CoP has therefore a much greater impact in education, and thus the change requires greater effort [2].

According to Merriam et al. [4], who stress the importance of community and practice in learning, ”the classroom itself can be viewed as a community of practice where knowledge is shared and expertise resides not only with the instructor but also across the community”. When taking a constructivist approach to learning, it is suggested that all members - including students - of a learning community are teachers with their own experiences [5].

### CoP in online platforms

When using technology to communicate, a CoP is no longer place- and time- based [6]. The design of a virtual or online community based on a network technology (e.g., the World Wide Web) does not automatically form a CoP, rather a CoP can emerge from the virtual community by how its participants use it to interact with each other. A well-designed online community can lead to, but does not guarantee a CoP [7]. Still, there are many examples of successful online CoP, such as in nursing [8].

Schwen and Hara [9] argue technology should be designed to support *existing* CoP. They stress that Wegner's theory around CoP does not offer instructions on how to form one. Therefore, social patterns in an existing CoP should be analysed to find where technology is suitable to help achieve its goals. Nichani and Hung [10] also find that virtual environments should complement and not replace non-virtual communities to function as a CoP.

Four techniques to benefit from technology in CoP corresponding to the C4P framework [11] are (1) *connections*: linking people in the same practice, (2) *content*: provide a shared repository of information resources, (3) *conversation*: support communication by providing discussion tools and (4) *information context*: providing awareness of the information context of resources (e.g. Wikipedia's editing history and entry discussions, or web shop recommendations such as "Others who bought this book also bought....") [12].

A benefit of the online community is that group norms are no longer depending on physical presence, such as voice or visible reactions, enabling introverted and extroverted people to participate equally [6].

Needing to work around the limitations of the technology can inhibit the growth of a CoP, and members must be well versed in using the technology [6]. Another major threat to the online CoP is members disengaging from the online community, after all the community is not formed around physical presence: the members must actively use the platform to not fade out [13].

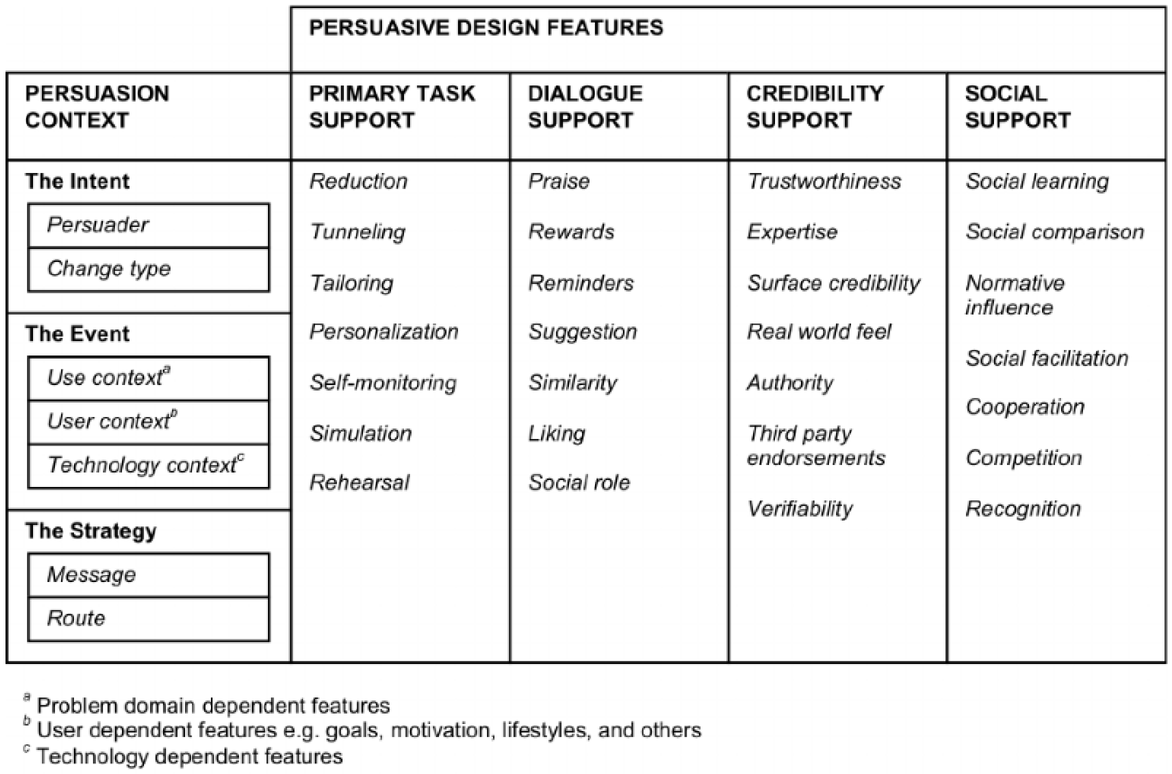
### 1.1.3 Engagement & Persuasive design

In section 1.1.2, members disengaging from the online community was found to be a threat to the CoP. While novice members of the community have the direct benefit of being able to learn from the community, expert members should also find value in helping others to remain active.

If we design a platform where we use engagement techniques to influence the behaviour of students, then the platform would be considered a *persuasive system* [14]. Features to make such a system successful are mapped in the PSD (persuasive systems design) model [15]. Figure 1.2 provides an overview of persuasive systems design features, and parts of the persuasion context that are useful to analyze, which are further detailed elsewhere [16].

In the most popular online (programming) communities, various features outlined in the PSD model are implemented in some form of gamification.

For instance, StackOverflow is an example of an important online community for programmers, with over 11 million users [17]. It is a question-and-answer (Q&A) site, that keeps users engaged using a reputation system: community-approved



|  | **PERSUASIVE DESIGN FEATURES** |  |  |  |
|---|---|---|---|---|
| **PERSUASION CONTEXT** | **PRIMARY TASK SUPPORT** | **DIALOGUE SUPPORT** | **CREDIBILITY SUPPORT** | **SOCIAL SUPPORT** |
| **The Intent** | *Reduction* | *Praise* | *Trustworthiness* | *Social learning* |
| *Persuader* | *Tunneling* | *Rewards* | *Expertise* | *Social comparison* |
| *Change type* | *Tailoring* | *Reminders* | *Surface credibility* | *Normative influence* |
| **The Event** | *Personalization* | *Suggestion* | *Real world feel* | *Social facilitation* |
| *Use context*[a] | *Self-monitoring* | *Similarity* | *Authority* | *Cooperation* |
| *User context*[b] | *Simulation* | *Liking* | *Third party endorsements* | *Competition* |
| *Technology context*[c] | *Rehearsal* | *Social role* | *Verifiability* | *Recognition* |
| **The Strategy** |  |  |  |  |
| *Message* |  |  |  |  |
| *Route* |  |  |  |  |

[a] Problem domain dependent features
[b] User dependent features e.g. goals, motivation, lifestyles, and others
[c] Technology dependent features

**Figure 1.2:** The PSD model (from Lehto & Oinas-Kukkonen [15])

answers to questions and other forms of contribution to the Q&A database yield points, badges and levels. Novice users can become core users by improving their skills and using these to answer questions and share knowledge; in that sense, it shows characteristics of an online CoP. Studies show that the virtual reward system of StackOverflow has a positive impact on answering activity [18] [19].

Another major knowledge-sharing website is Reddit [20]. Here, users can post information or comment on other people's posts. Redditors give posts and comments points by 'upvoting' and 'downvoting'. In this context, subforums such as '/r/processing' are relevant: Processing coders can share programs they are proud of, and ask the community questions, similarly to StackOverflow. Research indicates found 'community building' to be the foremost reason to participate in online learning on Reddit, with the second being 'status-seeking' [21]. This is achieved through the 'Karma' system. So, we see that StackOverflow and Reddit both successfully engage users using a simple point system to indicate rank.

Techniques such as those mentioned are increasingly adopted in education, primarily by computer science/IT educators: the most popular gamification techniques are points, virtual badges (or achievements) and leaderboards; much like the StackOverflow and Reddit examples [22] [23]. Studies find that these techniques increase user activity in online academic or learning platforms (e.g. [24] [25] [26] [27]). Pitfalls in gamification mechanics for education include steering behaviour too much to rewarded activities [28], demotivating intrinsically motivated students ('overjustification') [29], and stimulating compulsive behaviours to the point that getting rewards is prioritised over high-quality results [30] [31]. Gamification mechanics are therefore recommended to be optional, and should not be relied on as the only motivation to learn; the use of gamifying elements does not make low-quality content engaging on its own [23].

These game-elements are examples of various principles of the PSD model, particularly *praise*, *rewards* (such as rewarding points), and *tunneling*, *tailoring* and *self-monitoring* by creating various reachable achievements that motivate the user to do specific actions.

The mentioned game mechanics also correspond to social persuasive design features such as *social comparison* and *competition*. There are pitfalls attached to these: some users may feel left out if they do not score as well as their peers and might even quit using the platform for this reason. These negative effects are recognized in various studies (e.g. [32] [33] [34]).

The Community of Practice is, of course, a social system. If we look beyond mostly competition-oriented game mechanics, there are other persuasive design features that support this.

Central to the CoP is *cooperation*. According to Oinas-Kukkonen and Harjumaa [16], a system that allows coorperation can motivate users to adopt a target attitude or behavior by leveraging human beings' natural drive to co-operate. *Social learning* means that users are more motivated to perform a behaviour if they see other doing it, which means the system should allow the observation of others doing the target behaviour.

The persuasive design features in the *credibility support* category are also relevant for the CoP. Even though a CoP rejects absolute authority for its own sake, learning from other members with more *expertise* is central to its structure. Especially in a CoP, it is therefor useful to make members' expertise explicit.

### 1.1.4 Recommendations for a CoP support tool design

Currently, the tool Atelier consists only of a place to upload and comment on code submitted by students. The research above resulted in recommendations that show how a tool can help foster the Community of Practice. These are labeled R1-5.

When looking at the C4P framework, the current design of Atelier incorporates *con-*

| | **Source** | **Recommendation** |
|---|---|---|
| **R1** | C4P Framework: Connections & Conversation | Allow conversation about code between all members of the community |
| **R2** | C4P Framework: Context | Let members display their work online, to provide context for (Q&A) discussions |
| **R3** | C4P: Content | Store information from discussions and code on the platform to create a shared repository of information by and for members |
| **R4** | PSD | Take inspiration from the PSD model and popular online communities to incorporate engagement and persuasion techniques in the system |
| **R5** | Various literature | Support the offline existence of the CoP in conjunction with the online tool |

**Table 1.1:** Recommendations for an online tool to foster the Community of Practice

*nections* (all students in the programming course are in Atelier) and *conversation* (conversation threads are available for every piece of submitted code). The latter can be extended by allowing conversations between all members of the CoP: students should be able to discuss code among each other (R1).

We can place the conversation in *context* by using a Q&A framework and allowing students to display their programs online (R2). By saving noteworthy discussions and programs, we create a repository of information for this *content*. A wiki is often used for these purposes and fits framework (R3).

Learning from online engagement techniques - notably, gamification - used on websites such as StackOverflow and Redddit, such techniques (e.g. achievement, point and rank systems) are suggested to promote the usage of the platform (R4). To further specify which persuasive design features are most useful in our context, we can use the PSD process model [16]). This should help keep both novice and expert users engaged with the system.

Literature also recommends to extend the offline CoP instead of trying to create a purely online CoP: as such, it is worth looking into technology that facilitates discussion within the classroom in conjunction with online discussion (R5). These recommendations are summarised in table 1.1. They will be incorporated in a proposal for an online tool in the next chapter.

## 1.2 Research Questions

To give a quick overview of the main research question and sub-questions they will be summarized below.

The main research question is:

**How can a digital tool enhance the quality of help seeking and giving, between students and teaching staff in Creative Technology programming tutorials?**

Subquestions:

- RQ1: What challenges are faced in tutorials concerning interaction between students and teachers/TAs?
- RQ2: How can these challenges be addressed in an improvement of the online education platforms?
- RQ3: What is the user acceptability of the design solution prototype?

## 1.3 Overview

There is a practical problem to solve; the challenges identified in the tutorials. The solution should be applicable in the practical context, rather than just being interesting in a 'laboratory' setting. Stakeholder needs - those of teachers, teaching assistants and students - have to be taken into account. As such, This research can be classified as design science, as described by Wieringa [35]. In design science, the practical problem to solve (RQ1) is part of the research. This means a broader starting point. Specifically, this thesis consists of validation research; there is no practical experience with the tool yet and any prediction of its future behaviour will have some degree of uncertainty [35].

The structure of the thesis is adapted from the Creative Technology Design Process [36]: it starts with a design question, then follows the pattern of Ideation, Specification, Realization and Evaluation. This process is summarized in Figure 1.3.

The overview of the rest of the report is as follows:

Chapter 2, Ideation, starts of with a brainstorm with a focus group TAs about their concerns about their work, and suggestions for a solution. Additionally, a questionnaire was given to all TAs. The results of the brainstorm and questionnaire form the basis of further literature research. This chapter concludes with recommendations for designing a solution.

Chapter 3, Specification, details the design of a digital tool using an analysis of the target audience and the technical context. The target users are illustrated using persona's. Combined with the results from chapter 2, this leads to the product idea. Requirements for the product are presented. The UI, UX and technical design is given in this chapter.

Chapter 4, Realization, describes the prototype made based on the design in the previous chapter. User test plans, how the test turned out, and the result of the user test are also described in this chapter.

Chapter 5, Interpretation, starts by discussing the test results from the previous chapter. Which requirements were and were not met is also described in this chapter. A look back is taken at persona's that were introduced in chapter 3.

Chapter 6, Discussion, describes the implication of the results from the previous chapter. The limitations of the research are also given. Recommendations for meeting unmet requirements, and recommendations for further research are presented.

Finally, the Conclusion to this thesis is chapter 7. The main research questions, as well as the subquestions as given in section 1.2, are answered in this chapter.

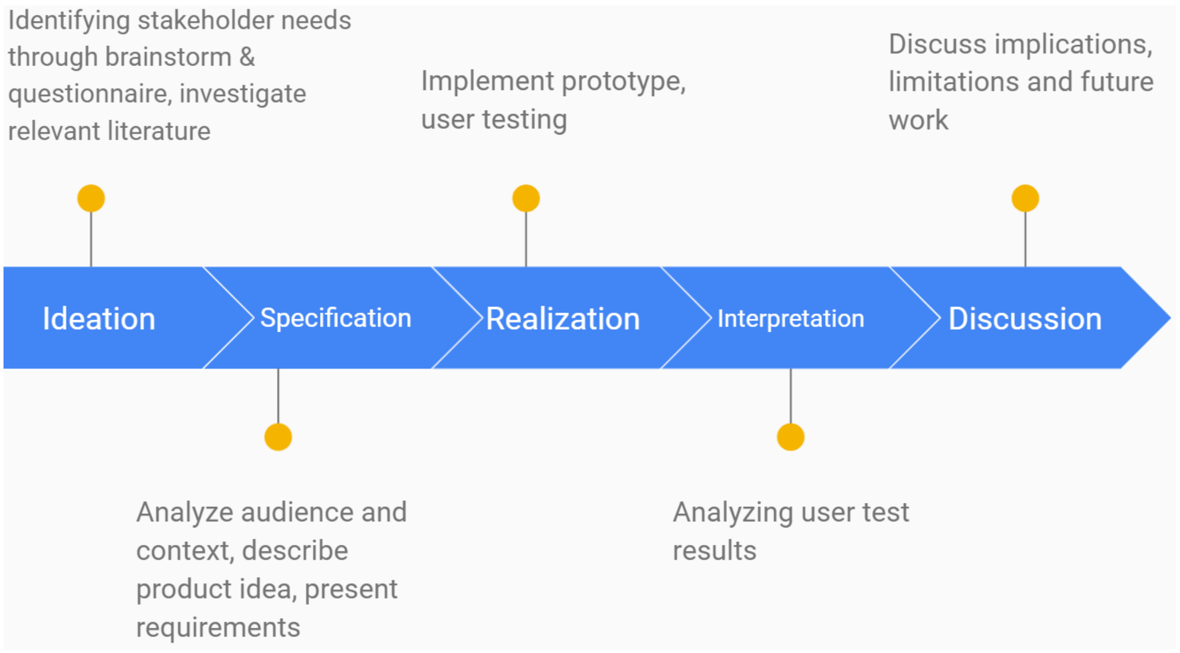

Identifying stakeholder needs through brainstorm & questionnaire, investigate relevant literature
Implement prototype, user testing
Discuss implications, limitations and future work
Ideation
Specification
Realization
Interpretation
Discussion
Analyze audience and context, describe product idea, present requirements
Analyzing user test results

**Figure 1.3:** Design process of this thesis

# Chapter 2

# Ideation

In the previous chapter, recommendations were made for creating a platform on which a Community of Practice can exist (CoP). Some of these recommendations are already present in the current Atelier design (e.g. uploading and discussing student's code).

In this chapter, we use the recommendations to find a design direction for a tool to enhance Atelier. In order to answer RQ1 (What challenges are faced in tutorials concerning interaction between students and teachers/TAs?), a brainstorm was organized with some TAs (section 2.1). In addition, a questionnaire was issued to all current TAs (section 2.2. The results of these are discussed in this chapter.

Based on the topics raised in the questionnaire and brainstorm, additional literature research is conducted (section 2.3), in order to give a direction moving forward.

## 2.1 Brainstorm

A brainstorm with a focus group of teaching assistants (TAs) was organised to learn their views. Five TAs (all male, average age 24) attended, all of whom had had at least multiple modules of experience. These TAs were recruited through TA chat groups. The brainstorm was held offline, in a tutorial classroom[1]. Notes were taken by the researcher.

This brainstorm consisted of two parts: first, TAs were asked about their concerns about their job, second, they were briefed on the research of chapter 1 as inspiration, and then asked for thoughts and suggestions.

[1]This brainstorm was held when the Covid-19 preventive measures were not in effect yet

TAs were given a few minutes to write down **problems/concerns**. Then, they had the opportunity to share and discuss these. The following issues came up:

- TAs spend much time answering the same questions
- Students have problems identifying the cause of programming problems, and as such have a hard time asking the right questions to TAs.
- Some students do not ask questions at all when stuck
- Students isolate themselves from the group by not showing up.
  - Students who find the course difficult escape the course temporarily, but do not have the skills to individually reach the level required to pass the course
  - Students who find the course easy, feel that showing up is a waste of time. They often overestimate their skills and end up falling of the wagon in later weeks. Also, these students are often needed to support their peers, but now cannot do so.

The background of this project, including Atelier, the CoP and the recommendations of chapter 1 (table 1.1), were then explained to the TAs. With those as inspiration, the TAs were asked for suggestions. These suggestions were written down and shown below.

The suggestions given by the TAs were mostly based around *question asking procedures*. A first suggestion given by the group was to recreate a StackOverflow-like environment for students to ask questions, which included gamification techniques detailed below - very much corresponding to the ideas of chapter 1.

Suggestions to **motivate students to use the system** were:

- Make it obligatory to ask questions through the online system first
- Using a Reddit-like voting system, teachers should answer the most popular questions in class in a mini-lecture
- Answering questions properly count towards a bonus for your grade
- Showcasing an individual's strong points through a 'skill-tree' system (and not a comparative ranking)
- Taking gamification further using elements such as:
  - Collect Pokemon-like creatures (*Rubber Duckies were suggested, after a famous programming phenomenon*) through use of the system

  - Earning points for using the system, that can be traded for real-life rewards
  - Grouping students into large teams, where collaboration leads to higher scores and one team wins at the end of the course
  - Having students check-in at tutorial locations using QR/RFID tags in exchange for points

**Pitfalls** of directing questions to a forum identified were:

- TAs do not want to give up face-to-face interaction
- TAs do not want to become first and foremost a forum moderator
- Increasing the odds of students skipping tutorials in favour of using the forum
- The existence of a forum does not always lead to the class populating it.
  - For many courses, a forum exists but is not used. A notable exception, where it was actively used, was a course where it was obligatory to ask any question through the forum, which was disliked by some.
- Students putting entire programs out for others to debug, learning nothing in the process themselves
- Students being unsure of where (online or offline) to ask which question and to whom (other students or TAs), leading to questions not being asked at all

### 2.1.1 Conclusion

The suggestions from this brainstorm session will be taken into account in further design. We can also derive some recommendations from it:

- The tool should focus on making time between students and TAs more useful, instead of completely moving discussions to an online environment
- Course forums are *not* being populated without making them obligatory
- The tool should be useful within the tutorial, so as to not encourage students to skip tutorials
- Gamification elements are favoured by TAs, but they should away from individual competition

## 2.2 TA Questionnaire

After the brainstorm, an online (Google Forms) questionnaire about help seeking and giving was issued to a chat group of all (18) of the TAs active at that point in the module. The aim of the questionnaire was to find out

- Whether interactions are initiated by students or by TAs, in order to determine more about the workflow when giving or seeking help
- What help requests and answers look like
- What is enjoying or frustrating to TAs

The full questionnaire can be found in appendix A.

10 TAs filled in the questionnaire.

### Interaction initiation

First, TAs were asked "When helping students, (approximately) what percentage of the time was this [*started by a student's help request / started by a TAs initiative / started by another TA redirecting their student to you*]?". For each of these, they could freely fill in an estimated percentage, as such it was possible for these to not fully add up to 100%.

- The average percentage of interaction indicated as **started by a student's help request**: 70.5%
- The average percentage of interaction indicated as **started by a TAs initiative**: 18.5%
- The average percentage of interaction indicated as **started by another TA redirecting their student to you**: 15.5%

### Help request

Using a 5-point Likert scale, TAs were asked to rate answers to the question *"What does a help request from a student typically first look like?"*. Every answer was an example statement of what a help request can first look like. They could be rated 'never' to 'very frequently'.

These statements, with the results, can be seen in Figure 2.1.

For evaluation, the answers are converted to scores (1 for 'never' and 5 for 'very frequently') to calculate an average. In the figures below, colours indicate the share of people voting for each of the 5 categories (dark red for 'never' and dark blue for 'very frequently'). The figures also show the average score for each answer, overlaid on the stacked bar charts with the x-axis ranging from 1 - 5.

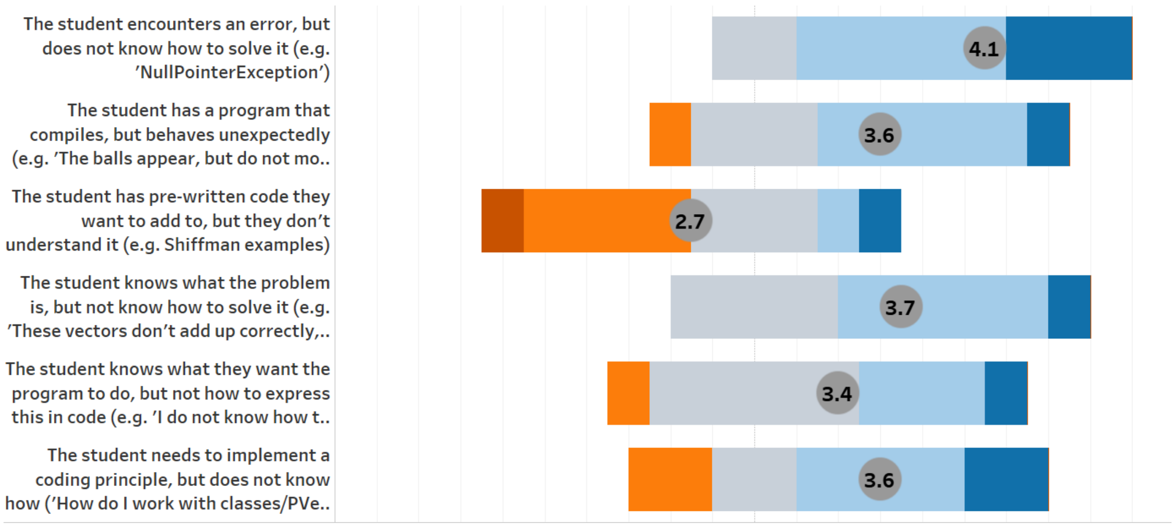


**Figure 2.1:** What does a help request from a student typically first look like?

Notably, we see that the most common help request is "The student **encounters an error**, but does not know how to solve it" (averaging on 4.1 or 'frequently'). Other frequently encountered help requests are "The student knows what the problem is, but does not know **how to solve it**" (averaging on 3.7) as well as "The student has a program that compiles, but **behaves unexpectedly**" and "The student needs to implement a **coding principle, but does not know how**" (both averaging on 3.6).

In addition, TAs were asked to rate answers to the question "*What details do students provide to their problem statements on their own initiative?*".

The results can be seen in Figure 2.2.

Frequently given details are **"Actual and expected results"** (3.6), **"Goal statement"** (3.8), **"Error messages"** (3.7), **"Actual and expected results"** (3.6) and **"The specific part of the code that seems problematic"** (3.6).

Scoring on average negative are **"Description or relevant documentation found"** (1.7) and **"Description of what they Googled"** (1.7). While we can not definitely say that this means that students rarely look at the documentation or Google their

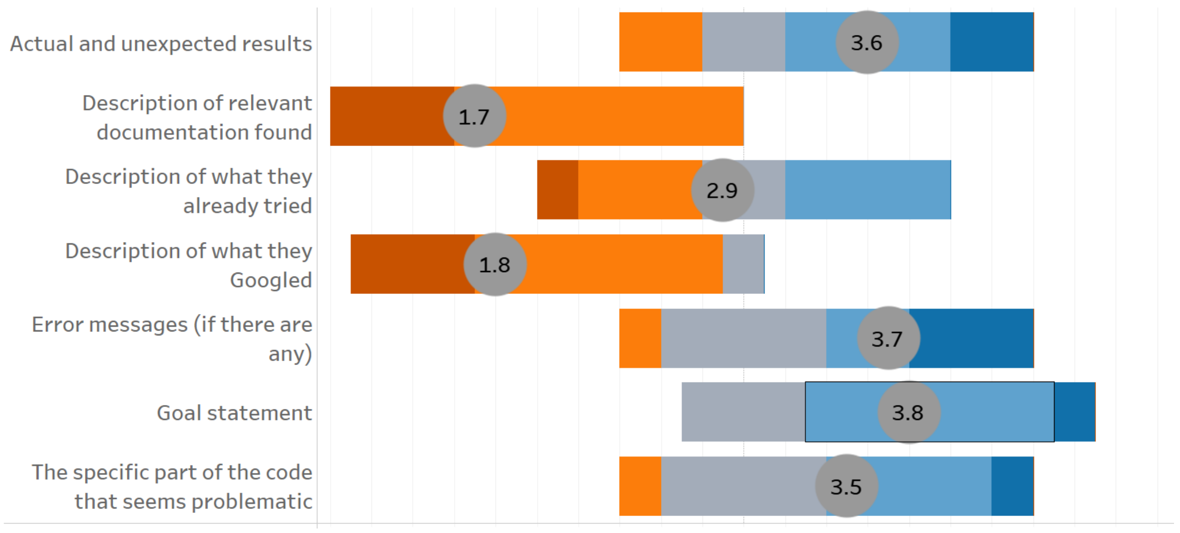


**Figure 2.2:** What details do students provide to their problem statements *on their own initiative*?

problem, combined with TA statements, this seems to be the case.

### TA help provided

Next, TAs were asked to rate answers to the question "*What details do you ask for?*", which had the same problem details as the previous question.

The results can be seen in Figure 2.3.

We find that **all these details are asked for by TAs**, all averaging around 4 ("frequently").

Additionally, TAs were asked to rate answers to the question "*Which of the below does a help session further entail?*"

The results can be seen in Figure 2.4.

The answers indicated to be most frequently part of a help session are **"You explain the cause of the problem"** (4.4), **"You ask the student questions to get to the problem statement"** and **"You go through the code yourself, in search of the problem"** (both on 4.0). Also a frequent even it that **"While talking/explaining, the student figures out the solution by themself"** (3.7).

In this survey, participants could freely fill in any additional steps that TAs often take. Among the answers are **helping the student get to the documentation** or **how to find information on Google**.

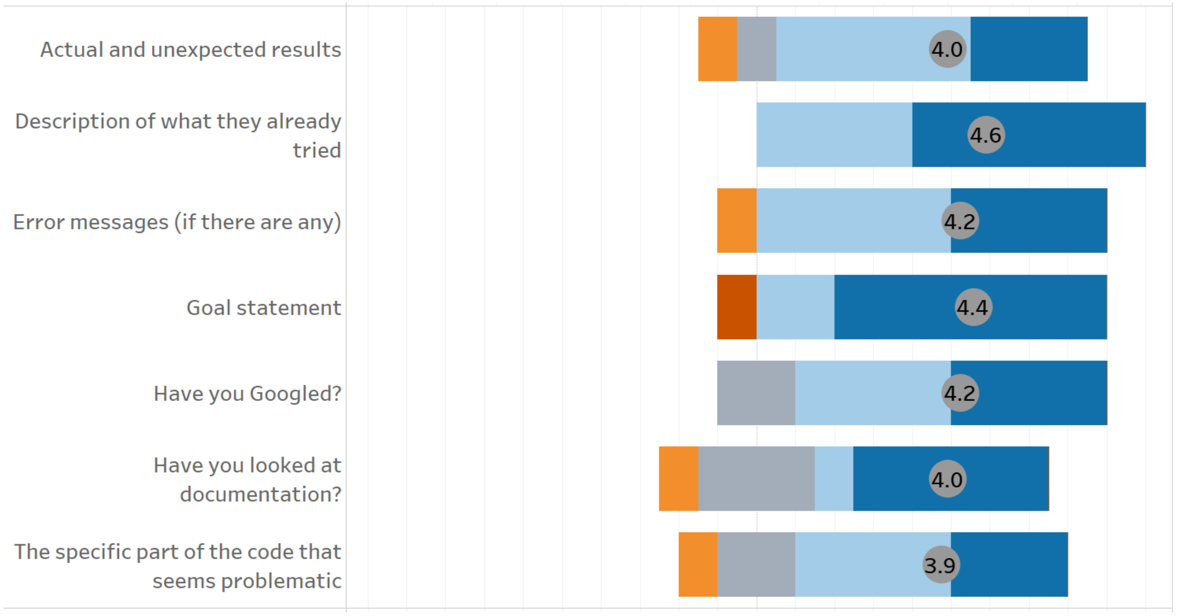


**Figure 2.3:** What details do you ask for?

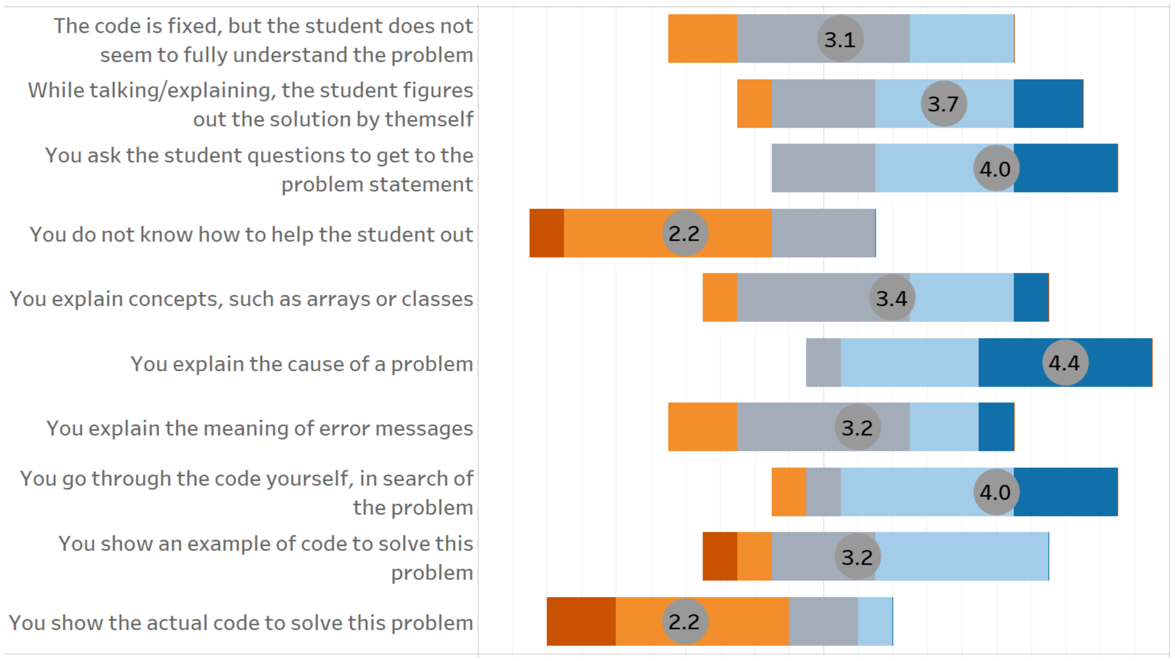


**Figure 2.4:** Which of the below does a help session further entail?

### Having a good or bad time as a TA

The TAs were asked: *"Can you describe what might make a tutorial session an enjoyable one for you as TA?"*. This was an open question.

The open answers were labeled and frequently mentioned factors were counted in

| **Factor described** | **Amount of times mentioned by TAs** |
|---|---|
| Pro-Active Students | 5 |
| Well-prepared questions from students | 2 |
| Interaction with students | 2 |
| Giving valuable help | 2 |
| The right workload | 4 |

**Table 2.1:** Factors that make tutorials enjoyable for TAs

| **Factor described** | **Amount of times mentioned by TAs** |
|---|---|
| Students lack interest | 5 |
| Students are passive | 2 |
| Low workload | 2 |
| High workload | 2 |
| Students lack knowledge | 4 |

**Table 2.2:** Factors that make tutorials frustrating for TAs

table 2.1.

Similarly, the TAs were asked *"Can you describe what might make a tutorial session frustrating for you as TA?"*.

The open answers were labeled and frequently mentioned factors were counted in table 2.2.

### 2.2.1 Conclusion

From the answers to the questionnaire, we can conclude that

- Interactions between students and TAs are typically started by a student asking for help, instead of a TA approaching them. This also seems to be favourable, as pro-active students are found to make TA time more enjoyable.
- There are various types of help requests frequently encountered. The most common of these are errors, problems the student does not know how to solve, unexpected behaviour of the program, and students not understanding a coding principle.
- In a list of problem statement details, we found that while much information is given by the students, they rarely describe trying to find in documentation or use Google.

- All problem statement details, including documentation or Googling efforts, are liked to be asked by TAs.
- Pro-activeness of students, and having well-prepared questions, make tutorials more enjoyable for TAs. The workload is also important; a high workload is a problem that is encountered, but a low workload (either due to low attendance or passive student) is not appreciated either. Valuable interaction with students is valued.

From these results, combined with the statements given during the brainstorm, we may direct our efforts in improving the online tooling for the courses towards

- Nudging students towards being pro-active during tutorials
- Decreasing high workload if necessary
- A tool that encourages TA/student interaction, rather than taking away from it/digitalizing it away
- A tool that increases the quality of these interactions

## 2.3 Literature

To investigate the conclusions above themes such as help seeking and giving, interactions with TAs and fellow students, and self-reliance (such as to decrease high workload), these were researched in literature. The literature research will lead to conclusions that are helpful in designing a digital tool to be of assistance. This helps answer RQ2, *How can these challenges be addressed in an improvement of the online education platforms?*

If students are better self-reliant problem solvers (debuggers), individually or within the group, this could have the effect of both decreasing TA workload and increasing the quality of the interactions between TAs and students.

In a study on novice Java programming students, Ahmadzadeh et al. [37] found that good debuggers tend to make good programmers, while less than half of good programmers are good debuggers. Examples like these tell us that debugging is a skill in itself, and one that can can improve the skills of programming students at all levels. Keys to becoming a good programmer, according to them, are a good understanding of the program implementation, as well as the ability to read and understand other people's code.

For example, reading and understanding other people's code can be trained using pair programming. We have previously mentioned the preference for programming with a network of friends [38], as well as the success of peer assessment strategies [39] [40] [41] [42]. Pair programmers are also found to ask fewer questions than solo programmers [43]. Leveraging the power of other people's knowledge, Hartmann et al. [44] created a tool that (automatically and within the IDE) would show other programmers solutions to problems encountered while programming.

During tutorials, students are able to use help from their peers, both fellow students and teaching staff. Seeking help would be one of the first steps to learn the advantages of using peers to solve problems. In fact, Razzaq et al. found that students learn reliably more when asking for help themselves, rather than waiting for it [45].

However, seeking help can feel threatening to students, leading to seeking less help when it is needed [46] [47], for example if it is seen as an admission of failure or if it requires much effort to effectively seek help [48]. A digital tool might ease the process of asking for help. For example, in a study on asking for help while programming, Price et al. [49] found that to their participants, a human tutor seemed more trustworthy, perceptive, and interpretable, while a computer tutor seemed more accessible and less threatening to a student's sense of independence. Roll et al. [50] recognise that there is a lack of computer tutors that teach metacognition ('What is my knowledge gap' and 'How do I overcome it?') instead of knowledge. Sillito et al. believe that there are missed opportunities for tools to make use of the larger context to help programmers more effectively scope their questions [51].

When not knowing how to *effectively* ask for help - and students often do not [52] [50] - students may avoid help when they need it, which can impede learning [53]. Nelimarkka & Hellas advise creating a wireframe for social help-seeking, to help students ask questions that can help to ensure they can engage in a deeper course-wise discussion. Students could use this to ask more detailed questions, connecting the exercise and problem to a wider learning context [54].

In a study on adaptive help seeking, i.e. asking for the help needed to learn independently, not simply to obtain the correct answer, Newman [55] identifies knowing how to ask a question that yields precisely the information that is needed, contributes to adaptive help seeking, and it is advised that teachers give opportunities to learn this skill.

When studying Q&A sites for programming, Nasehi et al. found that both including code examples for the erroneous program and a good explanation to go along with it, are important to obtaining well-received answers. Making use of question constraints avoids shortcomings in solutions [56].

There is evidence that students who do not collaborate, but still cooperate, improve their understanding of their problems by simply explaining them to each other, or even simply to themselves [57] [58]. A famous example in programming, brought up during the TA brainstorm session, is explaining a program or problem line-by-line to a rubber duck [59] (or perhaps a cardboard cutout of a dog [60]). StackExchange referenced this method by creating an online rubber duck to talk to, prompting users to think twice before asking a question there. This duck would do nothing, except respond with 'quack' [61].

In another example, Aleven and Koedinger [62] created intelligent, instructional software to which students could explain their reasoning in solving mathematical problems. They found that students who explained their steps to it, learned with greater understanding than those who did not. We call this strategy self-explaining [57].

## 2.4 Conclusion

During the TA brainstorm, it became apparent that there are some concerns in help seeking and giving during tutorials. Students have problems identifying the cause of programming problems, and as such have a hard time asking the right questions to TAs, some students do not ask questions at all or do not show up to tutorials. TAs favour a digital tool that makes real-life interaction between students and TAs during tutorials more useful, rather than moving them online. A follow-up questionnaire to all TAs (also those not present during the brainstorm), supported these sentiments. Additionally, pro-activeness of students was sometimes found lacking.

From the literature research, we can then conclude that

- Working with peers, such as learning how to read and understand other people's code, pair programming or help seeking, contributes to learning programming.
- Seeking help can be difficult for some students, both because the process feels threatening, and because they do not know how to effectively ask for help. This can impede learning.
- A computer-based solution can assist in both making the process less threatening, and guiding the student in effective help seeking. Learning how to ask the right questions is key. While there are studies into the types of questions programmers asked, and how tools deal with these questions (e.g. [63] [51]), literature gives us little information on what a programming-related question should consist of in order to be an effective one in getting to the answer.
- Self-talk is a strategy that can be incorporated as well to support learning.

# Chapter 3

# Specification

The previous chapter presented a brainstorm and questionnaire to investigate opportunities where a digital tool can help improve the tutorial session from the TA's perspectives. The literature review discussed more about how these issues can be resolved from a theoretical point of view, using strategies that are focused on the student perspective.

In this chapter, we specify the design of a digital tool using an analysis of the target audience and the technical context. We specify requirements and make a design based on this.

## 3.1 Context analysis

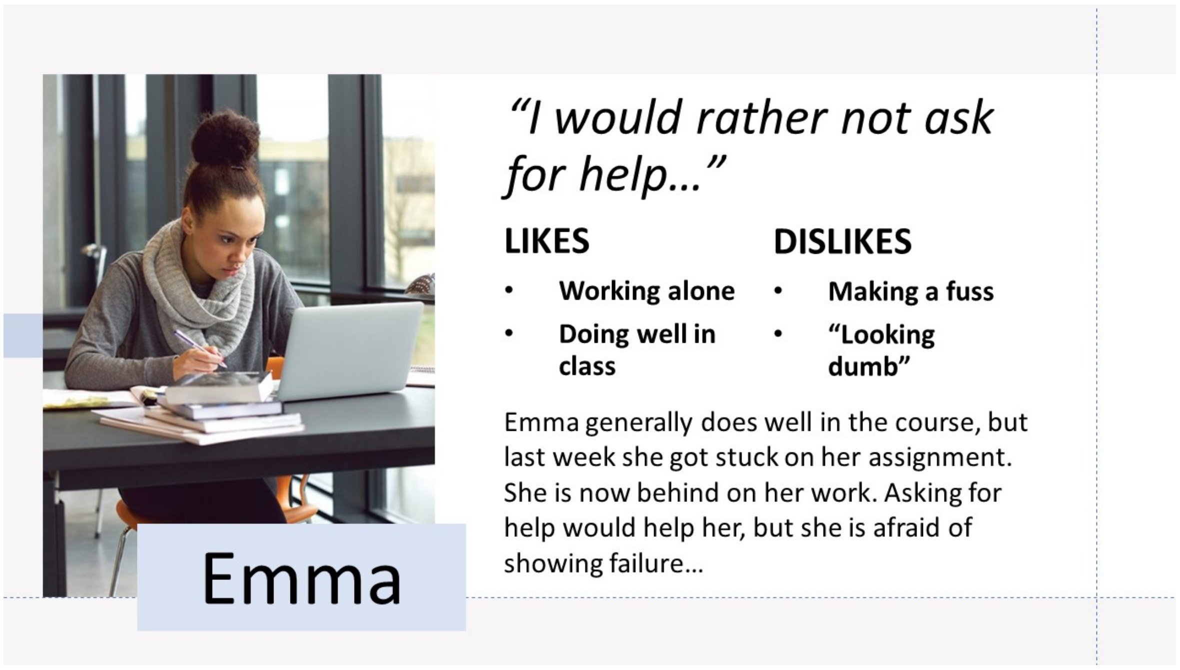
### 3.1.1 Target audience - Personas

In this section, personas based on the target group and literature review are used to illustrate what different types of students and TAs there are to design for. In chapter 5, we will check back on these 'characters', to see if our solution would benefit them.

#### Students

From the literature review in section 2.3, we know that an obstacle to asking questions for a student is being afraid that they show failure to those who might help them. This principle is the basis for the persona of Emma, shown in figure 3.1.

In the brainstorm and questionnaire, TAs have named students who do not show a proactive attitude to problem solving as a problem in tutorials. Students like these are not necessarily bad students: it can be understandable to behave this way if programming is not a favourite course for them. This forms the basis for the persona of John, see figure 3.2.



*"I would rather not ask for help..."*

**LIKES**

- **Working alone**
- **Doing well in class**

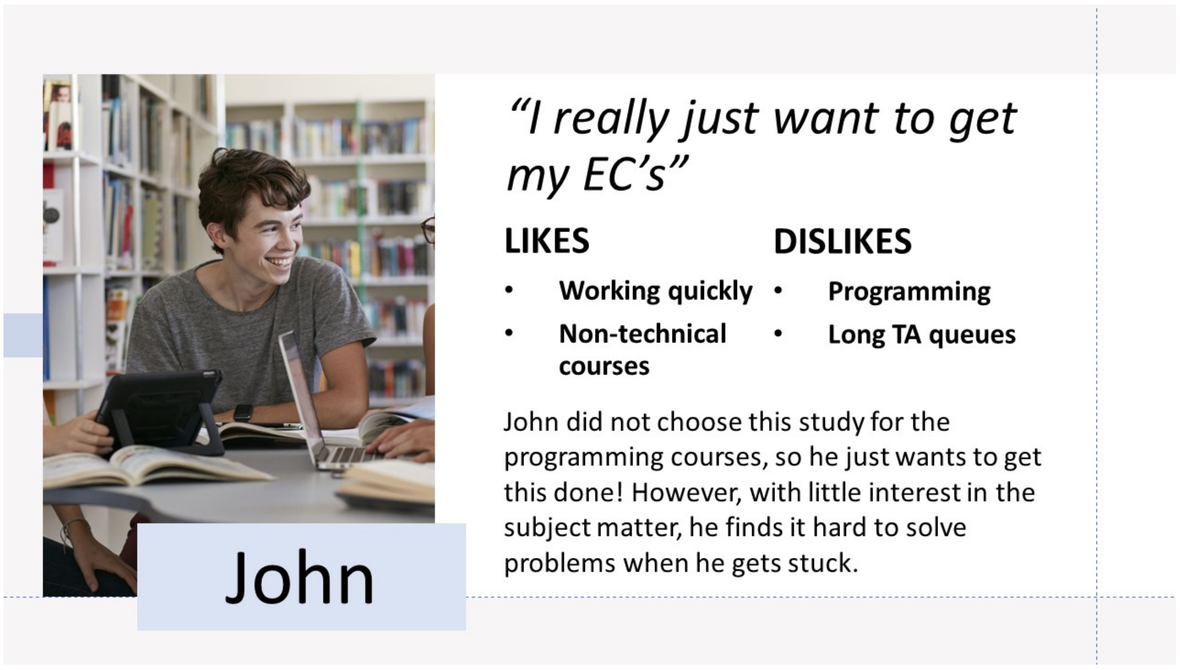
**DISLIKES**

- **Making a fuss**
- **"Looking dumb"**

Emma generally does well in the course, but last week she got stuck on her assignment. She is now behind on her work. Asking for help would help her, but she is afraid of showing failure...

Emma

**Figure 3.1:** Student persona 1: "Emma"



*"I really just want to get my EC's"*

**LIKES**

- **Working quickly**
- **Non-technical courses**

**DISLIKES**

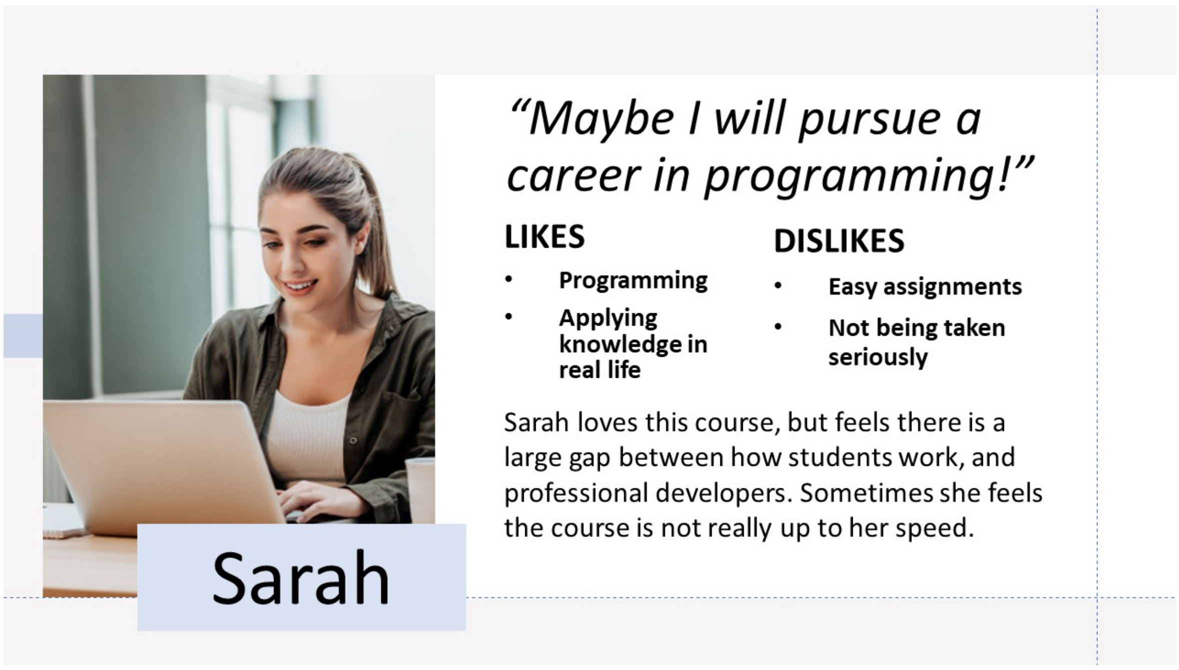
- **Programming**
- **Long TA queues**

John did not choose this study for the programming courses, so he just wants to get this done! However, with little interest in the subject matter, he finds it hard to solve problems when he gets stuck.

John

**Figure 3.2:** Student persona 2: "John"

A student whose behaviour would be approved of by TAs is illustrated by the persona of Sarah (figure 3.3). She actively wants to learn about programming. The tool we design should be useful for all students, including those like Sarah.



*"Maybe I will pursue a career in programming!"*

**LIKES**

- Programming
- Applying knowledge in real life

**DISLIKES**

- Easy assignments
- Not being taken seriously

Sarah loves this course, but feels there is a large gap between how students work, and professional developers. Sometimes she feels the course is not really up to her speed.

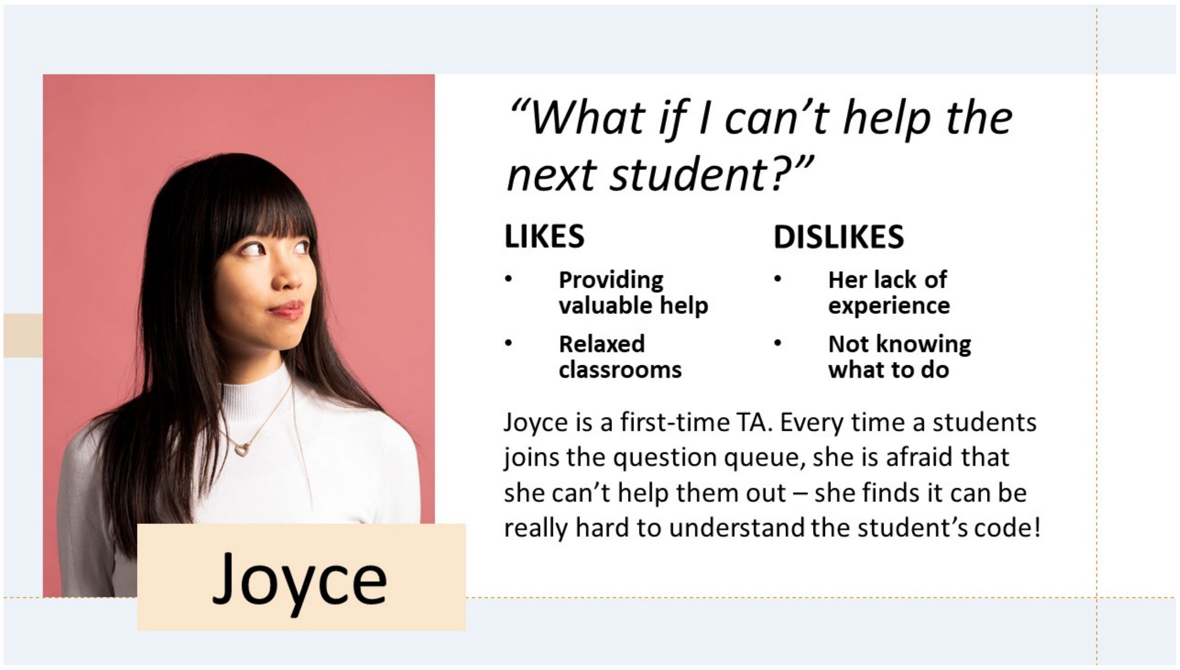
Sarah

**Figure 3.3:** Student persona 3: "Sarah"

#### TAs

When a student has difficulty asking for help, this impacts the TAs too. For example, the persona of Joyce (figure 3.4) shows a TA that is new to the course. These TAs can be especially nervous about questions she is not able to answer, and this is more likely if a student cannot communicate their problem well.

In the questionnaire and brainstorm, we found that workloads that are both too high and too low are a problem for TAs. These are not opposing opinions, as shown by the persona of Tim (figure 3.5). Tim is experienced and values quality time with his students, which is hard if they are absent during the module and all show up at the end.



*"What if I can't help the next student?"*

**LIKES**

- **Providing valuable help**
- **Relaxed classrooms**

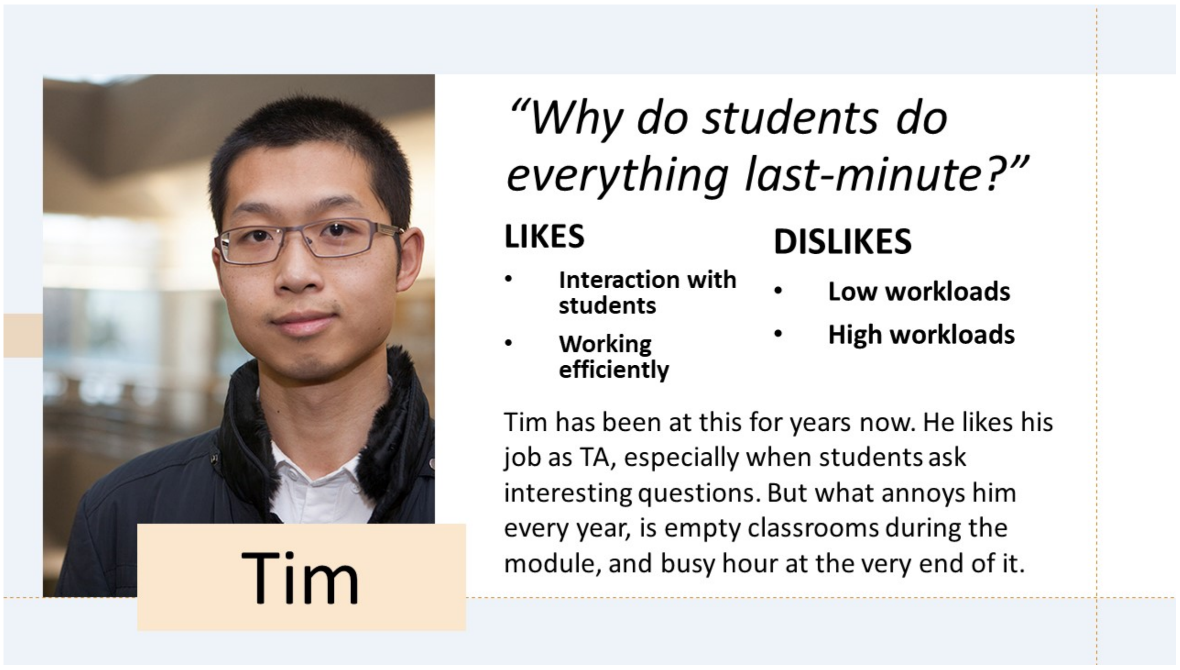
**DISLIKES**

- **Her lack of experience**
- **Not knowing what to do**

Joyce is a first-time TA. Every time a students joins the question queue, she is afraid that she can't help them out – she finds it can be really hard to understand the student's code!

Joyce

**Figure 3.4:** TA persona 1: "Joyce"



*"Why do students do everything last-minute?"*

**LIKES**

- **Interaction with students**
- **Working efficiently**

**DISLIKES**

- **Low workloads**
- **High workloads**

Tim has been at this for years now. He likes his job as TA, especially when students ask interesting questions. But what annoys him every year, is empty classrooms during the module, and busy hour at the very end of it.

Tim

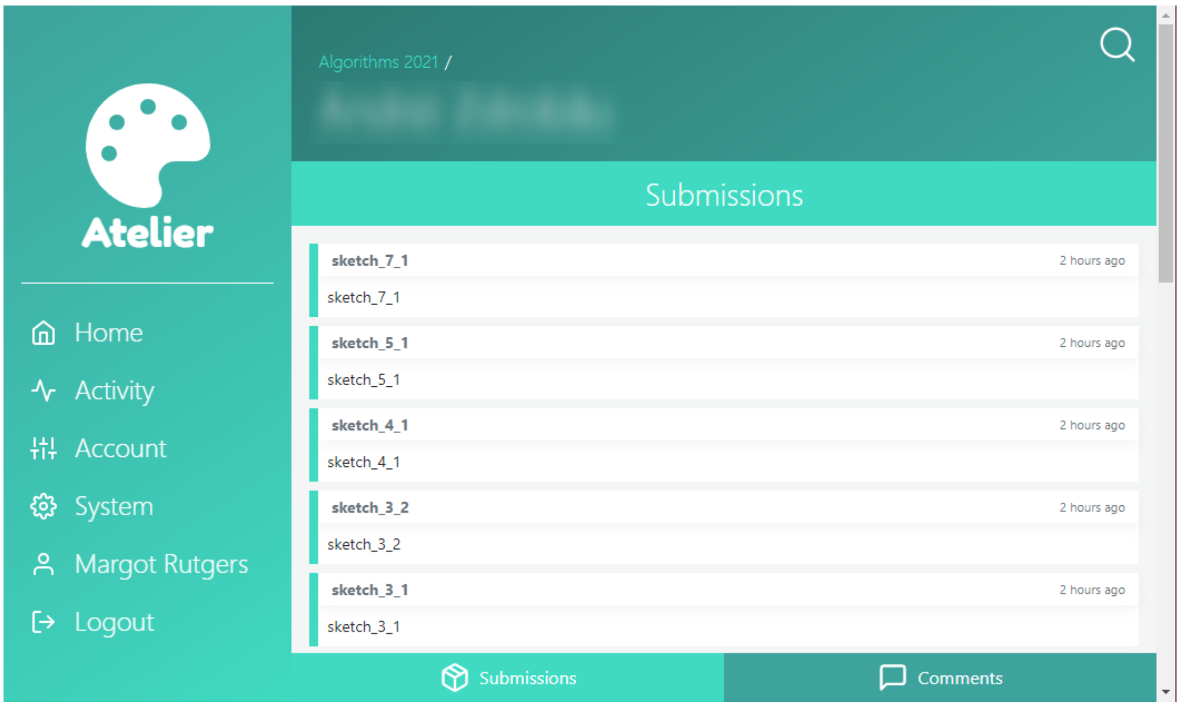
**Figure 3.5:** TA persona 2: "Tim"

### 3.1.2 Software context

It would be beneficial for the tool to be designed to work with the current software that is used in the programming courses. This would lower the threshold for users to to adapt it in their work or study; within a familiar environment it will be easier to learn, and require less effort to get to the tool. Typically, the following are used:

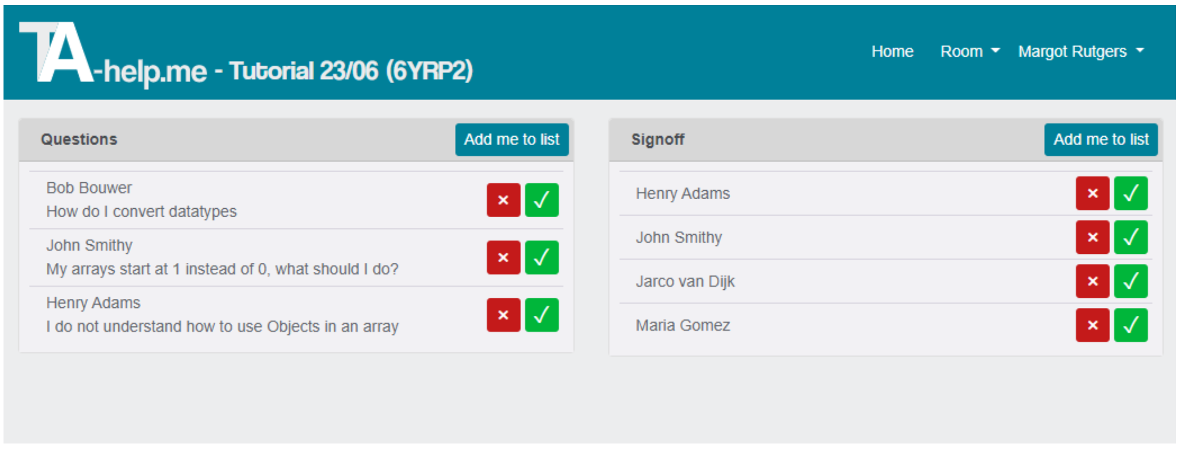
- **TA-Help.me** (or similar systems, such as Horus) - Here, a student can sign up to a digital queue to request help from TAs. In TA-Help.me, it can be made obligatory to include a question when signing up. See figure 3.7.
- **Atelier** - A website where students can upload and view their code so that they may discuss it with fellow students or the TAs whose help they have requested. The Atelier project is described in chapter 1. See figure 3.6. Uploading to Atelier is sometimes made mandatory after asking a question on TA-Help.me, so that the TAs can quickly find the code they are going to work with.

At the time of this research, classes were held online due to the ongoing pandemic. Videocalling tools such as MS Teams and BigBlueButton were used for this. However, this is not the normal learning situation, and these tools are therefore not explicitly taken into account when designing.



sketch_7_1 2 hours ago
sketch_7_1
sketch_5_1 2 hours ago
sketch_5_1
sketch_4_1 2 hours ago
sketch_4_1
sketch_3_2 2 hours ago
sketch_3_2
sketch_3_1 2 hours ago
sketch_3_1

**Figure 3.6:** A student's submissions on Atelier



Bob Bouwer
How do I convert datatypes

John Smithy
My arrays start at 1 instead of 0, what should I do?

Henry Adams
I do not understand how to use Objects in an array

Henry Adams
John Smithy
Jarco van Dijk
Maria Gomez

**Figure 3.7:** Queues for questions and signoff on TA-Help.me

## 3.2 Product idea

As presented in section 2.4, there are opportunities for improvement in regard to help seeking and question asking. In courses, TA-Help.me is where this is done. Therefore, it would make sense to design/improve a tool within TA-Help.me.

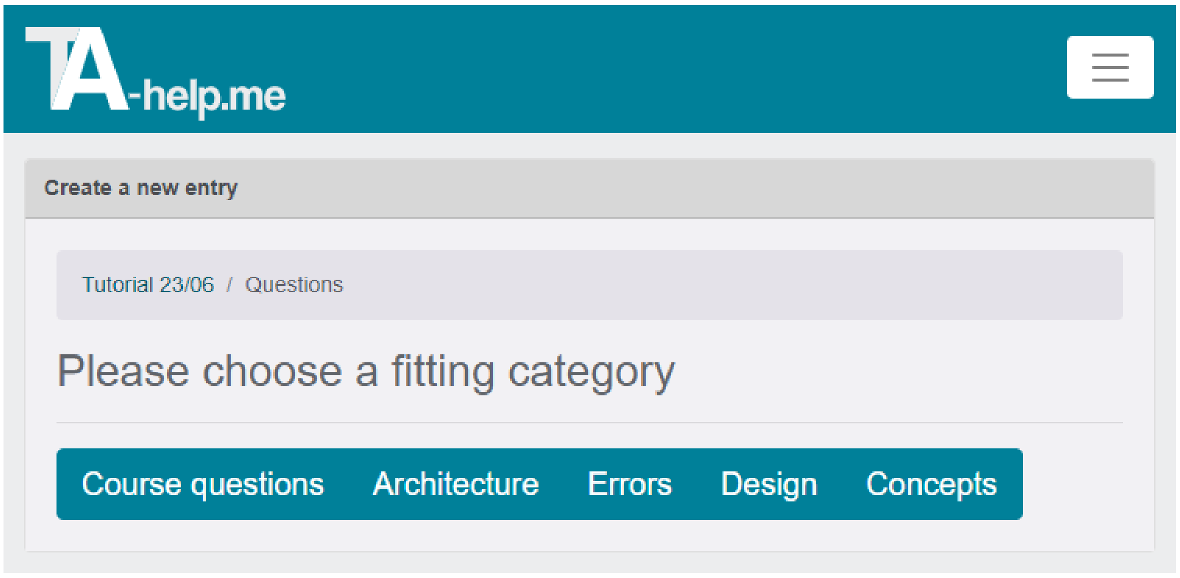


**Figure 3.8:** Picking a question category on TA-Help.me

When students seek help via TA-Help.me, they are taken to a screen where they to choose a category for their question (Figure 3.8). The categories can be chosen and changed by the teaching staff for every tutorial. Then, they are taken to the next screen where they can ask their question (Figure 3.9).

In an effort to improve the quality of help-seeking and giving on TA-Help.me, research by Heleen Kok [64] resulted in question tips displayed next to the field where questions can be typed, as can be seen in Figure 3.9. Students typing out their questions and choosing categories were seen as added value by TAs, however the tool did not improve the quality of questions asked. She also found that the tool did not cause TAs to better guide students towards good quality questions. In fact, the suggestions were quite easy to ignore. Therefore, she proposed the following questions for future work:

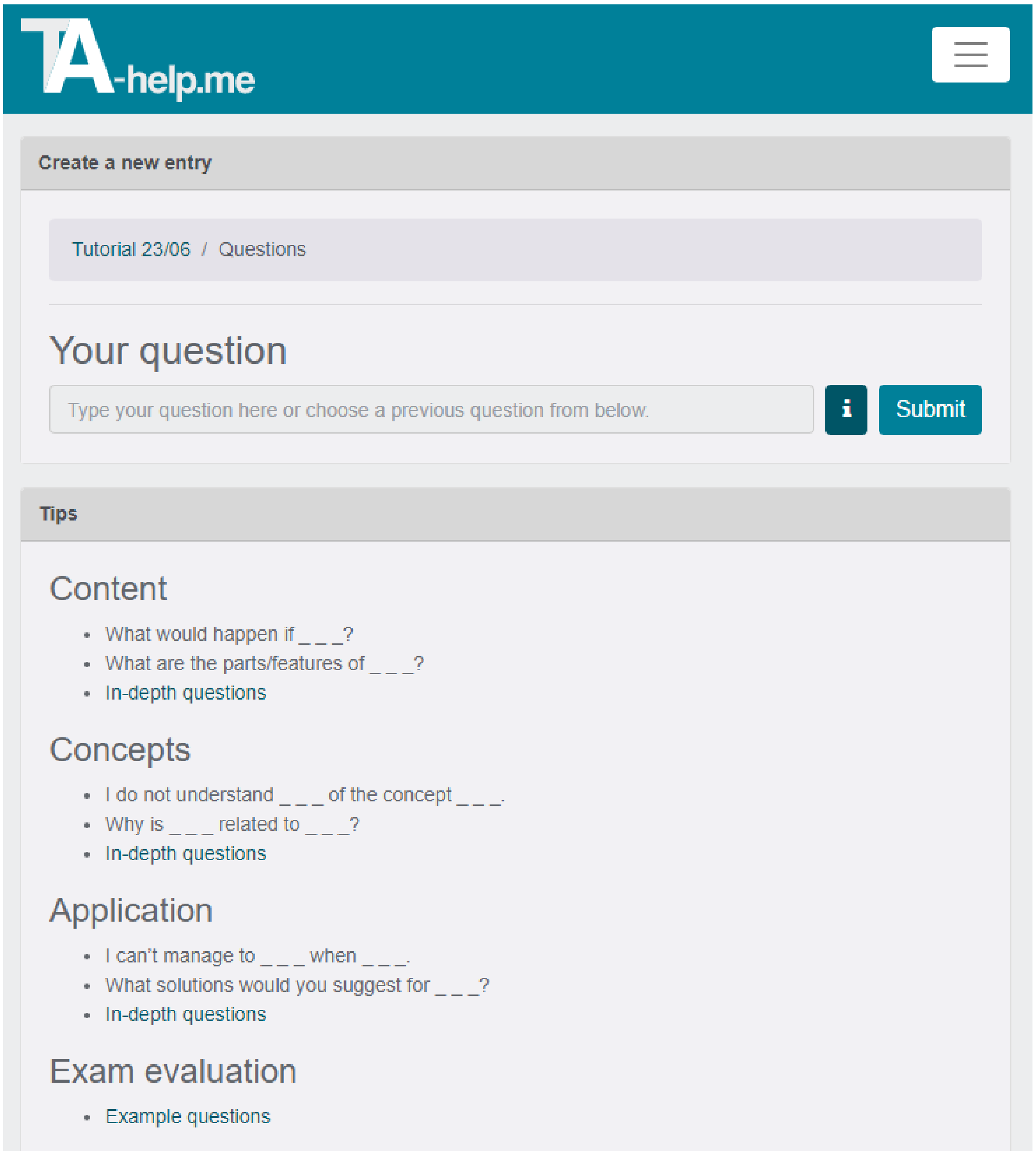
- How can a TEL [technology enhanced learning] tool intuitively steer students to formulate specific questions?
- How should TAs moderate question asking to increase the quality of the questions the students ask?

Currently, Kok's addition to TA-Help.me is easy to ignore, which may in part be a reason for it not improving the quality of questions. Therefore, a solution might be a feature that students need to *directly interact* with, to steer them towards formulating specific questions. A form would be the simplest way to directly steer students. However, it is very static, and different question types may require different solutions. A question on a program that is stuck requires information such as any error messages and related code, while a question about an upcoming deadline requires none of these.

In e-learning, **conversational agents** have been found a natural and practical interface, that is capable of tailoring the interaction to an individual [65]. There are various successful examples of conversational agents acting as digital tutors (e.g. [66], [67]). One example where a conversational agent adds value over simple means such as forms or menus, is the Geometry Explanation Tutor, which engages students in a natural language dialogue to help them state good explanations for problem-solving steps [68]. Students who explained problem-solving steps in a dialogue with the conversational agent did learn better to state explanations than those who were helped by a menu [69]. Mayer [70] found that people learn more deeply when the words in a multimedia presentation are in conversational style rather than formal style. To satisfy Kok's proposal of an *intuitive* tool, conversational agents in e-learning have the key feature of being a natural method of communication in e-learning; dialogue is a medium through which nearly all learners are familiar with expressing themselves [65]. They can also elicit self-explanation [67], a strategy found to be of use from the literature research in section 2.3.

So, for us, a conversational agent can

- Fully guide students step-by-step to ask quality questions



Create a new entry

Tutorial 23/06 / Questions

Your question

Type your question here or choose a previous question from below.

Tips

Content

- What would happen if _ _ _?
- What are the parts/features of _ _ _?
- In-depth questions

Concepts

- I do not understand _ _ _ of the concept _ _ _.
- Why is _ _ _ related to _ _ _?
- In-depth questions

Application

- I can't manage to _ _ _ when _ _ _.
- What solutions would you suggest for _ _ _?
- In-depth questions

Exam evaluation

- Example questions

**Figure 3.9:** Asking a question on TA-Help.me

- Make the process of asking questions less threatening (recommended from section 2.4), as computer tutors are seen as less threatening than real tutors, and leads the student from a computer to a real tutor (section 2.3), and because students learn *how* to effectively ask for help.
- Allow aspects of explaining and self-talk (recommended from section 2.4)
- Be built into the existing platform TA-Help.me, thus not adding yet another tool

for students and TAs to use. It can easily be a mandatory part of asking a question.

Specifically, we can call or conversational agent a **chatbot**. This is a class of conversational agent that receives natural language as input, and gives a conversational response as output, as well as having the ability to perform (simple) tasks [71].

We propose the concept of Duckbot (named after the programmer self-talk method ’Rubber Ducking’, aka explaining their problems to a rubber duck [59]). Duckbot will ask the student to refine questions to include important details (from now on called Problem Statement Details) that will both help the TA understand the problem and work with it more efficiently, but more importantly help the student understand their own problem better (an element of self-talk).

The owner of TA-Help.me agreed to work with Duckbot, however they made the constraint to change as little as possible out TA-Help.me to do so. Redirections to new pages are OK, changing the layout of TA-Help.me itself is not.

### 3.2.1 Product requirements/MoSCoW

Based on the chatbot idea ’Duckbot’ proposed in section 3.2, we make product requirements. These are sorted using a MoSCoW analysis [72]. This categorises requirements into four categories, ’Must have’, ’Should have’, ’Could have’ and ’Won’t have’. These requirements can be found in table 3.1. In chapter 5, we will check whether our solution has fulfilled the requirements.

### 3.2.2 User interaction

The chatbot should guide the student into asking a proper question. To see what makes a question ’good’, we can take inspiration from StackOverflow, that with over 21 million questions is perhaps the most widely recognised Q&A resource for programmers [73]. They identify the following important parts (’Problem statement details’) of asking a question:

- Summarize the problem:
  - Include details about your goal
  - Describe expected and actual results
  - Include error messages
- Describe what you have tried
- Include relevant code samples

| **Label** | **Requirement** | **Priority** | **Source** |
|---|---|---|---|
| **Chat** | | | |
| 1 | The student can use Chat to communicate with Duckbot | Must | Product idea |
| 2 | Duckbot will guide the student to a detailed question | Must | Brainstorm/Questionnaire |
| 3 | Duckbot will ask the student for specific Problem Statement Details, such as, 'goal statement', 'actual and expected results', 'relevant error messages', 'relevant code', 'steps taken' | Must | Literature, StackOverflow inspiration |
| 4 | DuckBot will feed back gathered Problem Statement Details to the student | Must | Brainstorm/Questionnaire |
| 5 | The student can talk to Duckbot in natural language | Must | Literature: Self-talk |
| 6 | DuckBot will create a summary of Problem Statement Details, visible to TAs. | Must | Brainstorm/Questionnaire |
| 7 | DuckBot will categorize problems | Could | Literature: Research Kok |
| 8 | Duckbot will try to give solutions to problems | Won't | Brainstorm/Questionnaire |
| | | | |
| **Integration** | | | |
| 9 | Duckbot will have integration with TA-Help.me | Should | Product idea |
| 10 | Duckbot will have integration with the Atelier platform | Could | Product idea/Stakeholder: Atelier |
| 11 | The summary of Problem Statement Details will be attached to the relevant code on Atelier | Could | Product idea/Stakeholder: Atelier |
| 12 | TA-Help.me itself does not change in layout | Must | Stakeholder: TA-Help.me |
| | | | |
| **Other features** | | | |
| 13 | The (anonimized) Problem Statement Details summary will be made public, so that other students may benefit from it | Could | Literature: Community of Practice |
| 14 | If a chat helped a student to find a solution, the chat may be made public (anonimized) | Could | Literature: Community of Practice |

**Table 3.1:** Requirements

Evidence has shown that short questions that contain code snippets are the most successful. Additionally, the tone should be neutral and focused on relevant facts (e.g., the problem statement details above) rather than positive or negative utterances (e.g. ’Any help would be really great! :-)’ or ’I have a simple and stupid problem’) [74]. We can incorporate these recommendations in our chat bot design by asking for these *specific* details, while still using natural and friendly conversation with the student. We have the advantage that this longer chat can be summarised to be presented *concisely* back to the student and eventually back to the TA.

It should be noted that, while chatbots can be programmed to have many types of dialogues, including longer conversations that are purely self-explaining and have little guidance, for a prototype that is within the scope of this project, we will start out with a ‘skeleton structure’ of Duckbot. This means that it is primarily focused on eliciting the problem statement details.

## 3.3 Design

### 3.3.1 Use case diagram

Because asking questions is currently done on TA-Help.me, this is a logical place for Duckbot to ’live’. The use case diagram in figure 3.10 shows a comparison between the current use case and the suggested new use case.

We can see that the student now asks their question Duckbot, instead of to TA-Help.me. If they want to refine their question, they can do so with the help of Duckbot. Where before a student directly adds their question to the queue, Duckbot now does this for them at the end of their chat session. TAs can manage the queue as usual (and remove names/questions once dealt with), as well as read the student’s questions.

### 3.3.2 UI design

As mentioned previously in section 3.2, the owner of TA-Help.me did not allow changes to be made to the layout of their website. They are, however, okay with redirections to new pages. This means we cannot build the chat window into the website, but we can make a full-screen chat application that the user is redirected to when they click ’Add me to the list’ on a question list (see the screenshot in figure 3.9/.

Figure 3.11 shows what the chat could look like on desktop and mobile.

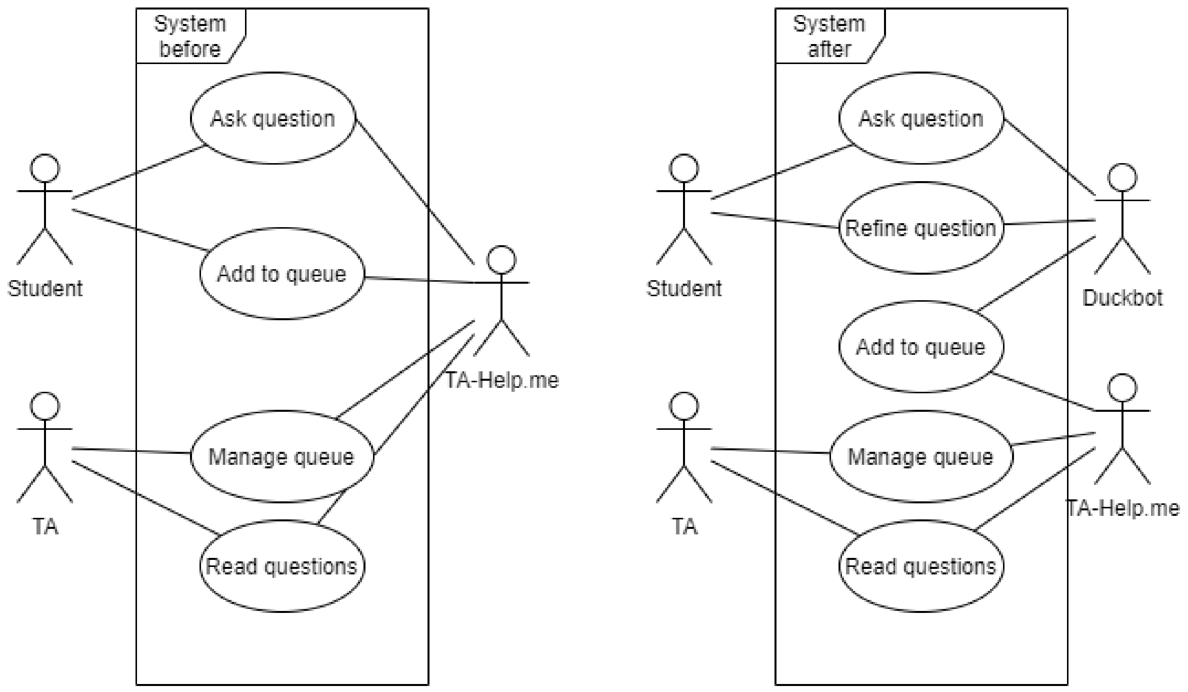
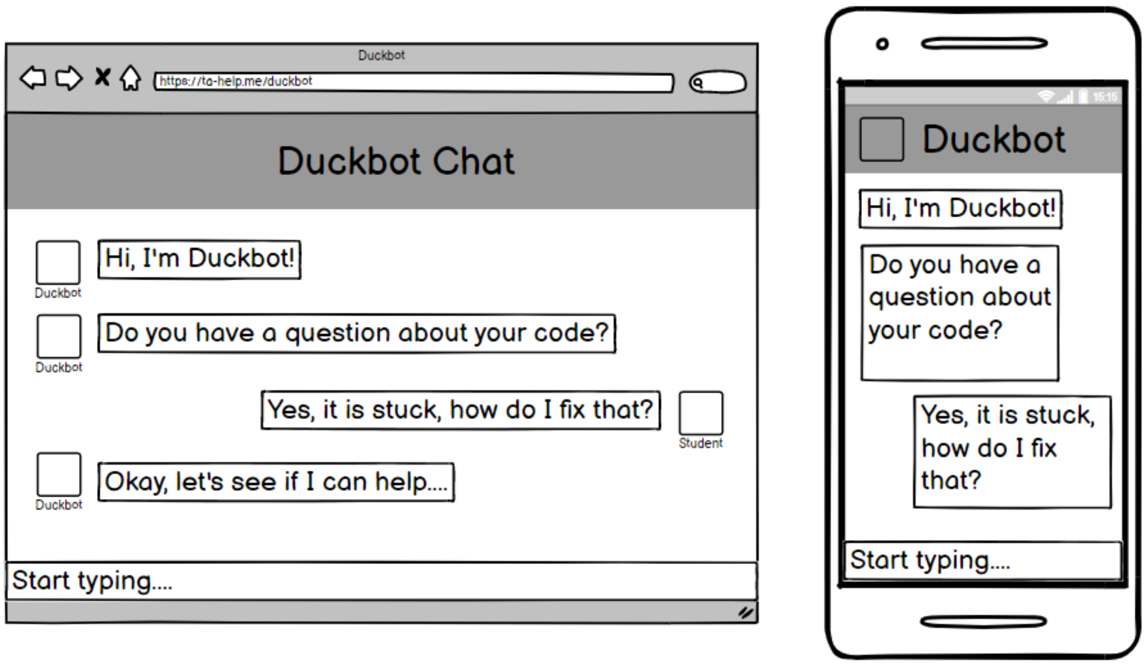


**Figure 3.10:** Use case diagram of Duckbot & TA-Help.me



Hi, I'm Duckbot!

Do you have a question about your code?

Yes, it is stuck, how do I fix that?

Okay, let's see if I can help....

Start typing....

Duckbot

Hi, I'm Duckbot!

Do you have a question about your code?

Yes, it is stuck, how do I fix that?

Start typing....

**Figure 3.11:** Mock-up of Duckbot chat on desktop and mobile

### 3.3.3 Technology stack

**Chat agent**
Because of previous experience with their APIs, products from the Google APIs were preferred for development. They include one for creating chatbots, namely *Dialogflow*. The help of a student experienced with chatbots was enlisted to learn about Dialogflow.

Dialogflow CX is anatural language understanding platform used to design and integrate a conversa-tional user interface into web applications. Dialogflow takes care of controlling theconversation in terms of translating the users' input to a pre-set 'intent' (example: 'I want help on my coding question') and matching it to a preset response (example:'Okay, is it about an error?').

First work was done in the original Dialogflow API (Dialogflow ES), however a new version of the API called Dialogflow CX was released during development. This version allowed building the chatbot much more easily through a graphical interface, that also supported more complex chat scenarios. However, being in a beta phase, this gave some restrictions, e.g. no external data imports are possible.

**Database**
Choosing another Google API solution for our database meant easy integration in the back end. Cloud Firestore, a NoSQL cloud database, was chosen for this reason.

It should be mentioned that using Google products is free for small amounts of users (such as for testing prototypes). However, should it be used in greater numbers or become part of a commercial application, it is no longer a free product. Depending on the possible budget, this might mean these solutions are not future proof.

**Front end framework**
ReactJS is an open-source front-end JavaScript library that is especially suitable for single page applications. Duckbot is designed to be such an application. As React is only used for state management and rendering of the front page, a backend framework is still needed for additional functionality.

**Back end framework**
ExpressJS is a back end web application framework for Node.js. It is lightweight, in that it is fairly minimal and contains no more features than needed. In our case, we use ExpressJS only for communicating with the Dialogflow CX API.

### 3.3.4 Architecture

The sequence diagram in figure 3.12 shows an overview of Duckbot's architecture.

The *user* (a student) enrolls in the question queue in *TA-Help.me* as normal, but TA-Help.me redirects them to the *chat interface of Duckbot*.

When a message is sent to Duckbot, it is processed by *Dialogflow CX*, which tries to match the message to an intent, which in turn gets the appropriate response. Meanwhile, the message is stored in the *Firebase* database.

When all necessary information is obtained from the student, Dialogflow creates an overview with this information, and is shown back formatted to the student in Duckbot. After approval, it gets sent to TA-Help.me, where it can be seen by the *TA*.

## 3.4 Conclusion

Based on TA concerns and previous research in this field, this chapter suggests the use of a conversational agent - a chatbot - to guide students into asking quality questions. A design - both in terms of use case, UI and technology, was presented. For a prototype that is within the scope of this project, we will start out with a 'skeleton structure' of Duckbot, primarily focused on eliciting the problem statement details. The personas and requirements in this chapter will be revisited in chapter 5.

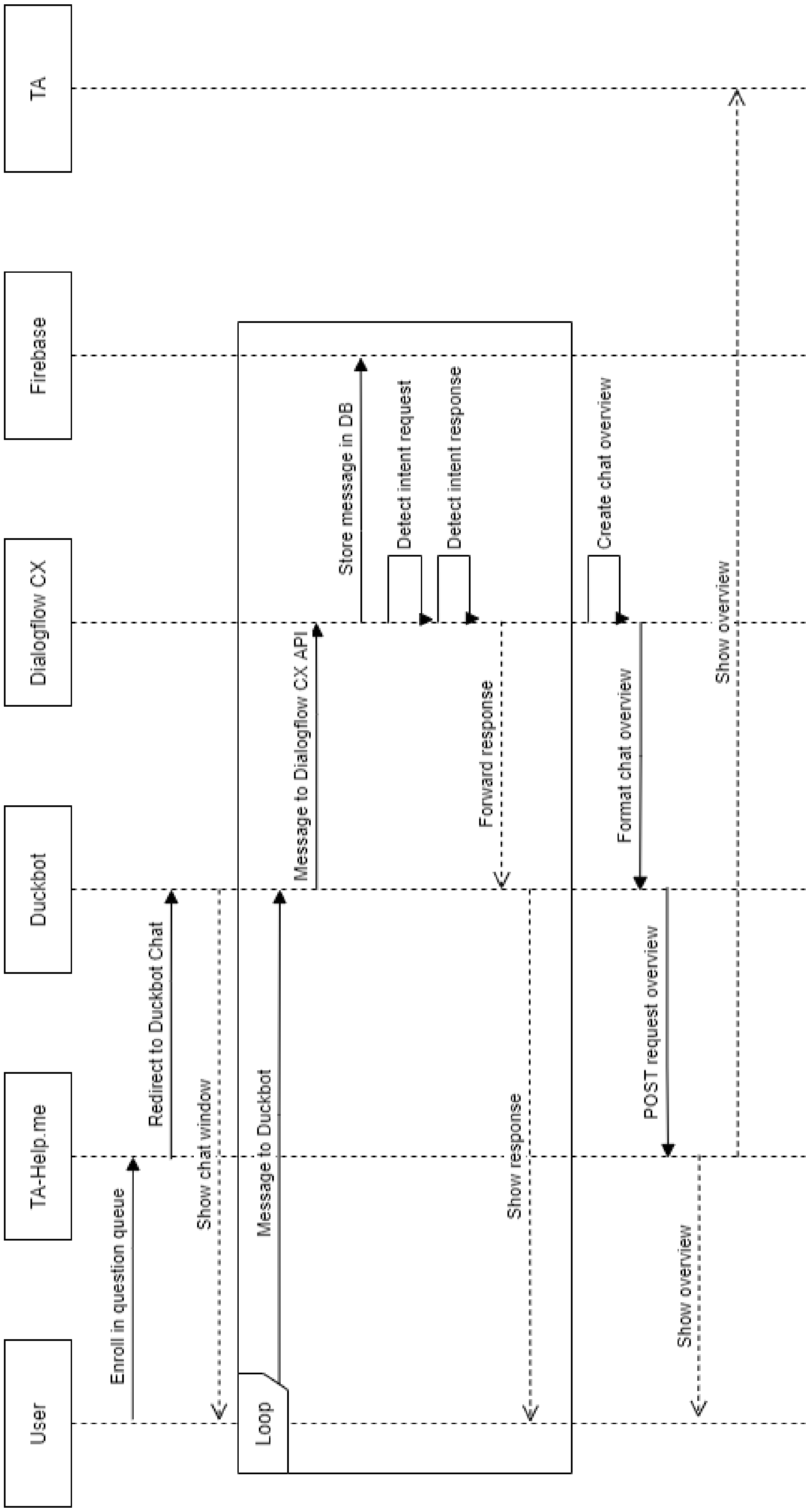


**Figure 3.12:** Sequence diagram of Duckbot

# Chapter 4

# Realization

Based on the specifications in the previous chapter, a prototype was made. This chapter shows the prototype and the user tests to study its usefulness and possible flaws.

In section 4.2 we describe our first test plan, including why this did not work out in reality. Section 4.3 covers the test plan that did succeed and its results. These results will be discussed for interpretation in chapter 5.

## 4.1 Prototype realization

In section 3.2.2, it was discussed that the chatbot guides the student towards asking specific questions. The chatbot can ask for specific problem statement details using natural language.

Appendix B, a screenshot of the Dialogflow interface, gives an idea of how the conversation is 'programmed', where the student is guided through the different problem statement details. The conversation is not fully linear: the student can indicate the type of question they want to ask, whether they are able to provide the details asked for, and they can make changes during the conversation.

Duckbot also gives some additional troubleshooting help, while never providing direct answers. For example, when the student shares an error message they received, we could let Duckbot search for the error message in the documentation and give them the information about it. Instead, Duckbot encourages the students to do this themselves, and tells them:

*“ The error message does its best to tell you what went wrong. At the very least, it will tell you what line number it got to in your program before crashing, which gives*

*you a great clue for places to start hunting for bugs.*

*If you can't seem to figure out what your error message is trying to tell you, your best bet is to copy and paste the last line of the stacktrace into Google. Chances are, you'll get a few stackoverflow.com results, where people have asked similar questions and gotten explanations and answers.*

*Pro-tip: Processing is an extension of Java. Including 'Java' in your search will make sure solutions for the right language turn up!*

*So, have you tried searching for the message? "*

Figure 4.3 shows a screenshot of the chat interface, closely resembling the mockup in figure 3.11. Instead of designing a mobile and desktop version, Duckbot is fully responsive and works on any screen size. For some questions, we also provide some 'shortcut-buttons', for example where an answer of 'yes' or 'no' can be given. This can make it faster for students to work with Duckbot, and was a suggestion by the participant of the pilot-test with Duckbot before going into the 'actual' use tests.

An example where the realisation of the prototype ran into problems, was when a bug in the beta version of Dialogflow CX occurred that made it impossible to save "any other input" than preset intents, if preset intents (such as 'yes' or 'no') were available answers.. In the case of the error messages, for example, the following is the desired flow of the conversation:

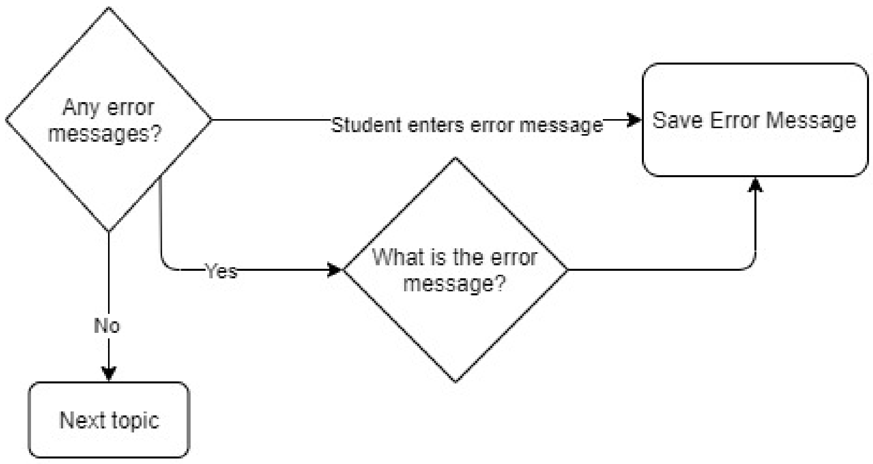


**Figure 4.1:** Desired flow of a 'Yes/No'/"Any other input" question

Because in this case, an immediate answer of an error message to the question could not be saved, the following workaround was necessary:

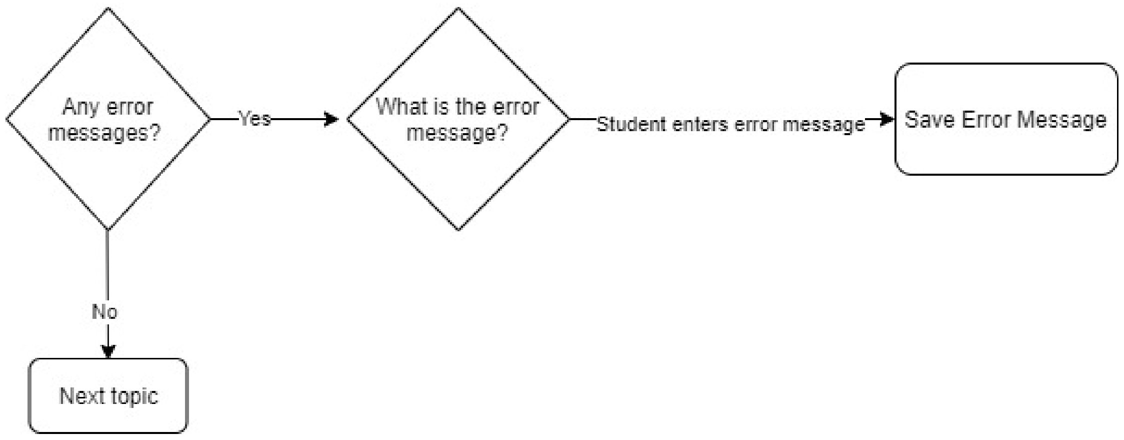


**Figure 4.2:** Workaround to deal with the bug

The shortcut buttons were in this case implemented to make clear that the accepted answers to this question were 'yes' and 'no'. Textual input was not removed for these questions, so that a user may also type these answers out if they prefer not to switch to using the mouse to click the buttons.

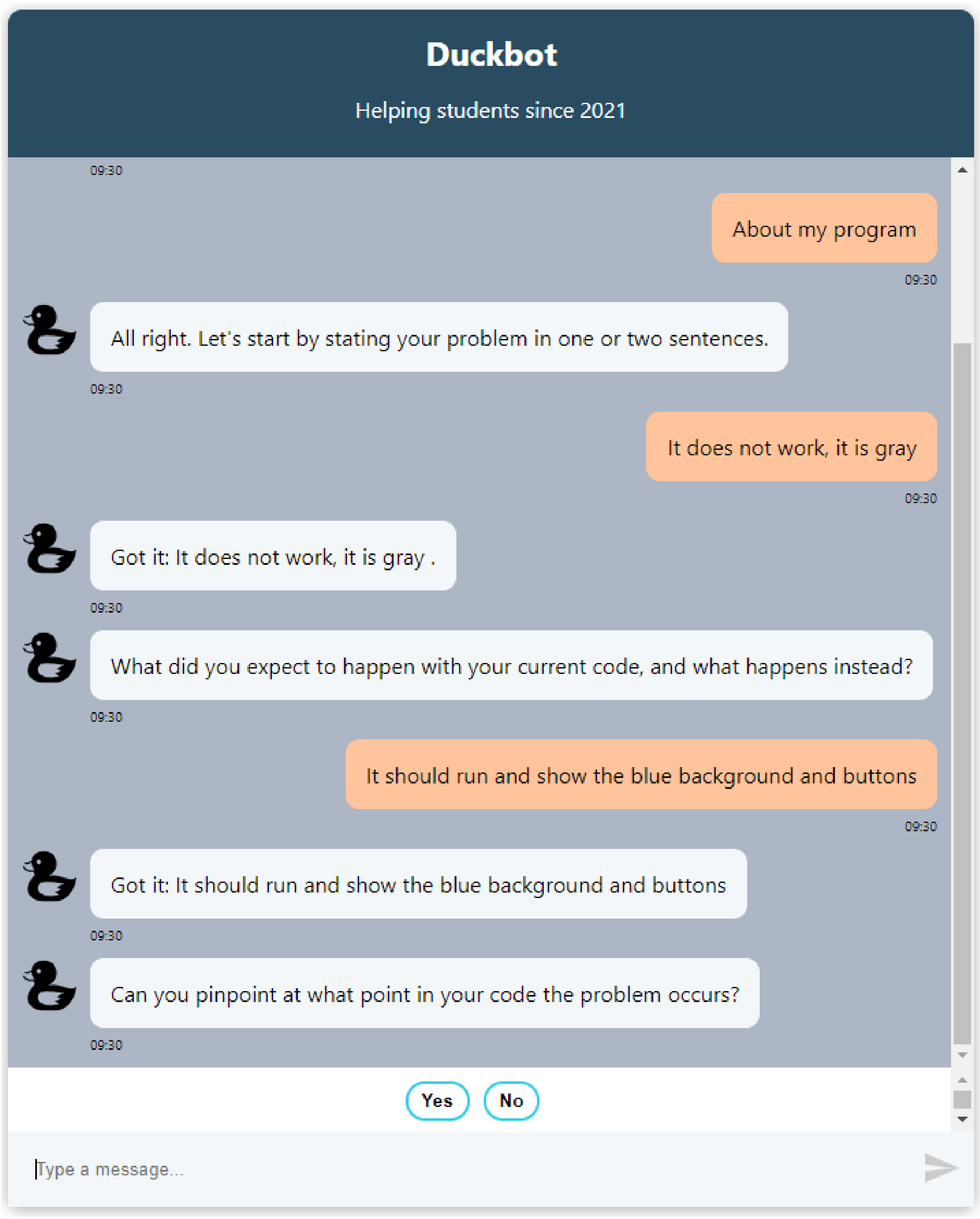
## 4.2 Original test plan

The original test plan was to test Duckbot in a **workshop** that closely resembled an actual tutorial, while the learning material was not part of the actual programme course. This was because there were no ongoing programming courses at that point in time, and students could not have an unfair advantage in future courses over others students by taking part in this workshop.

However, this workshop was organized twice, and both times only one or two students showed up. This was likely because of the ongoing Covid-19 pandemic, during which few students wanted to take extra online classes. In this section, we share the original setup, of which we do not use the results.

**Scope:** We test the entire flow as described in the sequence diagram (figure 3.12). This means that the student is sent from TA-Help.me to Duckbot, and that the TAs can see the results of their chat sessions in TA-Help.me.

**Purpose:** The goal of this test is to see 1) if Duckbot improves the student's understanding of their problems, 2) if Duckbot improves student's questions from a TA point of view, and 3) whether there are any technical or UX flaws in the prototype design.



Duckbot

Helping students since 2021

09:30

About my program

09:30

All right. Let's start by stating your problem in one or two sentences.

09:30

It does not work, it is gray

09:30

Got it: It does not work, it is gray .

09:30

What did you expect to happen with your current code, and what happens instead?

09:30

It should run and show the blue background and buttons

09:30

Got it: It should run and show the blue background and buttons

09:30

Can you pinpoint at what point in your code the problem occurs?

09:30

**Figure 4.3:** A screenshot of the prototype of Duckbot

**Schedule & Location:** Two afternoons (from 13.30 - 16.30) were scheduled, both online via the messaging and videocall platform used in real tutorials (BigBlueButton).

**Equipment:** All participants use their own devices. These are usually laptops. The researcher will be able to store conversations with Duckbot, as well as observe and transcribe interactions between students and TAs when assistance is given by joining them on BigBlueButton.

**Participants:** First and second year students of Creative Technology are recruited through a) learning management system Canvas, b) the official email announcement list for these years and c) promotion in the communication channels of the Creative Technology study association.

**Scenarios:** The lecturer will give an introductory lecture on the topic of the workshop (3D graphics in Processing). Afterwards, they get assignments. While completing them, they can ask questions.

There are 2 scenarios: a student that asks a question is either directed to a normal TA-Help.me queue, where they ask their question in the usual way (control group), without Duckbot, OR they are redirected to Duckbot, which helps them refine their question. The TAs do not know whether the questions they see coming up were asked through Duckbot or through the old system.

The researcher will join the conversations between TAs and students that follow, and take notes.

**Metrics:** From the transcribed conversations, the researcher will label returning conversation details, so that the differences between the Duckbot and non-Duckbot situation can be compared. Additionally, TAs and students will receive questionnaires after the tutorial for *qualitative* evaluation of the workflow with/without Duckbot. This includes the topics ”Successful Task Completion”, ”Likes, Dislikes and Recommendations”, and ”Critical or Non-Critical Errors”.

**Support roles:** The actual lecturer of the programming courses in Creative Technology will provide the lecture. TAs in this workshop are also actual TAs from previous courses.

## 4.3 New test plan

After failing to get enough participants for the workshop, a substitute was needed. In this case we wanted a lower threshold to entry by making it a much shorter test. Additionally, the test had to be individual, so that we would not depend on multiple people being there at the same time. The solution used was to give a specific scenario of programming errors, and ask students to pretend to want to solve these errors in a tutorial.

**Scope:** We test the entire flow as described in the sequence diagram (figure 3.12).

This means that the student is sent from TA-Help.me to Duckbot, and that the TAs can see the results of their chat sessions in TA-Help.me.

**Purpose:** The goal of this test is to see 1) if Duckbot improves the student's understanding of their problems, 2) whether there are any technical or UX flaws in the prototype design.

**Schedule & Location:** Students were able to pick from a list of 45 minute timeslots. Communication was done via Google Hangouts.

**Equipment:** All participants use their own devices. These are usually laptops. All conversations between the students and Duckbot are automatically stored. During the test, the researcher takes notes on any questions that are asked, mistakes that occur or other events of note.

**Participants:** First and second year students of Creative Technology are recruited through promotion in the communication channels of the Creative Technology study association.

**Scenarios:**

Students were given a Processing program assignment from a previous tutorial; simulating raindrops falling on a water surface. To prevent student participants having to program themselves, they were given 3 versions of the solution code. The first one was a working version (see figure 4.4) that showed the student what the the program should look like. The second version had an error in it, and could therefore not run. The third version did run, but showed behaviour that was unexpected, as seen in Figure 4.5.

A test for a control group was designed as well. In this test, students would be shown the same programs with the same errors. Likewise, they would be instructed to request help in TA-Help.me by asking a question. This would allow comparison between questions asked through Duckbot, and those asked without.
Unfortunately, within the time of the research, only 5 students were found willing to participate. To get as much information on the experience with Duckbot, it was opted to have all 5 students do the test with Duckbot.

In a staging environment of TA-help.me, a room with a list that redirects to Duckbot was created. The students were given a link to it (without telling them that enrolling them in the list would redirect to Duckbot, or what it exactly is).

First, they were shown the working version. Then, they were asked to open the

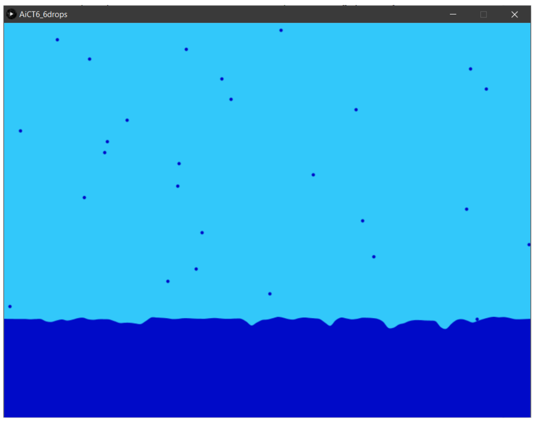


**Figure 4.4:** Normal behaviour

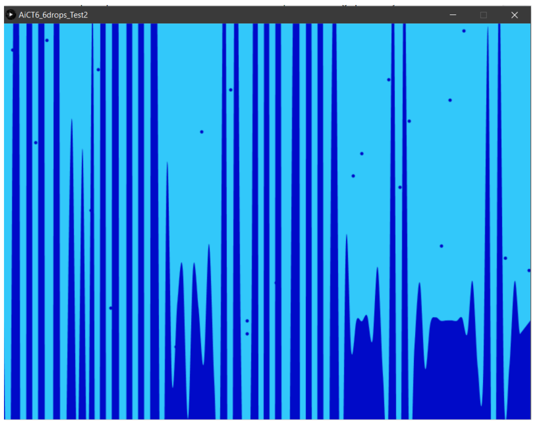


**Figure 4.5:** Unexpected behaviour

second version, run it, and then ask a question through the Duckbot list, as though the behaviour of the program was a problem they encountered during the lecture. This was done twice; for each ’bugged’ version of the program.

They were asked to share their screen with the researcher, so that it could be observed how the participants interacted with Duckbot. Additionally, it made the researcher able to help students who were stuck in a stage of the Duckbot chat (due to flaws in the system, or the inability of Duckbot to detect intent).

One pilot test was also conducted to test whether this setup worked as expected, and whether any flaws were immediately found. One suggestion that came out of the pilot test and was implemented, was to add buttons where specific answers such as ’yes’ or ’no’ were expected.

**Metrics:** From the notes taken, the researcher will label returning conversation details, errors encountered, questions asked and other interesting events.

Finally, with the researcher not present, participants were asked to fill in a short questionnaire. It contained the following statements, to be rated on a 5-point Likert scale from ’*strongly disagree*’ to ’*strongly agree*’:

- *The tool helps me communicate my problem to the TA’s*
- *I am bothered by having to use this tool to communicate with TA’s*
- *The tool used to communicate with TA’s is user friendly*
- *Using this tool in itself would help me understand my problem more*

## 4.4 Results

For this user test, 5 students were willing to participate.

Additionally, 1 student was asked to participate in a pilot test. During this, it was assured that the exercises were easy to understand. In addition, it gave some extra phrases that the pilot participant would say to Duckbot in answer to its questions. These phrases were added to the training phrases, some of which were not understood by Duckbot before.

### 4.4.1 During the test

During the test, there were several things of note that happened multiple times;

The first question Duckbot asks is: “Do you have a question about your code/program or about the course itself?” The chatbot expects an answer that either has the intent ’About my code’ or ’About the course’. However, some students interpreted that this was the time to ask the question already. As Duckbot would not be able to extract an intent from that, it would reply with for example ’Sorry, what was that?’ which causes confusion. In this case, the researcher told the student to specifically type an answer with the intent of ’I have a question about my code’.

Then, the students would move on to the problem statement. Duckbot says: “All right. Let’s start by stating your problem in one or two sentences.” Responses to this question varied from very vague statements (“My program does not behave as it should”) to well-rounded problem statements (“Why do I get an ArrayIndexOutOfBoundsException in the for loop in the update() method in class Water?”).

Duckbot moves on to ask: “What did you expect to happen with your current code, and what happens instead?” In the example of test 1, where the student will encounter an error, we can see that the students anwer the question in one of two general ways:

- Repeat the fact that they receive an error (e.g. “I expected that the for loop was running but I got an error”), thereby not adding additional information to the problem statement or
- Diving deeper in the specific part that causes the problem, and what it should do. Examples are “i expect the force variable to be changed based on that line.” or “It should run through every water particle and calculate the force on it”. In this case, additional information is added and shows signs of an existing or gained understanding of what their problem actually is.

As discussed in section 4.1, due to a bug in Dialogflow, some questions required very specific intents. To answer a question such as ”Can you pinpoint at what point in your code the problem occurs”, an answer of ’yes’ was first needed before actually specifying the location. To indicate this, two buttons appeared above the chat’s typing field with ’yes’ and ’no’. Only one student understood immediately and clicked the button with ’yes’ (or typed ’yes’, leading to the same result). The other students immediately went on to answer the question with the actual line of code where the error occured. This lead to an answer like “Sorry, what was that?” from Duckbot. In this case, the researcher needed to step in and explain that they first had to answer a the question with yes/no.

Similar was the case with the “Do you have any error messages?” section that followed, where again the yes/no-intent needed to be given first.

In the ’error’-section, the test participants needed to use a bit of imagination to deal with questions from Duckbot such as “Have you already tried searching for this message on Google?”. While stated beforehand that in case of non-valid questions because of the test situations, students could use their own imagination to answer the question as they would during a tutorial, some participants were taken aback by this question. In this case, the researcher needed to explain them that they could answer with ’yes’ or ’no’.

The session would always end with an overview by Duckbot, which lists the answers to their questions where applicable, ending with “Can I send this to the TA’s? You can also tell me to edit this overview, or end the session altogether.” Most students did not use the ’edit overview’ option, even if some parts of it were (either due to user error or a Duckbot bug) clearly unintended.

In some cases, Duckbot would not catch an intent that would be clear to a human reader. For example, when asked “Do you have a question about your code/program or about the course itself?” the answers “yes about a error in my code” and “the question is about the code/program” where not interpreted correctly. Another example was the answer “You can send this” to the question “Can I send this to the TAs?”.

### 4.4.2 Questionnaire

The students were asked to fill in a short questionnaire after the test was over, the answers to which were saved anonymously.

The resulting answers to the Likert-scale questions can are shown in table 4.1. To calculate an average, the Likert-scale was interpreted as a continuous scale from 0

| **Statement** | **Strongly dis-agree (0)** | **Disagree (1)** | **Neutral (2)** | **Agree (3)** | **Strongly agree (4)** | **Average** |
|---|---|---|---|---|---|---|
| The tool helps me communicate my problem to the TAs | | | | 5 (100%) | | 3 |
| I am bothered by having touse this tool to communicate with TAs | | 3 (60%) | 2 (40%) | | | 1.4 |
| The tool used to communicate with TAs is user friendly | | 1 (20%) | 1 (20%) | 2 (40%) | 1 (20%) | 2.6 |
| Using this tool in itself would help me understand my problem more | | 1 (20%) | 1 (20%) | 1 (20%) | 2 (40%) | 2.8 |

**Table 4.1:** Results of questionnaire

(strongly disagree) to 4 (strongly agree).

Additionally, the students made the following statements:

S1: *"It seems that the tool didn't really help to solve a specific problem, but it was helpful (from student perspective) because the tool let you think about the problem (what can I do to solve the problem? where in the code is the problem? did I look on the Internet to solve my problem?). This could be really helpful for a student. From a TA perspective, I think the tool is also very helpful to improve and fasten the communication between student and TA. It often happens that a student got stuck and immediately goes to the TA. If the TA asks the student to pinpoint where it goes wrong in the program, the student sometimes have no clue and then it takes a lot of time to help the student."*

S2: *"It is a good moment to reflect on what the actual problem actually is and what should happen. The advice to look it up is not that useful in my opinion."*

S3: *"I have a couple of remarks about the tool. First I think it would be better if the answers are more closed for some questions. For example just a button with problem with code or question about the course. After one use you kind of know* *what the questions will be so i don't know how helpful the tool will be. I can see it working as some sort of checklist to go through first and then sending a problem description to the TA if the student did not fix the issue by going through that list. I like the concept of the tool though. I can imagine a lot of students who do not google an error before asking a TA for help. Hopefully this tool can help TA's help students more efficiently."*

S4: *"This tool makes a student clearly indicate what the problem is and where it occurs. For TAs this would probably be a great help since some students will find out on their own what the problem is or have already thought about it. Although there are still a few bugs in the system I really think this could help improve the way of contacting a TA!"*

S5: *"It teaches students to read and understand error messages more and how to deal with them. In general cases where you do not get an error message I think the tool is a bit unnecessary. Overall I think it will be an improvement compared to the current system."*

# Chapter 5

# Interpretation

In this chapter, the results from the realization phase (chapter 4), i.e. the results of the prototype user test (section 4.4), are interpreted in terms of usefulness, design and technical flaws. The requirements presented in table 3.1 are reviewed, as well as the personas presented in section 3.1.1.

## 5.1 User test interpretation

In this section we evaluate the results of the questionnaire that was presented in section 4.4.2, including the textual statements labelled S1-S5.

### 5.1.1 Usefulness for students

One important aspect of the use of the tool is the communication between students and TA's. All test participants answered the question "The tool helps me communicate my problems to the TA's" with 'Agree'. In all the participant statements, S1 - S5, we find comments on the usefulness.

Perhaps because of the limited scope of Duckbot right now, for some participants there was a large focus on the part of the conversation that concerns Googling. For example in S5: *"In general cases where you do not get an error message I think the tool is a bit unnecessary."* and in S3: *"I can imagine a lot of students who do not google an error before asking a TA for help."* While these thoughts reflect the idea that Duckbot is mainly useful in case of errors that have not been looked up by the user, S2 opposes this with *"The advice to look it up is not that useful in my opinion"*. Duckbot asks for this question, as it is indicated as often asked by TAs (see the TA questionnaire in section 2.2), but few times provided by students on their own initiative. The question is whether, if Duckbot were larger and had more varying branches, this would be such a focal point for the users.

A repeated thought was that the tool helps the student understand the problem, e.g. in S1: *"[...] it was helpful [...] because the tool let's you think about the problem (what can I do to solve the problem? [...])"* and in S2: *"It is a good moment to reflect on what the actual problem actually is and what should happen."* This corresponds with the high score of 2.8 (Agree) on the questionnaire statement "Using this tool in itself would help me understand my problem more". S5 contains: *"It teaches students to read and understand error messages more and how to deal with them. [...] Overall I think it will be an improvement to the current system"*.

These results indicate that the concept of Duckbot is accepted. However, it also shows that the prototype is limited in scope; it is not intended to have a main focus on error messages for example. Duckbot does not tailor the conversation to the individual to a large extent, which is one of the goals of using a conversational agents. All participants had more or less the same conversation. The participants' statements show that, even given the same program and the same chat, the opinions on what is helpful to talk about vary. If anything, this is an indicator that a flexible conversational agent is a good step forward and could offer value over a more static form of guidance.

### 5.1.2 UX Design

The following section discusses the UX related questions in the questionnaire.

The questionnaire ratings on the question "The tool used to communicate with TA's is user friendly" averaged out on 2.6, closest to 'Agree'. Only one student disagreed with this statement.

A clear UX design flaw exists for cases with closed-answer questions, specifically 'yes'/'no' questions. As explained in section 4.1, right now buttons appear to indicate possible options, while still leaving the text input box available to type. The user either does not notice the buttons, or does not realise that those are the expected answers. A relevant remark by a test participant is *"[...] I think it would be better if the answers are more closed for some questions. For example just a button [...] "* - so omitting the option to input text. From a UX point of view, this could be a good strategy, as it eliminates the option to give an answer that Duckbot does not understand. Another option would be to keep the textual input along with the buttons, but have some indicator that would state, for example, "The buttons show the possible answers to this question. You can also type these answers manually."

Overall, the participants seemed to have few problems with the format of Duckbot. Most messages to Duckbot were interpreted correctly. No participant agreed with the statement “I am bothered by having to use this tool to communicate with TAs”, averaging out on 1.4 (Disagree - Neutral).

### 5.1.3 Technical flaws

There are some technical flaws in the tool that should be solved for an official launch.

First, some user messages are not interpreted as the desired intent, such as in the example Q: ”Can I send this to the TAs?”, A: ”You can send this”, where the answer was not interpreted as the ”Yes” intent. While Dialogflow CX can intelligently fills out the phrase list with similar expressions and solves for typo’s, more training phrases are needed. The training phrases list is now biased by the researcher’s own chat style. User tests such as this are a valuable source of training phrases to add.

A small flaw noted by most of the participants was that Duckbot contained messages that should include newlines, but that were simply displayed as \n \n, which caused some confusion in 2 cases. This can and should be fixed in a new version.

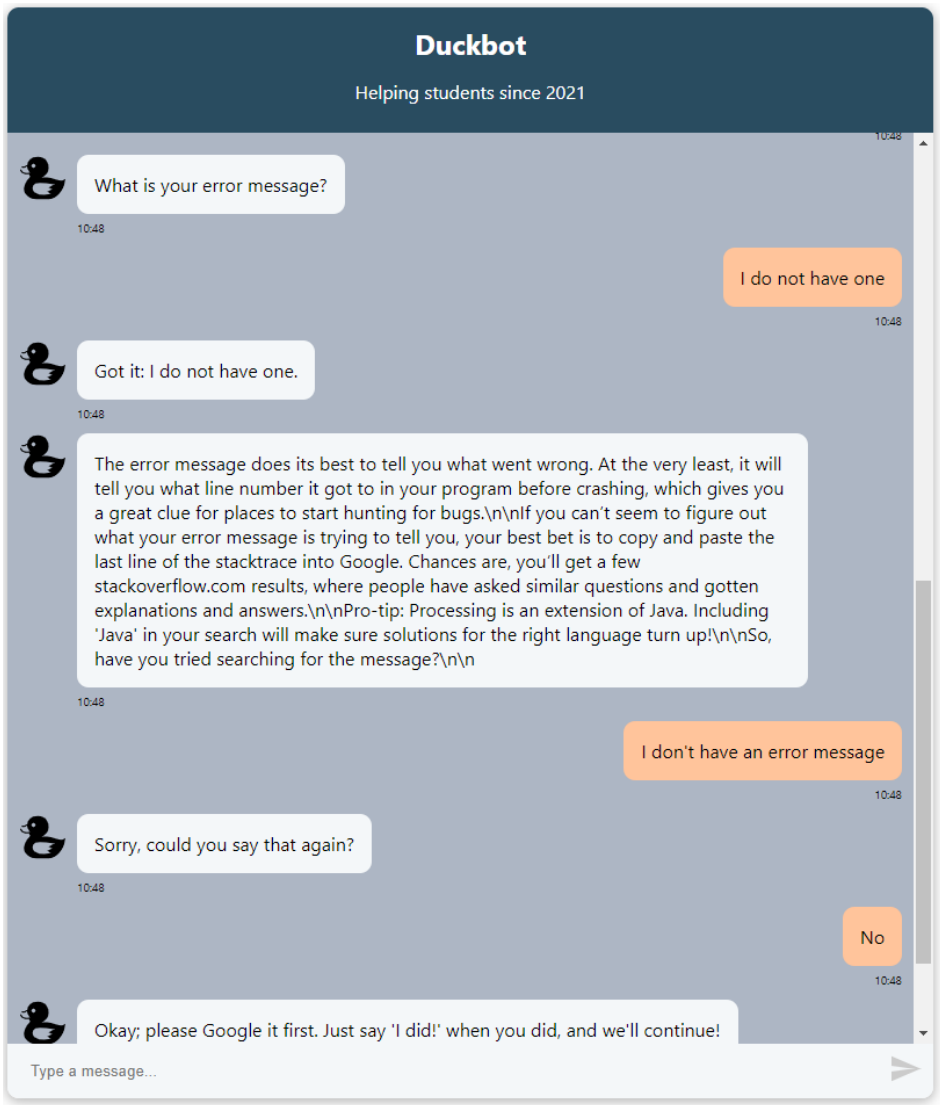
Another flaw was when the Duckbot would ask “Do you have any error messages?” Any input including ’I do not have one’ was stored as an error message itself, leading Duckbot to go into a dialogue branch related to error messages. At that point, there is no way for the user to correct this, and they have to answer Duckbot’s subsequent questions as though they had an error message, to proceed. An example can be seen in figure 5.1. This error is related to the Dialogflow bug that was explained in section 4.1 and figure 4.1. At this point in time, Dialogflow’s technical support has assisted in creating a fix for this problem that can be implemented in a future version.

## 5.2 Requirements fulfilment

Table 3.1 presented a list of requirements, as proposed by the researcher. In table 5.1, this list is shown with *if* the requirement was fulfilled, and if judged as fulfilled by the researcher; *how*.

**’Musts’:** Most of the requirements were fulfilled. Requirement 5, however, needs some improvement, as there were still some issues with matching the correct intent to a user’s chat message (see section 5.1.3).

**’Shoulds’:** The requirement was fulfilled. Integration with TA-Help.me was done by redirecting to the Duckbot application when signing up to a list.


Duckbot

Helping students since 2021

10:48

What is your error message?

10:48

I do not have one

10:48

Got it: I do not have one.

10:48

The error message does its best to tell you what went wrong. At the very least, it will tell you what line number it got to in your program before crashing, which gives you a great clue for places to start hunting for bugs.\n\nIf you can't seem to figure out what your error message is trying to tell you, your best bet is to copy and paste the last line of the stacktrace into Google. Chances are, you'll get a few stackoverflow.com results, where people have asked similar questions and gotten explanations and answers.\n\nPro-tip: Processing is an extension of Java. Including 'Java' in your search will make sure solutions for the right language turn up!\n\nSo, have you tried searching for the message?\n\n

10:48

I don't have an error message

10:48

Sorry, could you say that again?

10:48

No

10:48

Okay; please Google it first. Just say 'I did!' when you did, and we'll continue!

**Figure 5.1:** A screenshot of the faulty error message dialogue and the displayed \n \n

**'Coulds':** These requirements were not fulfilled. These features may be implemented in a future version, but they are not part of the core functionality within the scope of this project.

**'Won't':** This requirement was, according to plan, not fulfilled, as giving solutions to problems is not the aim of Duckbot.

Any requirements not met are analysed in the discussion, section 6.3.

## 5.3 Looking back at the personas

In section 3.1.1, 5 personas were presented to represent target users. In this section, their characteristics and problems are compared to the offered solution of Duckbot, to analyse its effectiveness.

#### Students

"Emma" (Fig 3.1 in chapter 3) is a well-doing student whose pitfall is asking for help. Duckbot might help her because

- The threshold of talking to a chatbot first is lower than going to a TA straight away, thus lowering the chances of not asking for help at all,
- The chatbot takes care of part of the conversation, thereby easing the face-to-face conversation,
- The chatbot helps her communicate relevant information, thereby lowering the fair of 'asking dumb questions'.

"John" (Fig 3.2 in chapter 3) is a student who is more interested in getting the ECs for the course as quick as possible, then the subject matter itself. Duckbot might help him because

- When asking a question, he is forced to think about his problems more thoroughly, thereby making him more able to solve his problems,
- Duckbot streamlines the process of asking questions, making it more efficient.

"Sarah" (Fig 3.3 in chapter 3) is a student who takes the programming course seriously and is looking for knowledge that might be useful in her career. She seems like the student who could do without the aid of Duckbot, but in fact it might be useful to her because

- Duckbot guides students into tackling problems in a manner that professional programmers prefer, and so it helps bridging the gap between the classroom

| Label | Requirement | Priority | Fulfilled |
|---|---|---|---|
| **Chat** | | | |
| 1 | The student can use Chat to communicate with Duckbot | Must | YES (Chat implemented) |
| 2 | Duckbot will guide the student to a detailed question | Must | YES (See 3) |
| 3 | Duckbot will ask the student for specific Problem Statement Details, such as, 'goal statement', 'actual and expected results', 'relevant error messages', 'relevant code', 'steps taken' | Must | YES (All these asked for during chat) |
| 4 | DuckBot will feed back gathered Problem Statement Details to the student | Must | YES (Data recorded by Duckbot shown to student for approval) |
| 5 | The student can talk to Duckbot in natural language | Must | YES (Spoken - improvement needed) |
| 6 | DuckBot will create a summary of Problem Statement Details, visible to TAs. | Must | YES (Summary shown to student first for approval, manual editing by student possible) |
| 7 | DuckBot will categorize problems | Could | NO |
| 8 | Duckbot will try to give solutions to problems | Won't | YES (Won't) |
| **Integration** | | | |
| 9 | Duckbot will have integration with TA-Help.me | Should | YES (Duckbot opens in separate page redirected to when enrolling to question list) |
| 10 | Duckbot will have integration with the Atelier platform | Could | NO |
| 11 | The summary of Problem Statement Details will be attached to the relevant code on Atelier | Could | NO |
| 12 | TA-Help.me itself does not change in layout | Must | YES (Duckbot opens in separate page redirected to when enrolling to question list) |
| **Other features** | | | |
| 13 | The (anonimized) Problem Statement Details summary will be made public, so that other students may benefit from it | Could | NO |
| 14 | If a chat helped a student to find a solution, the chat may be made public (anonimized) | Could | NO |

**Table 5.1:** Requirement fulfilment

and professional programming,

- Sarah gets to show preparation effort in the questions themselves, giving her a chance to stand out even in the TA-Help.me queue, which might be of interest to her.

This shows that, while simple in its design, Duckbot has many different ways of helping students depending on their attitude and personality.

#### TAs

”Joyce” (Fig 3.4 in chapter 3) is a new TA who needs some help overcoming her lack of experience to deal with difficult to answer questions. Advantages Duckbot offer her are

- The ability to read through and prepare for the questions asked before going up to the student,
- Needing less effort figuring out the student’s problem,
- Alleviating some pressure on busy classrooms

”Tim” (Fig 3.5 in chapter 3) is an experienced TA who has little problems with being able to answer questions, but highly values pro-active students throughout the duration of the course. To him, the advantages of Duckbot might be

- Seeing prepared questions instead of dreaded ’I don’t know’s and ’It doesn’t work’s
- Students that are encouraged to deal with their problems in a pro-active manner
- Lowering the threshold to (know how to) ask questions earlier in the course, which may help spreading the workload throughout the course

## 5.4 Conclusion

This chapter discussed that overall, Duckbot was found to be a success by its users. All test participants found that Duckbot helps them communicate their questions to the TAs. Some students indicated that the tool helps the student understand the problem.

The user test also helped pointing out flaws in the tool, including some user messages not interpreted as the desired intent and (ironically) some errors in the dialogue branch related to errors.

Most of the requirements were met, by redirecting students from TA-Help.me to a chat that guided students into giving specific problem statement details. Some requirements - Atelier integration, categorisation of problems and publicising chats - were not met, and will be discussed in section 6.3.

By looking back at the personas, ways Duckbot could be of use to various students were identified. These included lowering the threshold to chat to TAs, helping students deal with their problems pro-actively and guiding students into tackling problems in a manner that professional programmers prefer. To TAs, Duckbot is useful because it gives the TA the ability to better understand and prepare for student's questions, seeing pro-active students and easing the workload during busy tutorials.

# Chapter 6

# Discussion

In this chapter, we discuss the implications of the results that are interpreted in chapter 5. The limitations of the research are discussed, and recommendations for future work are given.

## 6.1 Discussion of results

The key result of this thesis is a solution designed to help overcome common challenges that are faced in programming tutorials concerning interaction between students and TAs. TAs expressed a desire for increasing the quality of the interaction. Helping students take a proactive approach in understanding problems and formulating questions was found to be a key element. As such, the tool Duckbot is a design solution to answer the main research question: *"How can we improve the online education platforms for Creative Technology programming to enhance the quality of help seeking and giving, between students and teaching staff?"*

The main users of Duckbot are therefore students, as their interaction with it improves the interaction with the TAs in turn. During the user test, they all agreed that Duckbot helps them communicate their programming problems to TAs. They were not bothered much by having to use the tool. Overall, the tool was found to be user friendly and helpful in understanding problems, however opinions on this were more varied, with 1 student not finding the tool user friendly or helpful to understanding problems.

This implies Duckbot is on the right track to become a valuable instrument during tutorials. It adds to existing tutoring tools by not being designed to solve problems, rather help students do so themselves and with TA's help. This was an explicit desire expressed by TAs .

Within the digital environment used already for asking questions, namely TA-Help.me, Duckbot directly expands its functionality. TA-Help.me already employs some strategies to help ask useful questions, e.g. giving written tips and example questions. In the research of Heleen Kok, this was found not to help achieve the target behaviour of asking better questions [64] (see section 3.2).

In the context of Persuasive System Design, Duckbot provides 'tunneling', which is "through a predetermined sequence of actions or events, step by step" [75]. Tunneling is found to be effective to reduce the cognitive workload of the user, and helps them to achieve a target behaviour [76]. Duckbot improves Kok's strategy by providing direct assistance, which also 'forces' the student to go through the process, to avoid the current situation where they can simply ignore the tips. As such, Duckbot is a solution to answer Kok's proposal for future work: *"How can a TEL tool intuitively steer students to formulate specific questions?"* and *"How should TAs moderate question asking to increase the quality of the questions the students ask?"* - although it is perhaps a surprising answer to the latter question, in that Duckbot takes over some of the moderation of questions.

## 6.2 Limitations

There were various limitations to this research that need to be addressed.

The most important of these is that the user testing of Duckbot could be improved. The following were limitations in the tests:

- The tool was not used in a real-world scenario (an actual programming tutorial in Creative Technology). Several attempts were made to organise a workshop (section 4.2), but a lack of attendance made it impossible to get enough results. The ongoing Covid-19 pandemic at the time seemed to have played a large part here, with students less willing then usual to spend time on non-obligatory workshops, especially since these had to be given fully online. In addition, recruitment of students was reduced to advertising on digital platforms, instead of the possibility to recruit face-to-face in area's such as study association spaces.
- The alternative test had students work on programming problems that were not their own, in order to save time and lower the threshold to participate. This meant that the test was less successful in testing the hypothesis of whether the students were able to understand *their own* problems better. Some students saw the answer to the problem straight away, and had to pretend to not understand it to go through with the test (this was encouraged, so as to still be

able to test UX related questions).

- Even with the alternative test having a lower threshold to participation, the attendance was low (5 participants). Low attendance was a known problem to various researchers looking for test participants at the time, again quite probably related to Covid-19. However, time ran out to further postpone testing until more participants could be recruited. Ideally, the first tests could have been treated as a pilot test to find and remove more flaws. In the current situation, only one pilot test was conducted, during which not all flaws were discovered, as every individual interacts with the chatbot differently.
- The test was also solely student-focused, with the results not validated by TAs. While we can theorise that Duckbot contributes to alleviating some problems indicated by TAs, this needs to be tested.

There are also some limitations in the design process:

- Due to privacy concerns, it was not possible to access previously asked questions in TA-Help.me. Due to no Creative Technology programming courses being given at the time, it was not possible to collect new questions asked in TA-Help.me. Had this been possible, the results could have given the design of the conversations in Duckbot direction - in that there are direct examples of how students ask their questions, rather than statements from memory by TAs. It can also help give Duckbot validity: examples of questions currently asked on TA-Help.me could be used to do a 'before and after' comparison of questions asked when Duckbot is being used. Both students and TAs can then reflect; which method is preferable, both in terms of usability and effect on the tutorials?
- Studies show that human-computer interface design, an important part of Duckbot, can greatly benefit from an iterative design methodology [77] [78]. With more time, it could be useful to make several versions of Duckbot based on test results.
- More research is needed into the content of a conversation with Duckbot, e.g. tone, questions asked and linearity vs. openness of the conversation flow.

Some parts of the tool itself are also limited:

- There are some technical flaws in the tool, as discussed in section 5.1.3. Fixing these might change the user's opinion on Duckbot. For some, this would require waiting for an update of the Dialogflow CX API, or considering moving to a different platform.

- Due to limited time, the scope and flexibility of the conversation is limited. Duckbot asks for some specific details, but the conversation could be tailored to the user's situation more.
- Duckbot is built in Dialogflow, with storage in Firebase. Both of these are paid Google services, from a certain number of interactions with them. Whether this is affordable to the university remains to be discussed with the relevant people. In case it is not affordable, the database is fairly easy to exchange for a different one. As Dialogflow supports the entire conversation, however, it is much harder to replace. Similar services by different companies include Amazon's Lex and Microsoft's Azure Bot Service. However, company platforms such as these sometimes elicit questions on privacy by educational institutions [79]. If an open source platform is preferred, machine learning libraries such as Tensorflow can be used.

Finally, not all requirements as given in section 3.2.1 were met. The next section will specifically address these, and offer recommendations for future feature implementations.

## 6.3 Recommendations for meeting all requirements

The following requirements (labeled according to table 5.1) were not met:

**CHAT-7:** *Duckbot will categorize problems* - While Duckbot distinguishes between questions concerning programs (e.g., errors, class design) and questions concerning the course itself (e.g. deadlines), these categories are not further refined. This feature exists for asking questions in TA-Help.me the 'normal' way, where students first pick a category (such as 'course questions', 'architecture', 'errors', 'design' and 'concepts). This can help TAs help students more effectively. There are various ways Duckbot can implement categories:

- Ask the student straight away at the start of the conversation, similar to the existing feature in TA-Help.me. This has the advantage of then being able to customise the dialogue. For example, should the student indicate she has a problem with a concept, Duckbot can offer guidance in figuring out what parts of the concepts are problematic, instead of asking for any error messages. A disadvantage is that the student might find it difficult to indicate a category straight away, while Duckbot should be able to help them figure such things out.
- Figure out the category automatically from the conversation. For example, cer-

tain keywords and phrases can be recognized to belong to a certain category. While Dialogflow is perhaps too limited for this, there are plenty of examples in NLP literature of categorisation of texts [80] [81]. It has the advantage of helping out students who may find it difficult to assign their questions to a certain category. A disadvantage if this part of the process is taking out of the student's hands, they may learn less themselves.

- Combinations of these methods are also possible. For example, using NLP to figure out the appropriate category in case the student is not able to determine it at the start. Such a method might make the dialog design much more complex as some possibilities for linearity are lost.

**INTEGRATION-10:** *Duckbot will have integration with Atelier* - The Atelier team expressed an interest in incorporating Duckbot into the Atelier platform. Integration may mean a higher perceived usefulness of the tools and the ability to work more efficiently by having to switch between tools less. A possibility for integration is a redirection to Duckbot when uploading code to Atelier, if the student indicates that they have a question about it. The code could be used in the chat with Duckbot, for example to highlight relevant code snippets. Additionally, when the summary of the conversation with Duckbot is presented to the TA, it could link directly to the relevant code, so that the student and TA do not need to manually communicate this. Finally, it may be useful to include this summary to the Atelier page of this relevant code, for future reference. This would then also fulfil the next unmet requirement: **INTEGRATION-11:** *The summary of Problem Statement Details will be attached to the relevant code on Atelier*.

**OTHER-13**: *The (anonymized) Problem Statement Details summary will be made public, so that other students may benefit from it* - This research has its root in Atelier and in turn, in the 'Community of Practice' (section 1.1.2). This requirement would be particularly useful in a more elaborate platform where students can discuss and share code. The brainstorm session in section 2.1 shows some thoughts on a forum-like platform, where sharing problems might be a good base for discussion. However, it requires more time, research, and resources to build a well-functioning platform, which is outside the scope of this project.

**OTHER-14**: *If a chat helped a student to find a solution, the chat may be made public (anonymized)* - Similar to the previous requirement, a feature like this might be useful in a more elaborate sharing platform. The particular requirement should be treated with care, as literal conversations are more private than summaries.

## 6.4 Recommendations for further research

Duckbot is a solution that contributes a small part to a broader problem. By doing further research, it can have more impact. There are various topics touched upon in this thesis that could be further researched.

Sections 1.1.3 and 1.1.4 presents some research on Persuasive System Design (PSD), which included some recommendations on PSD features to incorporate in this project.

However, there are opportunities to improve the design in this area. For example, Duckbot makes use of the PSD element 'tunnelling', but could improve on 'personalization/tailoring'. Tailoring has helped chatbots be more successful health coaches [82] [83] [84] [85], counselors [86], education [87], personal service [88] and more. The rise of AI-powered 'personal assistants' on smartphones shows that chatbots can be capable of providing many different services to many different people. Researching elicitation strategies for chatbots can help figure out the user needs [89].

The ability of Duckbot to understand the user now depends on a small set of possible answers, and the conversation is quite linear. In an ideal case, the student can talk more freely and casually to Duckbot. Then still being able to extract relevant information would be an interesting topic for future research. Duckbot can only say preset phrases, as is dictated by the chosen framework of Dialogflow. However, using NLG (Natural Language Generation), chatbots can generate output in natural language based on the user's input [90] [91] [91]. This would allow a conversation that is unique every session, which can further assist in self-talk and in general have more possibilities in conversations than a dialogue system that is limited by a preset flow.

The PSD element of 'gamification' was mentioned by TAs during the brainstorm. Gamification has been combined with conversational agents for education before. Interesting virtual agents can enhance the learning experience by experiencing the subject matter in a more interactive, engaging manner [92]. Fadhil et al. [93] designed a system where game elements (quizzes, jokes) are added to a conversational AI. They chatbots have the advantage of striping away interface complexity and reduce the interaction to a simple chat and UI, as well as the possibility to determine the user's emotions to offer a personalised learning experience. Conversational agents can also improve serious games: In a serious game to teach college algebra, virtual characters use text and speech to make learning more fun [94]. They have also, for example, been added to gamified mobile guide applications [95] and VR learning environments [96]. It is worth investigating whether gamification elements can help Duckbot be a better tutor, for example by making learning more engaging.

The elements of the PSD category ’social support’ can give guidance for incorporating Duckbot in a social platform as mentioned in the previous section. Chatbots have been integrated in social commmunities as peers [97], but can also enhance community interactions [98] [99]. Researching this can elevate Duckbot to become than just useful in one-on-one conversations.

There are also topics beyond the scope of PSD. For example, this thesis has the topic of problem giving and seeking at it’s core. However, the question of what exactly makes a programming question effective can use more research. In this research, the ’Problem Statement Details’ asked for by Duckbot were adapted from Q&A website StackOverflow, but do these cover a problem to a large enough extent? Such research would be more social then technical in nature, and may be suitable for research within a different study field.

This research, as discussed in section 6.2 suffered from a lack of user testing to validate results. Future research should not only test improvements to Duckbot using tests with students, but should also include the experience of TAs. Comparing questions asked on TA-Help.me with and without Duckbot can help determine the usefulness of Duckbot. Duckbot should also be tested in real tutorials.

# Chapter 7

# Conclusion

This research set out to improve programming tutorials in Creative Technology. Its roots are in the digital platform Atelier, that aids student-TA (teaching assistant) communication, and the Community of Practice theory. Based on literature research and research into the thoughts of TA, eventually the concept of the chatbot 'Duckbot' was formed.

In this chapter, the conclusion to this thesis is given.

## 7.1 Answering the research questions

The main research question was: **How can a digital tool enhance the quality of help seeking and giving, between students and teaching staff in Creative Technology programming tutorials?**

**RQ1:** *What challenges are faced in tutorials concerning interaction between students and teachers/TAs?*

As part of the Ideation (chapter 2, a brainstorm was held with TAs. Additionally, a questionnaire was sent out to TAs. From these, it is found that challenges faced in tutorials concerning interaction between students and TAs are:

- Students putting entire programs out for TAs to debug, learning nothing in the process themselves
- Students being unsure of where (online or offline) to ask which question and to whom (other students or TAs), leading to questions not being asked at all
- Some important details of questions are often left out

- Pro-activeness of students, and having well-prepared questions, make tutorials more enjoyable for TAs.
- A high workload is a problem that is encountered, but a low workload (either due to low attendance or passive student) is not appreciated either.

**RQ2:** *How can these challenges be addressed in an improvement of the online education platforms?*

When discussing online tools in the brainstorm, TAs gave some answers to help answer this question:

- Course forums are not being populated without making them obligatory
- The tool should be useful within the tutorial, to not encourage students to skip tutorials
- TAs want the tool to nudge students towards being pro-active during tutorials
- TAs want the tool to decrease high workload if necessary
- The tool should encourages TA/student interaction, rather than taking away from it/digitalizing it away
- The tool should increase the quality of these interactions

From the subsequent literature review, the notion that seeking help can be difficult for some students, both because the process feels threatening, and because they do not know how to effectively ask for help. The literature supports that a computer-based solution can assist in both making the process less threatening, and guiding the student in effective help seeking. Learning how to ask the right questions is key. Self-talk is a strategy that can be used as well to support learning.

A tool that, in section 3.2, was theorised to be able to incorporate these concepts was a chatbot, as it can fully guide students step-by-step to ask quality questions, make the process of asking questions less threatening, and lead the student from a computer to a real tutor. With the aid of a chatbot, students can learn how to effectively ask for help. Additionally, it can allow aspects of explaining and self-talk. It can be built into the existing platform TA-Help.me, which is currently used to queue questions for TAs.

The chatbot that was designed for this thesis was named Duckbot (after 'Rubber Ducking'). Duckbot asks the student to refine questions to include important details that will both help the TA understand the problem and work on it more efficiently, but more importantly help the student understand their own problem better (an element of self-talk).

Technically, the solution consists of a ReactJS front-end, a chat agent built on the Google product Dialogflow CX, an ExpressJS backend, and the Google product Firebase as database to save the conversations. Students were redirected to Duckbot when asking a question on TA-Help.me, and a summary of the chat with relevant details of the problem was then made visible to TAs. This is described in sections 3.3.3 and 3.3.4.

**RQ3:** *What is the user acceptability of the design solution prototype?*

A user test of Duckbot with students was described in chapter 4. During this test, the student were given erroneous, pre-made Processing programmes. They were asked to describe these problems to Duckbot, as though they were in a tutorial wanting to request help from TAs. The results of these user tests are given in section 4.4, and analysed in section 5.1.

It can be concluded that overall, Duckbot was found to be a success by its users. All test participants found that Duckbot helps them communicate their questions to the TAs. Some students indicated that the tool helps them understand the problem. However, the limitations of the user test as discussed in section 6.2, such as not testing with enough stakeholders in a real-world scenario, and the limited scope of Duckbot, should be taken into account.

## 7.2 Final words

This project started out with the broad suggestion to improve Atelier and its Community of Practice. Teaching assistants had a primarily role in determining the focus of this research. This focus became the question-asking process on the student side. With Duckbot, the threshold to ask for help becomes lower, and TAs are better prepared to deliver quality help. Whether they are keen question-askers or love to be more self-reliant, Duckbot can offer something to many students, and helps get them on even ground. With students becoming more confident communicators, they can not only benefit their community in the classroom, but the community of programmers as a whole as they take this knowledge into the world.

–

The source code to Duckbot will be made available on Github. Contact the author for more information.

# Bibliography

[1] A. Mader, A. Fehnker, and E. Dertien, “Tinkering in informatics as teaching method,” in *Proceedings of the 12th International Conference on Computer Supported Education*, INSTICC. SciTePress, 2020.

[2] E. Wenger, “Communities of practice: A brief introduction,” 2011.

[3] D. R. Millen, M. A. Fontaine, and M. J. Muller, “Understanding the benefit and costs of communities of practice,” *Communications of the ACM*, vol. 45, no. 4, pp. 69–73, 2002.

[4] S. B. Merriam, B. Courtenay, and L. Baumgartner, “On becoming a witch: Learning in a marginalized community of practice,” *Adult education quarterly*, vol. 53, no. 3, pp. 170–188, 2003.

[5] A. P. Rovai, “A constructivist approach to online college learning,” *The internet and higher Education*, vol. 7, no. 2, pp. 79–93, 2004.

[6] R. M. Palloff, K. Pratt, and D. Stockley, “Building learning communities in cyberspace: effective strategies for the online classroom,” *The Canadian Journal of Higher Education*, vol. 31, no. 3, p. 175, 2001.

[7] R. Nachmias, D. Mioduser, A. Oren, and J. Ram, “Web-supported emergent-collaboration in higher education courses,” *Journal of Educational Technology & Society*, vol. 3, no. 3, pp. 94–104, 2000.

[8] R. K. Valaitis, N. Akhtar-Danesh, F. Brooks, S. Binks, and D. Semogas, “Online communities of practice as a communication resource for community health nurses working with homeless persons,” *Journal of Advanced Nursing*, vol. 67, no. 6, pp. 1273–1284, 2011.

[9] T. M. Schwen and N. Hara, “Community of practice: A metaphor for online design?” *The Information Society*, vol. 19, no. 3, pp. 257–270, 2003.

[10] M. R. Nichani and D. W. L. Hung, “Can a community of practice exist online?” *Educational Technology*, vol. 42, no. 4, pp. 49–54, 2002.

[11] C. M. Hoadley and P. G. Kilner, “Using technology to transform communities of practice into knowledge-building communities,” *ACM SIGGroup Bulletin*, vol. 25, no. 1, pp. 31–40, 2005.

[12] C. Hoadley, “What is a community of practice and how can we support it?” *Theoretical foundations of learning environments*, p. 286, 2012.

[13] C. Haythornthwaite, M. M. Kazmer, J. Robins, and S. Shoemaker, “Community development among distance learners: Temporal and technological dimensions,” *Journal of Computer-Mediated Communication*, vol. 6, no. 1, p. JCMC615, 2000.

[14] H. Oinas-Kukkonen and M. Harjumaa, “Towards deeper understanding of persuasion in software and information systems,” in *First international conference on advances in computer-human interaction*. IEEE, 2008, pp. 200–205.

[15] T. Lehto and H. Oinas-Kukkonen, “Persuasive features in web-based alcohol and smoking interventions: a systematic review of the literature,” *Journal of medical Internet research*, vol. 13, no. 3, p. e46, 2011.

[16] H. Oinas-Kukkonen and M. Harjumaa, “Persuasive systems design: Key issues, process model, and system features,” *Communications of the Association for Information Systems*, vol. 24, no. 1, p. 28, 2009.

[17] “All sites - stack exchange,” https://stackexchange.com/sites?view=list#users, (Accessed on 11/08/2019).

[18] B. Vasilescu, A. Serebrenik, P. Devanbu, and V. Filkov, “How social q&a sites are changing knowledge sharing in open source software communities,” in *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*. ACM, 2014, pp. 342–354.

[19] H. Cavusoglu, Z. Li, and K.-W. Huang, “Can gamification motivate voluntary contributions?: the case of stackoverflow q&a community,” in *Proceedings of the 18th ACM conference companion on computer supported cooperative work & social computing*. ACM, 2015, pp. 171–174.

[20] “reddit: the front page of the internet,” https://www.reddit.com/, (Accessed on 12/11/2019).

[21] C. Moore and L. Chuang, “Redditors revealed: Motivational factors of the reddit community,” in *Proceedings of the 50th Hawaii International Conference on System Sciences*, 2017.

[22] D. Dicheva, C. Dichev, G. Agre, G. Angelova *et al.*, “Gamification in education: A systematic mapping study.” *Educational Technology & Society*, vol. 18, no. 3, pp. 75–88, 2015.

[23] I. Glover, “Play as you learn: gamification as a technique for motivating learners,” in *EdMedia+ Innovate Learning*. Association for the Advancement of Computing in Education (AACE), 2013, pp. 1999–2008.

[24] J. Hamari, “Do badges increase user activity? a field experiment on the effects of gamification,” *Computers in human behavior*, vol. 71, pp. 469–478, 2017.

[25] P. Denny, “The effect of virtual achievements on student engagement,” in *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM, 2013, pp. 763–772.

[26] A. Anderson, D. Huttenlocher, J. Kleinberg, and J. Leskovec, “Engaging with massive online courses,” in *Proceedings of the 23rd international conference on World wide web*. ACM, 2014, pp. 687–698.

[27] A. Iosup and D. Epema, “An experience report on using gamification in technical higher education,” in *Proceedings of the 45th ACM technical symposium on Computer science education*. ACM, 2014, pp. 27–32.

[28] Z. Fitz-Walter, D. Tjondronegoro, and P. Wyeth, “Orientation passport: using gamification to engage university students,” in *Proceedings of the 23rd Australian computer-human interaction conference*. ACM, 2011, pp. 122–125.

[29] F. Groh, “Gamification: State of the art definition and utilization,” *Institute of Media Informatics Ulm University*, vol. 39, p. 31, 2012.

[30] J. Thom, D. Millen, and J. DiMicco, “Removing gamification from an enterprise sns,” in *Proceedings of the acm 2012 conference on computer supported cooperative work*. ACM, 2012, pp. 1067–1070.

[31] G. Zichermann, “Gamification has issues, but they aren’t the ones everyone focuses on - o’reilly radar,” http://radar.oreilly.com/2011/06/gamification-criticism-overjustification-ownership-addiction.html, June 2011, (Accessed on 12/17/2019).

[32] J. Suls, R. Martin, and L. Wheeler, “Social comparison: Why, with whom, and with what effect?” *Current directions in psychological science*, vol. 11, no. 5, pp. 159–163, 2002.

[33] M. M. Engelbertink, S. M. Kelders, K. M. Woudt-Mittendorff, and G. J. Westerhof, “Evaluating the value of persuasive technology and the role of teachers in

a blended learning course for social work students," *Social Work Education*, pp. 1–17, 2020.

[34] J. Hamari, J. Koivisto, and H. Sarsa, "Does gamification work?–a literature review of empirical studies on gamification," in *2014 47th Hawaii international conference on system sciences*. Ieee, 2014, pp. 3025–3034.

[35] R. Wieringa, "Design science methodology: principles and practice," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*, 2010, pp. 493–494.

[36] A. Mader and W. Eggink, "A design process for creative technology," in *DS 78: Proceedings of the 16th International conference on Engineering and Product Design Education (E&PDE14), Design Education and Human Technology Relations, University of Twente, The Netherlands, 04-05.09. 2014*, 2014.

[37] M. Ahmadzadeh, D. Elliman, and C. Higgins, "An analysis of patterns of debugging among novice computer science students," in *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, 2005, pp. 84–88.

[38] J. Sheard and D. Hagan, "Our failing students: a study of a repeat group," in *ACM SIGCSE Bulletin*, vol. 30, no. 3. ACM, 1998, pp. 223–227.

[39] E. Williams, "Student attitudes towards approaches to learning and assessment," *Assessment and evaluation in higher education*, vol. 17, no. 1, pp. 45–58, 1992.

[40] M. M. Müller, "Two controlled experiments concerning the comparison of pair programming to peer review," *Journal of Systems and Software*, vol. 78, no. 2, pp. 166–179, 2005.

[41] C. Hundhausen, A. Agrawal, D. Fairbrother, and M. Trevisan, "Integrating pedagogical code reviews into a cs 1 course: an empirical study," in *ACM SIGCSE Bulletin*, vol. 41, no. 1. ACM, 2009, pp. 291–295.

[42] K. Reily, P. L. Finnerty, and L. Terveen, "Two peers are better than one: aggregating peer reviews for computing assignments is surprisingly accurate," in *Proceedings of the ACM 2009 international conference on Supporting group work*. ACM, 2009, pp. 115–124.

[43] B. Hanks, "Problems encountered by novice pair programmers," *Journal on Educational Resources in Computing (JERIC)*, vol. 7, no. 4, pp. 1–13, 2008.

[44] B. Hartmann, D. MacDougall, J. Brandt, and S. R. Klemmer, “What would other programmers do: suggesting solutions to error messages,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2010, pp. 1019–1028.

[45] L. Razzaq and N. T. Heffernan, “Hints: is it better to give or wait to be asked?” in *International conference on intelligent tutoring systems*. Springer, 2010, pp. 349–358.

[46] S. A. Karabenick and J. R. Knapp, “Relationship of academic help seeking to the use of learning strategies and other instrumental achievement behavior in college students.” *Journal of educational psychology*, vol. 83, no. 2, p. 221, 1991.

[47] B. E. Vaessen, F. J. Prins, and J. Jeuring, “University students’ achievement goals and help-seeking strategies in an intelligent tutoring system,” *Computers & Education*, vol. 72, pp. 196–208, 2014.

[48] R. Ames and S. Lau, “An attributional analysis of student help-seeking in academic settings.” *Journal of Educational Psychology*, vol. 74, no. 3, p. 414, 1982.

[49] T. W. Price, Z. Liu, V. Cateté, and T. Barnes, “Factors influencing students’ help-seeking behavior while programming with human and computer tutors,” in *Proceedings of the 2017 ACM Conference on International Computing Education Research*, 2017, pp. 127–135.

[50] I. Roll, V. Aleven, B. M. McLaren, and K. R. Koedinger, “Designing for metacognition—applying cognitive tutor principles to the tutoring of help seeking,” *Metacognition and Learning*, vol. 2, no. 2, pp. 125–140, 2007.

[51] J. Sillito, G. C. Murphy, and K. De Volder, “Asking and answering questions during a programming change task,” *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 434–451, 2008.

[52] S. Gross and N. Pinkwart, “How do learners behave in help-seeking when given a choice?” in *International Conference on Artificial Intelligence in Education*. Springer, 2015, pp. 600–603.

[53] S. Marwan, A. Dombe, and T. W. Price, “Unproductive help-seeking in programming: what it is and how to address it,” in *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, 2020, pp. 54–60.

[54] M. Nelimarkka and A. Hellas, “Social help-seeking strategies in a programming mooc,” in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, 2018, pp. 116–121.

[55] R. S. Newman, “How self-regulated learners cope with academic difficulty: The role of adaptive help seeking,” *Theory into practice*, vol. 41, no. 2, pp. 132–138, 2002.

[56] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns, “What makes a good code example?: A study of programming q&a in stackoverflow,” in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2012, pp. 25–34.

[57] M. T. Chi, N. De Leeuw, M.-H. Chiu, and C. LaVancher, “Eliciting self-explanations improves understanding,” *Cognitive science*, vol. 18, no. 3, pp. 439–477, 1994.

[58] S. Ainsworth and A. Th Loizou, “The effects of self-explaining when learning with text or diagrams,” *Cognitive science*, vol. 27, no. 4, pp. 669–681, 2003.

[59] D. Thomas and A. Hunt, “The pragmatic programmer,” 2000.

[60] “Cardboard cutout dog,” https://www.sjbaker.org/humor/cardboard_dog.html, (Accessed on 03/11/2020).

[61] “Stack exchange has been taken over by a rubber duck! - meta stack exchange,” https://meta.stackexchange.com/questions/308564/stack-exchange-has-been-taken-over-by-a-rubber-duck/308578#308578, (Accessed on 03/11/2020).

[62] V. A. Aleven and K. R. Koedinger, “An effective metacognitive strategy: Learning by doing and explaining with a computer-based cognitive tutor,” *Cognitive science*, vol. 26, no. 2, pp. 147–179, 2002.

[63] J. K. A. B. R. Robbes, “Asking and answering questions during a programming change task in pharo language,” 2014.

[64] H. Kok, “Improving the efficiency and quality of help seeking and help giving for programming tutorials,” Master’s thesis, University of Twente, 2019.

[65] A. Kerry, R. Ellis, and S. Bull, “Conversational agents in e-learning,” in *International conference on innovative techniques and applications of artificial intelligence*. Springer, 2008, pp. 169–182.

[66] A. C. Graesser, N. Person, D. Harter, T. R. Group *et al.*, “Teaching tactics and dialog in autotutor,” *International Journal of Artificial Intelligence in Education*, vol. 12, no. 3, pp. 257–279, 2001.

[67] C. Conati and K. Vanlehn, “Toward computer-based support of meta-cognitive skills: A computational framework to coach self-explanation,” *International Journal of Artificial Intelligence in Education (IJAIED)*, vol. 11, pp. 389–415, 2000.

[68] V. Aleven, K. R. Koedinger, and O. Popescu, “A tutorial dialog system to support self-explanation: Evaluation and open questions,” in *Proceedings of the 11th International Conference on Artificial Intelligence in Education*. IOS Press Amsterdam, 2003, pp. 39–46.

[69] V. Aleven, A. Ogan, O. Popescu, C. Torrey, and K. Koedinger, “Evaluating the effectiveness of a tutorial dialogue system for self-explanation,” in *International conference on intelligent tutoring systems*. Springer, 2004, pp. 443–454.

[70] R. E. Mayer, “Principles of multimedia learning based on social cues: Personalization, voice, and image principles.” 2005.

[71] N. M. Radziwill and M. C. Benton, “Evaluating quality of chatbots and intelligent conversational agents,” *arXiv preprint arXiv:1704.04579*, 2017.

[72] J. Kuhn, “Decrypting the moscow analysis,” *The workable, practical guide to Do IT Yourself*, vol. 5, 2009.

[73] StackOverflow, “All sites - stack exchange,” https://stackexchange.com/sites?view=list#questions, (Accessed on 07/05/2021).

[74] F. Calefato, F. Lanubile, and N. Novielli, “How to ask for technical help? evidence-based guidelines for writing questions on stack overflow,” *Information and Software Technology*, vol. 94, pp. 186–207, 2018.

[75] B. Fogg, “Persuasive technology: Using computers to change what we think and do. morgan kaufmann publishers,” *San Francisco*, 2003.

[76] C. Calefato, F. Vernero, and R. Montanari, “Wikipedia as an example of positive technology: How to promote knowledge sharing and collaboration with a persuasive tutorial,” in *2009 2nd Conference on Human System Interactions*. IEEE, 2009, pp. 510–516.

[77] G. S. Bailey, “Iterative methodology and designer training in human-computer interface design,” in *Proceedings of the INTERACT’93 and CHI’93 conference on Human factors in computing systems*, 1993, pp. 198–205.

[78] J. Nielsen, “Iterative user-interface design,” *Computer*, vol. 26, no. 11, pp. 32–41, 1993.

[79] “Privacywaakhond uit kritiek op gebruik google-diensten in onderwijs — rtl nieuws,” https://www.rtlnieuws.nl/tech/artikel/5235222/toezichthouder-maant-scholen-en-justitie-om-met-google-te-stoppen, (Accessed on 07/26/2021).

[80] W. Himmel, U. Reincke, and H. W. Michelmann, “Text mining and natural language processing approaches for automatic categorization of lay requests to web-based expert forums,” *Journal of medical Internet research*, vol. 11, no. 3, p. e25, 2009.

[81] D. D. Lewis and M. Ringuette, “A comparison of two learning algorithms for text categorization,” in *Third annual symposium on document analysis and information retrieval*, vol. 33, 1994, pp. 81–93.

[82] A. B. Kocaballi, S. Berkovsky, J. C. Quiroz, L. Laranjo, H. L. Tong, D. Rezazadegan, A. Briatore, and E. Coiera, “The personalization of conversational agents in health care: systematic review,” *Journal of medical Internet research*, vol. 21, no. 11, p. e15360, 2019.

[83] T. Beinema, H. op den Akker, L. van Velsen, and H. Hermens, “Tailoring coaching strategies to users’ motivation in a multi-agent health coaching application,” *Computers in Human Behavior*, vol. 121, p. 106787, 2021.

[84] S. Zhou, Z. Zhang, and T. Bickmore, “Adapting a persuasive conversational agent for the chinese culture,” in *2017 International Conference on Culture and Computing (Culture and Computing)*. IEEE, 2017, pp. 89–96.

[85] M. Alfano, J. Kellett, B. Lenzitti, and M. Helfert, “Proposed use of a conversational agent for patient empowerment.” in *HEALTHINF*, 2021, pp. 817–824.

[86] L. Ring, T. Bickmore, and P. Pedrelli, “Real-time tailoring of depression counseling by conversational agent,” *Iproceedings*, vol. 2, no. 1, p. e27, 2016.

[87] D. Song, E. Y. Oh, and M. Rice, “Interacting with a conversational agent system for educational purposes in online courses,” in *2017 10th international conference on human system interactions (HSI)*. IEEE, 2017, pp. 78–82.

[88] S. Reig, M. Luria, J. Z. Wang, D. Oltman, E. J. Carter, A. Steinfeld, J. Forlizzi, and J. Zimmerman, “Not some random agent: Multi-person interaction with a personalizing service robot,” in *Proceedings of the 2020 ACM/IEEE international conference on human-robot interaction*, 2020, pp. 289–297.

[89] F. Radlinski, K. Balog, B. Byrne, and K. Krishnamoorthi, “Coached conversational preference elicitation: A case study in understanding movie preferences,” 2019.

[90] M. Virkar, V. Honmane, and S. U. Rao, “Humanizing the chatbot with semantics based natural language generation,” in *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*. IEEE, 2019, pp. 891–894.

[91] K.-J. Oh, D. Lee, B. Ko, and H.-J. Choi, “A chatbot for psychiatric counseling in mental healthcare service based on emotional dialogue analysis and sentence generation,” in *2017 18th IEEE International Conference on Mobile Data Management (MDM)*. IEEE, 2017, pp. 371–375.

[92] D. Economou, I. Doumanis, F. Pedersen, P. Kathrani, M. Mentzelopoulos, and V. Bouki, “Evaluation of a dynamic role-playing platform for simulations based on octalysis gamification framework,” in *Workshop Proceedings of the 11th International Conference on Intelligent Environments*. IOS Press, 2015, pp. 388–395.

[93] A. Fadhil and A. Villafiorita, “An adaptive learning with gamification & conversational uis: The rise of cibopolibot,” in *Adjunct publication of the 25th conference on user modeling, adaptation and personalization*, 2017, pp. 408–412.

[94] U. Faghihi, A. Brautigam, K. Jorgenson, D. Martin, A. Brown, E. Measures, and S. Maldonado-Bouchard, “How gamification applies for educational purpose specially with college algebra,” *Procedia Computer Science*, vol. 41, pp. 182–187, 2014.

[95] I. Doumanis and S. Smith, “A framework for research in gamified mobile guide applications using embodied conversational agents (ecas),” *International Journal of Serious Games*, vol. 2, no. 3, pp. 21–40, 2015.

[96] S. Mystakidis, “Distance education gamification in social virtual reality: A case study on student engagement,” in *2020 11th International Conference on Information, Intelligence, Systems and Applications (IISA*. IEEE, 2020, pp. 1–6.

[97] J. Seering, M. Luria, C. Ye, G. Kaufman, and J. Hammer, “It takes a village: Integrating an adaptive chatbot into an online gaming community,” in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–13.

[98] R. G. Athreya, A.-C. Ngonga Ngomo, and R. Usbeck, “Enhancing community interactions with data-driven chatbots–the dbpedia chatbot,” in *Companion Proceedings of the The Web Conference 2018*, 2018, pp. 143–146.

[99] L. Wang, D. Wang, F. Tian, Z. Peng, X. Fan, Z. Zhang, M. Yu, X. Ma, and H. Wang, “Cass: Towards building a social-support chatbot for online health community,” *Proceedings of the ACM on Human-Computer Interaction*, vol. 5, no. CSCW1, pp. 1–31, 2021.

# Appendix A

# TA Questionnaire

On the next page, a printout of the questionnaire that was issued to TAs can be found.

# TAing for programming education

You are being invited to participate in a research study for a master graduation project in Interaction Technology.

The purpose of this research study is to gather information on the behaviour of students and TAs in programming education (typically for Creative Technology), and will take you approximately 10 minutes to complete. The data will be used for the design of a system that might ease the TA-student experience.

Your participation in this study is entirely voluntary and you can withdraw at any time. You are free to omit any question.

We believe there are no known risks associated with this research study; however, as with any online related activity the risk of a breach is always possible. To the best of our ability your answers in this study will remain confidential. We will minimize any risks by going for a fully anonymous approach.

Study contact details for further information: Margot Rutgers, m.t.rutgers@student.utwente.nl

## Time spent as TA

1. How many modules (approximately) have you been a TA for Algorithms or other programming courses?

2. How many years have you studied at a university? (The last time you were TA)

## Getting to the students

3. When helping students, (approximately) what percentage of the time was this in response to their help request?

4. When helping students, (approximately) what percentage of the time was this because you approached them on your own initiative?

5. When helping students, (approximately) what percentage of the time was this because another TA or student redirected you to a student that needed help?

## Getting to the problem

6. What does a help request from a student typically first look like?

*Mark only one oval per row.*

| | Never | Rarely | Sometimes | Frequently | Very frequently |
|---|---|---|---|---|---|
| The student knows what the problem is, but not know how to solve it (e.g. 'These vectors don't add up correctly, what am I doing wrong?') | ⬭ | ⬭ | ⬭ | ⬭ | ⬭ |
| The student encounters an error, but does not know how to solve it (e.g. 'NullPointerException') | ⬭ | ⬭ | ⬭ | ⬭ | ⬭ |
| The student has a program that compiles, but behaves unexpectedly (e.g. 'The balls appear, but do not move correctly') | ⬭ | ⬭ | ⬭ | ⬭ | ⬭ |
| The student knows what they want the program to do, but not how to express this in code (e.g. 'I do not know how to create natural looking wind particles') | ⬭ | ⬭ | ⬭ | ⬭ | ⬭ |
| The student needs to implement a coding principle, but does not know how ('How do I work with classes/PVectors/arrays...') | ⬭ | ⬭ | ⬭ | ⬭ | ⬭ |
| The student has pre-written code they want to add to, but they don't understand it (e.g. Shiffman examples) | ⬭ | ⬭ | ⬭ | ⬭ | ⬭ |

7. Please describe any additional parts or types of help requests you encounter frequently.

_______________________________________________
_______________________________________________
_______________________________________________
_______________________________________________
_______________________________________________

8. What details do students provide to their problem statements _on their own initiative_?

*Mark only one oval per row.*

| | Never | Rarely | Sometimes | Frequently | Very frequently |
|---|---|---|---|---|---|
| Goal statement | ⬭ | ⬭ | ⬭ | ⬭ | ⬭ |
| Actual and unexpected results | ⬭ | ⬭ | ⬭ | ⬭ | ⬭ |
| Error messages (if there are any) | ⬭ | ⬭ | ⬭ | ⬭ | ⬭ |
| Description of what they already tried | ⬭ | ⬭ | ⬭ | ⬭ | ⬭ |
| Description of relevant documentation found | ⬭ | ⬭ | ⬭ | ⬭ | ⬭ |
| Description of what they Googled | ⬭ | ⬭ | ⬭ | ⬭ | ⬭ |
| The specific part of the code that seems problematic | ⬭ | ⬭ | ⬭ | ⬭ | ⬭ |

9. What details do you ask for? Please be honest if you don't ask for these details typically.

*Mark only one oval per row.*

| | Never | Rarely | Sometimes | Frequently | Very frequently |
|---|---|---|---|---|---|
| Goal statement | ◯ | ◯ | ◯ | ◯ | ◯ |
| Actual and unexpected results | ◯ | ◯ | ◯ | ◯ | ◯ |
| Error messages (if there are any) | ◯ | ◯ | ◯ | ◯ | ◯ |
| Description of what they already tried | ◯ | ◯ | ◯ | ◯ | ◯ |
| Have you Googled? | ◯ | ◯ | ◯ | ◯ | ◯ |
| Have you looked at documentation? | ◯ | ◯ | ◯ | ◯ | ◯ |
| The specific part of the code that seems problematic | ◯ | ◯ | ◯ | ◯ | ◯ |

10. Can you describe any other details you ask for to get to a clear problem statemen?

11. Which of the below does a help session further entail?

*Mark only one oval per row.*

| | Never | Rarely | Sometimes | Frequently | Very frequently |
|---|---|---|---|---|---|
| You explain concepts, such as arrays or classes | ◯ | ◯ | ◯ | ◯ | ◯ |
| You explain the meaning of error messages | ◯ | ◯ | ◯ | ◯ | ◯ |
| You explain the cause of a problem | ◯ | ◯ | ◯ | ◯ | ◯ |
| You show an example of code to solve this problem | ◯ | ◯ | ◯ | ◯ | ◯ |
| You show the actual code to solve this problem | ◯ | ◯ | ◯ | ◯ | ◯ |
| You ask the student questions to get to the problem statement | ◯ | ◯ | ◯ | ◯ | ◯ |
| You go through the code yourself, in search of the problem | ◯ | ◯ | ◯ | ◯ | ◯ |
| While talking/explaining, the student figures out the solution by themself | ◯ | ◯ | ◯ | ◯ | ◯ |
| The code is fixed, but the student does not seem to fully understand the problem | ◯ | ◯ | ◯ | ◯ | ◯ |
| You do not know how to help the student out | ◯ | ◯ | ◯ | ◯ | ◯ |

12. Can you describe any additional steps you often take when helping out a student?

## Your experience

13. Can you describe what might make a tutorial session an enjoyable one for you as TA?

14. Can you describe what might make a tutorial session frustrating for you as TA?

15. Is there anything else you want to tell about the topic of helping students?

## Thanks for participation

If you have any questions or remarks, feel free to get in touch with me, Margot Rutgers, via m.t.rutgers@student.utwente.nl or WhatsApp at 0641705048.

# Appendix B

# Dialogflow implementation

Figure B.1 shows the dialog that is present in Duckbot, as seen from the Dialogflow CX Console. Table B.1 gives the actual dialog from Duckbot's side.

| | |
|---|---|
| Start | [Default Dialogflow Greetings] |
| QuestionType | Do you have a question about your code/program or about the course itself? |
| ProblemStatement | All right. Let's start by stating your problem in one or two sentences. |
| ActualExpected | What did you expect to happen with your current code, and what happens instead? |
| YesNoCode | Can you pinpoint at what point in your code the problem occurs? |
| Code | Copy-paste here at what point in your program the error occurs. |
| YesNoErrorMessages | Do you have any error messages? |
| ErrorMessages | What is your error message? |
| ErrorMessageFollowUp | The error message does its best to tell you what went wrong. At the very least, it will tell you what line number it got to in your program before crashing, which gives you a great clue for places to start hunting for bugs.<br><br>If you can't seem to figure out what your error message is trying to tell you, your best bet is to copy and paste the last line of the stacktrace into Google. Chances are, you'll get a few stackoverflow.com results, where people have asked similar questions and gotten explanations and answers.<br><br>Pro-tip: Processing is an extension of Java. Including 'Java' in your search will make sure solutions for the right language turn up!<br><br>So, have you tried searching for the message? |
| PleaseGoogleFirst | Okay; please Google it first. Just say 'I did!' when you did, and we'll continue! |
| IGoogleMyError | Okay - do you still need help? |
| TriedSoFar | What have you tried so far to solve your problem?<br><br>Please tell me more! |
| TroubleshootHelp | Would you like some help troubleshooting your issue? |
| TroubleshootHelp2 | Let's give you some help then! |

| | |
|---|---|
| Overview | if $session.params.code-sample != null<br>Okay, let's review what you did so far.<br>Your problem statement: $session.params.problem-statement.<br>Your expectations and what happened instead: $session.params.actual-expected.<br>What you tried so far: $session.params.tried-so-far.<br>The relevant code is $session.params.code-sample.<br>Can I send this to the TA's? You can also tell me to edit this overview, or end the session altogether.<br>Okay, let's review what you did so far.<br>Your problem statement: $session.params.problem-statement.expectations and what happened instead: $session.params.actual-expected.<br>What you tried so far: $session.params.tried-so-far.<br>Can I send this to the TA's? You can also tell me to edit this overview, or end the session altogether.<br>endif |
| Edit | What would you like to send to the TA's instead? Please make sure to provide all details necessary. |
| CustomOverviewConfirmation | Can I send that to the TA's? |
| CourseQuestion | Please state your question. |
| CouseConfirmation | I'll send to the TA: $session.params.course-question. Is that ok? |
| EndSession | Thank you! You will now be redirected to TA-Help.me |

**Table B.1:** Duckbot's dialog

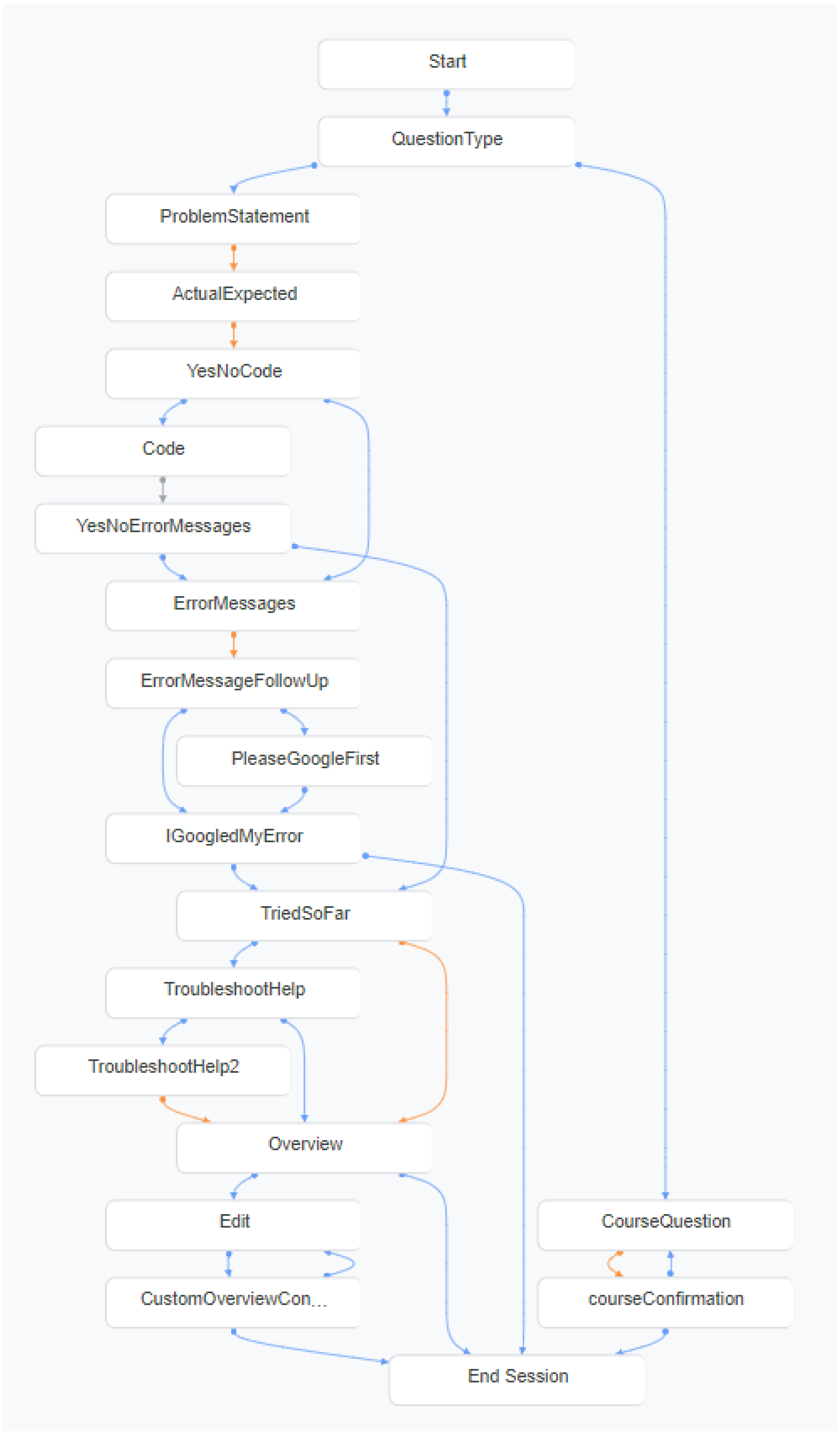


**Figure B.1:** Dialog tree of Duckbot in Dialogflow