Daniel Sandberg

Program Completion Exercise widget for Jupyter Notebook

A practical study

Master's thesis in Computer Science
Supervisor: Guttorm Singre
June 2020

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Daniel Sandberg

Program Completion Exercise widget for Jupyter Notebook

A practical study

Master’s thesis in Computer Science
Supervisor: Guttorm Singre
June 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

For my family
Who always believed in me
And never gave up on me

Summary

Many students partaking in introductory programming courses find learning to program difficult. It can be tedious, hard to understand and boring. Some students are motivated by the effects of knowing how to program, such as getting better jobs, getting higher income, or being required to partake in the courses, rather than being motivated by the learning of programming itself, or simply lack the motivation to put in the necessary effort to learn programming. Others are uncomfortable with writing programs and have difficulty understanding the concepts taught in the courses, or lack the self confidence to perform well when evaluated. Introductory programming courses are also often taught during the first couple of semesters for new students, adding stress to the already difficult challenge of learning programming.

This study involves the creation of the ProCoE widget, a custom Jupyter widget prototype that creates user interfaces for program completion exercises, and the testing and evaluation of this prototype. The study tries to determine if the widget is likely to be used by students in CS1 Python courses, if the widget is likely to be used by teachers in CS1 Python courses, and if the widget is an improvement on the traditional approach to solving program completion exercises. While the study was unable to conclusively determine with any statistical significance if the widget is likely to be used, the results of a user test of the widget and an interview with a former science assistant for the ”Information technology, introduction” course indicate that the widget may be used by both teachers and students. The results also indicate that the widget may be an improvement over the traditional method. The widget is also likely to be an improvement over the traditional method because it can be improved with new features through further development

Table of Contents

List of Tables

List of Figures

Abbreviations

ProCoE	=	Program Completion Exercise
NTNU	=	Norwegian University of Science and Technology
MVC	=	Model-View-Controller
CS1	=	First programming course in a computer science majors
SUS	=	System Usability Scale

Chapter 1

Introduction

Learning how program is known to be a challenge. Many people have a hard time acquiring the programming skill, for a variety of reasons. Many students participating in introductory programming courses have a hard time solving problems given to them by course staff (Gomes and Mendes, 2007), while others have trouble converting their solutions into working computer programs (Winslow, 1996). Many students have trouble understanding of the programming subject, and this has a significant impact on their performance in the programming course they are participating in (Bergin and Reilly, 2005b; Robins et al., 2003). Luxton-Reilly (2016) challenges this view that learning how to program is difficult, suggesting that, instead of the supposed inherent difficulty of learning programming, it is the unrealistic expectations the students meet in CS1 courses that make learning how to program difficult. While learning how program isn't necessarily easy for everyone, covering too much content and expecting too much of the students seem to make the learning process more difficult than it should be. Improving the teaching process and reducing what is expected of the students may make it easier for novice programmers to learn programming concepts and how to program.

Cognitive factors have also been shown to affect a student's ability to learn programming and their performance in CS1 courses. One such factor is a student's motivation for participating in a CS1 course. Jenkins (2001) shows that students participate in CS1 courses because they are mandatory, or because participating in such courses may lead to better salaries or lucrative career. These students are considered to be extrinsically motivated, and have been shown to have weaker performance than students that participate in CS1 courses simply to learn programming, i.e. students who are intrinsically motivated (Bergin and Reilly, 2005a). Comfort level is another cognitive factor that influences a student's performance. Bergin and Reilly (2005a,b) show that students that are more comfortable with programming concepts and writing programs tend to perform better.

Not only is learning programming challenging in itself, but CS1 courses are also often taught during a student's first year of higher education. This time can be difficult for many students, who are faced with many changes. Learning programming in this time of transition can make things even more difficult (Gomes and Mendes, 2007; Jenkins, 2002;

Trautwein and Bosse, 2017). CS1 courses can also be large in size as many different study programs include such courses early on, which is a challenge for both the students and the CS1 course staff (Gibbs and Jenkins, 1992), making it difficult to form relationships between students and course staff. The fact that introductory programming is often taught to large classes of hundreds of students makes this issue even more difficult, because teachers will have limited time to follow up individual students.

When training students in CS1 programming courses, teachers typically hand out programming exercises, where the students are tasked with writing a program based on a description of what the program should accomplish. Because the purpose of CS1 programming courses is to teach the students how to program, this type of exercise is dominant both as exercises (Nelson et al., 2017), and as tasks in final exams (Petersen et al., 2011). However, many weaker students have a low learning outcome from this type of program writing exercise. They often struggle with syntax errors because they don't fully understand the underlying concepts (Nelson et al., 2017), and there have a harder time of getting the code to work as it should. The weaker students can even have trouble writing shorter programs, because even shorter programs often require a good understanding of the multiple programming concepts at once (Zingaro et al., 2012). This indicates that it is important to supplement code writing exercises with other exercise types, such as code tracing (Nelson et al., 2017), and short answer questions that test student's knowledge about individual programming concepts (Zingaro et al., 2012).

To help students' learn programming more easily, program completion exercises have been used. Program completion exercises are exercises where students are given a block of code that are missing some parts, and the students task is to correctly fill inn the missing parts. Program completion is an instructional strategy (or "Reading" strategy) that emphasizes the reading, modification and extension of existing, well-designed and working programs, as opposed to the generation strategies "Expert" and "Spiral", which emphasize top-down program design and incremental learning, respectively (Merriënboer and Kreammer, 1987). Merriënboer and Krammer (1990) found the completion strategy to be superior to the generation strategy due to the necessity of testing and debugging of partial, working programs, and the many opportunities the completion strategy offers to vary the assignments given to the students.

This study is centered around the creation of a learning tool that is intended to assist CS1 courses teaching Python programming. Python is the focus of this tool because it is one of the most popular languages for teaching introductory programming courses at several colleges and universities (Ozgur et al., 2017). "Information technology, introduction" is a CS1 course at the Norwegian University of Science and Technology (NTNU), which is the university this study is performed at. In addition to being taught at educational institutes, Python is used by a wide variety of companies for products in many different industries (Lutz, 2013).

Jupyter Notebook is an open-source web application that allows users to create and share documents that contain live code, equations, visualizations and narrative text with other users[1]. Jupyter Notebook has been used in education (Zastre, 2019; Guerra et al., 2019; Hamrick, 2016; Ellis et al., 2019). "Information technology, introduction", an introductory Python programming course at NTNU, utilizes Jupyter Notebook in their exercise

[1]https://jupyter.org/

system to hand out exercises and provide students with a remote Python programming environment.

Sandberg (2019) proposes a custom Jupyter widget for creating user interfaces for program completion exercises. While the proposed widget was only explored in a theoretical sense, the widget should provide good functionality, and solving program completion exercises using the widget should improve the students' experience over the standard program completion exercises where students are simply handed the incomplete program.

The study consists of two parts; a technical part and an empirical part. The technical part is centered around the creation of a prototype of the program completion widget for Jupyter Notebook. The empirical part will collect and evaluate data from students and course staff regarding the research questions listed below. The widget prototype will be based on the suggestions put forth in Sandberg (2019) with some changes.

This study will try to answer the following research questions:

RQ1: Will the widget be used by students?

- RQ1.1: Is the widget user-friendly?
- RQ1.2: Do students prefer to use the widget over the traditional approach when solving program completion exercises?

RQ2: Will teachers use the widget in their exercise programs?

- RQ2.1: Would the widget be useful in teaching beginner programming to students?
- RQ2.2: Would creating exercises for the widget be practical for teachers?
- RQ2.3: How much effort can teachers expect to spend to make it likely that students will use the widget?
- RQ2.4: What improvements and features would it be desirable for the widget to have?

RQ3: Is the widget an improvement on traditional program completion exercises?

The rest of this report will be structured as follows:

- Chapter 2 presents the background of the study, with explanations of the technologies used to create the widget prototype and a description of the widget proposed by Sandberg (2019)
- Chapter 3 presents the research methods used to test the widget
- Chapter 4 presents the the implemented widget prototype, how it works, and the architecture of the widget
- Chapter 5 presents the results of the widget testing
- Chapter 6 discusses the results of the widget testing, how these results answer the research questions and the weaknesses and limitations of the study
- Chapter 7 presents a conclusion based on the discussion of the testing results

Chapter 2

Background

This section will go through previous work this project builds on, and the technologies the widget utilizes and depends on. The section includes a short description of the widget proposed by Sandberg (2019), a short introduction to Jupyter Notebook, short descriptions of the tools used to build the widget, and a description of the exercise system used in one of the introductory programming courses at NTNU and how the widget would fit into this system.

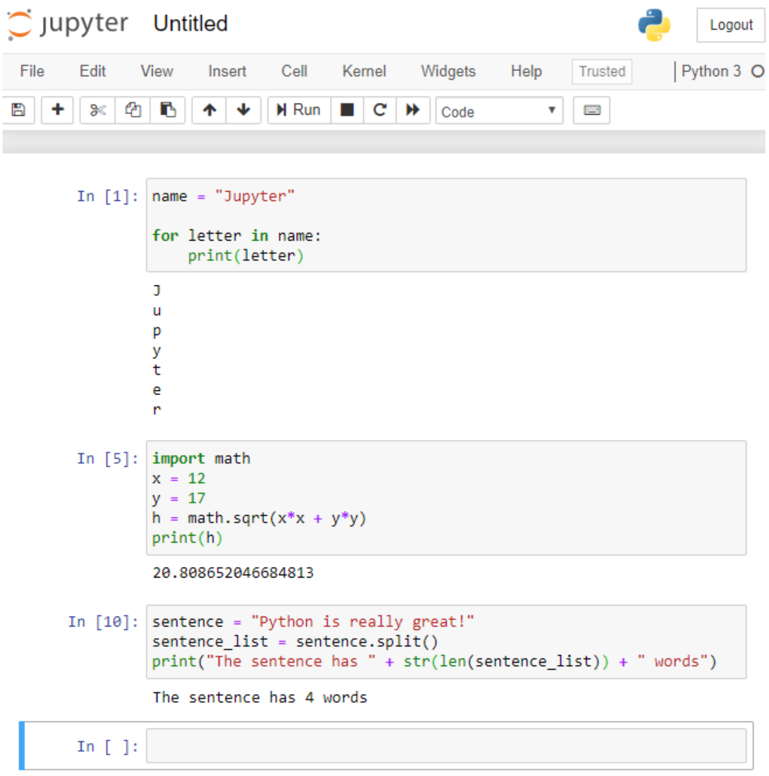
2.1 Jupyter Notebook

Jupyter Notebook is a web-based application used for developing, documenting, and executing code, as well as communicating the results[1]. An instance of a notebook provides text boxes where the user can write code. The code can be written in one of several different languages, depending on which language the notebook was instantiated with. The code in each text box is run individually by selecting the text box and clicking the ”Run” button in the top menu. A few examples of such code blocks are shown in Figure 2.1.

2.1.1 ipywidgets

Jupyter Notebook provides a library called *ipywidgets*. The library contains a set of interactive user-ready widgets that can be used to build user interfaces in a notebook[2]. The ipywidgets library can also be extended, allowing developers to create their own custom widgets to fit a specific purpose.

[1] https://jupyter-notebook.readthedocs.io/en/stable/notebook.html
[2] https://github.com/jupyter-widgets/ipywidgets



```
In [1]: name = "Jupyter"

        for letter in name:
            print(letter)

J
u
p
y
t
e
r

In [5]: import math
        x = 12
        y = 17
        h = math.sqrt(x*x + y*y)
        print(h)

20.808652046684813

In [10]: sentence = "Python is really great!"
         sentence_list = sentence.split()
         print("The sentence has " + str(len(sentence_list)) + " words")

The sentence has 4 words

In [ ]:
```

Figure 2.1: An example of a Jupyter Notebook

2.1.2 widget-cookiecutter

widget-cookicutter[3] is a cookiecutter[4] template that provides widget authors with a default widget project and helps them get started with best practices for packaging and distributing their custom Jupyter widgets. This tool will be used to create the base project for the widget and as a guide for deploying and publishing the widget.

2.1.3 DOMWidget

DOMWidget is a class provided by the Python kernel that is used for building custom widgets that render information in the notebook. The back-end of the widget will be built on top of this class by extending it in the core widget class in the back-end.

[3]https://github.com/jupyter-widgets/widget-cookiecutter
[4]https://github.com/cookiecutter/cookiecutter

2.1.4 Backbone.js

Backbone.js is an MVC framework. It gives structure to web applications by providing models, collections and views, and connects it all to the desired API over a RESTful JSON interface[5]. IPython widgets rely heavy on this framework, connecting widgets defined in the back-end to the Backbone.js models in the front end.

2.2 The Proposed Widget

The purpose of the widget proposed by Sandberg (2019) is to create user interfaces for program completion exercises in Python that people can use to solve those exercises. The widget consists of four core functions that will be explained in the following sections. These four core functions work together to provide the users with a fixed user interface that offers the same functionality as the traditional program completion exercises where the user solves the exercise in a regular text editor or IDE.

2.2.1 Read program from file

The first function of the proposed widget has is to read the exercise program from a file. The exercise is loaded into the widget as a string. This allows the exercise creators to choose the file type they desire when saving the exercise. This opens up for the possibility of using Python files for a more smooth experience when creating exercises.

2.2.2 Create user interface

The second function is to create a user interface for the imported exercise program. This would be accomplished by first parsing the exercise program, looking for the holes in the program. These holes would be marked by the ”**$**” symbol, which has no specific function in Python programs, and would therefor likely be unique to these exercises. Once the widget had parsed the program, it would create a user interface where the holes would be replaced with text input boxes where the user shall complete the exercise program. Any indentation and empty lines in the exercise program would be present in the user interface.

2.2.3 Execute user program

The third function of the widget is to execute the user's completed program. The user would at any time be able to execute their program. When the widget would execute the user's program, the widget would first rebuild the user's program into en executable program by extracting the user's inputs and placing them into the holes in the exercise program. The resulting program would be a string that could be executed using the **exec()** and **eval()** functions provided by the IPython kernel. Once the user's program had been reconstructed, the widget would execute it and store the result.

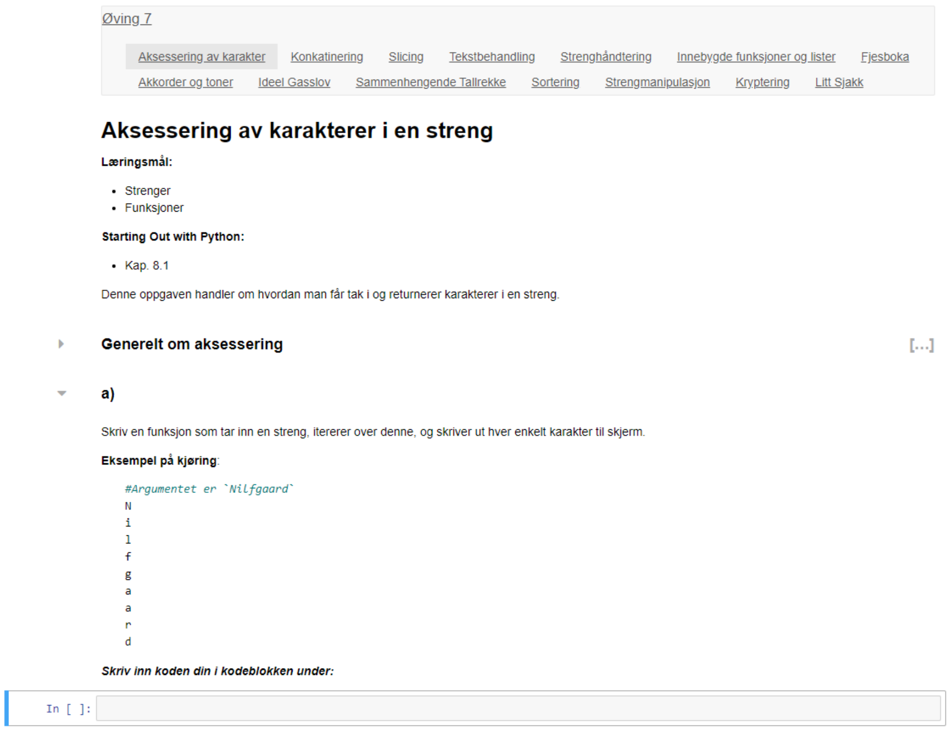
[5]https://backbonejs.org/

2.2.4 Display program output

The fourth function is to display the program result to the user. This function allows the user to see the result of their solution to the exercise. The widget would display any result of the widget, whether the result is an error, a warning or valid output.

2.3 Exercises in ”Information technology, introduction” at NTNU

The course staff for the ”Information technology, introduction” course at NTNU utilizes Jupyter Notebook for the mandatory practical programming exercises. The students are required to solve a certain number of these exercises throughout the semester in order to be eligible for the final exam in the course. These exercises are published on a remote Jupyter Notebook where the students can access and solve them. An example of such an exercise is shown in Figure 2.2.



Aksessering av karakterer i en streng

Læringsmål:

- Strenger
- Funksjoner

Starting Out with Python:

- Kap. 8.1

Denne oppgaven handler om hvordan man får tak i og returnerer karakterer i en streng.

Generelt om aksessering [...]

a)

Skriv en funksjon som tar inn en streng, itererer over denne, og skriver ut hver enkelt karakter til skjerm.

Eksempel på kjøring:

```
#Argumentet er `Nilfgaard`
N
i
l
f
g
a
a
r
d
```

Skriv inn koden din i kodeblokken under:

```
In [ ]:
```

Figure 2.2: Exercise 7 from ”Information technology, introduction” Fall 2019

This is where the proposed widget would come into play. While the mandatory exercises, like in Figure 2.2, are important for evaluating the students’ programming skills, providing students with non-mandatory program completion exercises would give the students great opportunities for practicing their programming skills at a relatively low time cost. Using the widget for these program completion exercises could improve the students' experience when solving the exercises and possibly reduce the time spent solving the exercises.

In addition to improving the experience and efficiency when solving program completion exercises, using the widget would also prepare the students for their final exam in the course. The final exam uses program completion exercises to test a wider range of programming concepts. The design of the user interface for the widget proposed in Sandberg (2019) is inspired by the design of the program completion exercises in previous exams in the course.

Chapter 3

Research Method

This chapter will describe and explain the different research methods used in the design, creation and evaluation of the program completion exercise widget prototype.

3.1 Design-science research

Design science is important when developing and managing information systems and technologies (Winograd, 1996, 1997). As this study involves the design, creation and evaluation of a custom Jupyter widget prototype, it is an example of a design-science research project. Because the study seeks to find out if the widget is likely to be used by students and teachers in CS1 Python programming courses, the widget prototype will be designed and developed to be attractive to students and teachers. Since there already are methods in place for solving program completion exercises, the widget is also intended to be an improvement on the current situation, and therefor will be developed with improved functionality not currently available.

3.1.1 Design methodology

Hevner et al. (2004) presents a framework for understanding, executing and evaluating design-science research in information technology. Figure 3.1 shows a graphical representation of this framework. The framework consists of the environment the designed artifact is in, the design-science research, and the knowledge base. These three parts are connected through the relevance and the rigor cycles. In addition to these two outer cycles, the design-science research also has an inner cycle.

The design process generally starts with identifying the business needs. These needs typically consist of the problems the artifact needs to solve, the context in which the artifact will operate and the processes the artifact will be involved in. These are related to the business part of the environment. The business needs also includes technological requirements and constraints, such as existing infrastructure, information systems and applications. Together, these business needs represent what the artifact needs to accomplish, and how the

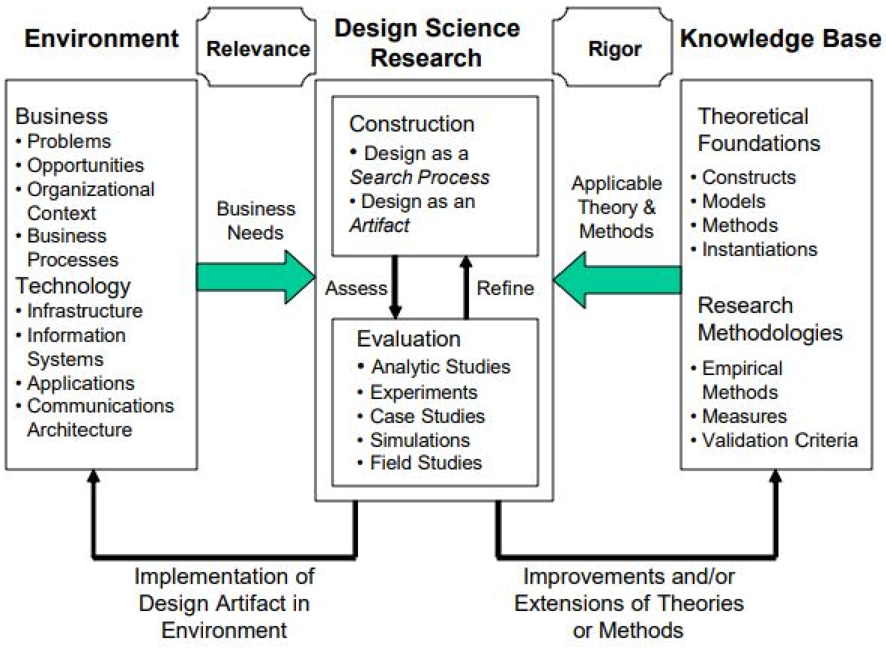


Figure 3.1: Design-Science Research Framework, reproduced from Hevner et al. (2004)

artifact can be created based on technological requirements and constraints. This is the first step first step in the design process as well as the first half of the relevance cycle. If the business needs are extensive and comprehensive, they can be decomposed into more manageable subsets of the original business needs. As the relevance cycle goes through multiple iterations, different business needs are considered as the artifact is developed.

The next step in the design process is to explore previous research to identify existing models, methods and constructs that are relevant to the development of the artifact. Applicable methods and theories provide a starting point the artifact, as well as a yard stick by which the artifact will be measured as it is evaluated. In addition to the theoretical foundation provided by previous research, a set research methods, measures and validation criteria is chosen to evaluate the artifact against previous work. This is the first step in the rigor cycle.

Next is the design and development of the artifact. This is when the internal cycle of the design-science research begins. During this cycle, the artifact is built based on the business needs and previous research. Once the artifact is constructed, it is tested and evaluated to determine if it fulfills the business needs. Based on the results of the evaluation, the artifact is refined. This cycle is typically repeated until the artifact is considered satisfactory.

Once the artifact is constructed, it is evaluated against previous research using the research methods chosen previously, to determine if the artifact improves upon or extends existing methods and theories. The results of this evaluation can in turn be used to further develop the artifact.

In addition to being evaluated against former research, the artifact is implemented into the environment. This is the test that checks the relevance of the designed artifact. From here the artifact can be moved back into development based on the results of the test in the environment, or based on new business needs or a different subset of the original business needs.

This design process, and each of the cycles in the process, can be repeated multiple times during the development of the artifact.

3.1.2 Design-science research guidelines

Hevner et al. (2004) presents 7 guidelines for performing design-science research. This section will go through each of these guidelines and explain how this research project fits in.

Guideline 1: Problem relevance

While program completion exercises can be solved using the traditional method of editing the incomplete program directly, there was a desire among the staff of the "Information technology, introduction" course to have a tool for Jupyter Notebook for creating static user interface for such exercises. Before the creation of the widget prototype, there was no tool for Jupyter Notebook that created such interfaces (Sandberg, 2019). So the problem is that the "Information technology, introduction" course staff want to use static user interfaces for program completion exercises in a Jupyter Notebook, but it's not possible yet. This problem is relevant to students in CS1 Python programming courses who want to get a lot of practice without spending too much effort, and teachers in CS1 Python programming courses who want to provide their students with easy practice when teaching them programming concepts.

Guideline 2: Research rigor

The evaluation of the widget prototype relies on data collected through user testing on students and questionnaires, and through an interview with a former science assistant for the "Information technology, introduction" course. Because of the Covid-19 pandemic and the students preparing for their exams during the test period, the evaluation rigor was impeded. The specific methods selected for evaluating the widget prototype and explanations for why these methods were chosen are described in Section 3.2.

Guideline 3: Design as a search process

The creation of the widget prototype depends on a set of technologies that make the creation of the widget prototype possible. The creation of the widget prototype is also aided by a set of tools that make developing and publishing the widget prototype easier. These technologies and tools are describe in Section 2.1.

The goal of the widget is to improve students experience when solving program completion exercises in Python, compared to using the traditional method of solving these exercises, by providing additional functionality that is not available to the traditional method.

The widget is required to work in Jupyter Notebook, and needs to be able to handle any type of program completion exercise created for it.

Because the widget is required to work in Jupyter Notebook, the widget prototype can only be created using a specific set of technologies, and the functionality the widget can provide is limited to what these technologies allow. The technological constraints are described in Section 4.1.4.

Guideline 4: Design as an artifact

The artifact of this study is the widget prototype. The architecture of the widget, design choices made during development and a description of how the widget works can be found in Chapter 4

Guideline 5: Design evaluation

The widget prototype was evaluated through practical user testing, where students who have previously participated in a CS1 Python programming course tested the widget by solving exercises with it. The user testing process is described and explained in Section 3.2

Guideline 6: Research contribution

The research contribution of this study is the widget prototype and the answers to the research questions posed by the study, based on the evaluation of the widget. The widget is described in Chapter 4, and the evaluation is discussed and concluded in Chapter 6 and Chapter 7.

Guideline 7: Presentation of design research

The research performed in this study is presented in this report.

3.2 Evaluation

One key part of this study is the evaluation of the widget prototype created in the study. To properly evaluate the prototype, several methods were used in the evaluation process. This section will describe the methods used and why these methods where chosen.

3.2.1 User testing

Testing the widget on students is important as they are the end-users who will be interacting the most with the widget. Data collected from this testing will help determine if the widget improves the experience of solving program completion exercises and identify issues and weak points in the widget. The test performed by the participating students consisted of a practical test with the widget and a questionnaire.

Practical test

The practical test consisted of a set of program completion exercises both using and not using the widget. The practical test had to be performed remotely. To make this possible, a repository[1] on GitHub was created that contained the Jupyter Notebook with the tasks used to test the widget and instructions on how to perform the test. Figure 3.2 shows the notebook with the and the three tasks it contained. The number of tasks in the test was limited to three in order to keep the test duration short and make it more likely that students would decide to participate.

Task 1 uses the traditional approach where the test subject is handed the code for the incomplete program where the incomplete parts are marked with '$'. The test subject is asked to complete the program in the exercise so that the program finds all the primes between 1 and 20. This task provides the test subject with a baseline to compare the widget to.

Task 2 uses the widget for the program completion widget. The test subject is asked to complete the program in the exercise so that the function defined in it finds the nth Fibonacci number. This task is the primary task of the test where the student gets hands-on experience with the widget. This is necessary because the widget is the artefact being tested, and will help answer RQ1.1.

The programs in Task 1 and task 2 are both about for-loops. This similarity is intentional, because making the tasks similar ensures that the only major difference is widget. This will provide a basis for discussing RQ1.2, and help answer RQ3.

Task 3 tests the program correctness feature of the widget. The test subject is asked to 5 different programs that give the wrong output. This triggers the widget to show the result of the correct implementation to the test subject. This task helps answer RQ3.

Questionnaire

The questionnaire is the primary method used to collect data and consists of some questions about the students' background, a SUS schema, some questions directly related to the tasks in the practical test, some questions about the likelihood that the students would use the widget in different scenarios, and some questions about issues and improvements.

The questions about the students' background will help establish an understanding of who is being tested without identifying the test subject directly, and they provide a beneficial perspective when evaluating the results and the widget. For each question the test subject is given a set of answers to choose from. The questions and answers are as follows:

Q1: Have you participated or are you currently participating in an introductory Python programming course?

- Yes, I am currently participating in an introductory Python programming course
- Yes, I have previously participated in an introductory Python programming course
- No

[1]https://github.com/DSandberg93/procoe-beta-test

Q2: What level would you say your Python programming skills are?

- Beginner
- Intermediate
- Expert
- Master

Q3: Are you currently studying or have you previously studied information technology or computer science at a college or university?

- Yes, I am currently studying information technology/computer science
- Yes, I have previously studied information technology/computer science
- No, I am currently studying something else
- No, I am not currently studying anything
- I don't want to say

The SUS schema (Jordan et al., 1996) is a commonly used tool to help analyse the usability of a system or software. It is considered to be very reliable (Bangor et al., 2008) and will provide very useful data regarding the usability of the widget prototype. The statements in the SUS schema are related to task 2 and task 3, and the final score resulting from the answers given by the test subjects will help evaluate the usability of the widget prototype and help answer QR1. In the SUS schema the test subjects are asked to answer the following statements on a scale from 1 to 5, where 1 is strongly disagree and 5 is strongly agree:

S1: I think that I would like to use this system frequently

S2: I found the system unnecessarily complex

S3: I thought the system was easy to use

S4: I think that I would need the support of a technical person to be able to use this system

S5: I found the various functions in this system were well integrated

S6: I thought there was too much inconsistency in this system

S7: I would imagine that most people would learn to use this system very quickly

S8: I found the system very cumbersome to use

S9: I felt very confident using the system

S10: I needed to learn a lot of things before I could get going with this system

The test subjects are asked to answer three questions about their experience performing the practical test. These questions are as follows:

Q1: Compare task 1 and task 2. Did you prefer solving the task without the widget (task 1) or with the widget (task 2)?

Q1.1: Why? (Optional)

Q2: Task 3 in the test demonstrated the widget's program correctness feature. Did you find this feature useful?

The answers to Q1 is intended to provide an answer to QR1.3. To get a better understanding of the answers to Q1, Q1.1 was added as a follow-up question. Combining the answers to these two questions will provide a better basis for determining if students prefer using the widget over the traditional method, and thereby answer RQ1.2. The answers to all three questions together will help answer if the widget is an improvement over the traditional method, answering RQ3.

Next on the questionnaire, the test subjects are presented with two scenarios. In the first scenario the test subjects are asked to imagine that they are participating in an introductory Python programming course and the teacher gives them non-mandatory program completion exercises for practice. In order to use the widget, the test subjects would have to download and install both Jupyter Notebook and the widget. They are then asked to answer the following questions on a scale from 1 to 5, where 1 is very likely and 5 is very unlikely:

Q1: How likely is it that you would do these exercises if you could not use the widget, like in task 1?

Q2: How likely is it that you would use the widget when solving these exercises if you had to install Jupyter and the widget on your pc, like in the test?

In the second scenario, the teacher from the first scenario decides to set up a central server with a Jupyter Notebook with the widget installed. No longer faced with the obstacle of having to download and install Jupyter Notebook and the widget, but rather register an account online to get access to the practice exercises and the widget, the test subjects are then asked to answer the following question:

Q3. How likely is it that you would register for remote access in order to use the widget to solve the practice exercises mentioned above?

These three questions illuminate a very important aspect when using the widget, namely how much effort students are likely willing to spend in order to solve program completion exercises, both with and without the widget. Based on this amount of effort, course staff can determine how much effort they need to spend in order to facilitate the use of the widget, and help answer RQ2.3. Determining the cost of making the widget easily available to the students is likely to have a significant impact on whether or not course staff is likely to use and make exercises for the widget. The three questions describe three different levels of effort, both for the course staff and for the students, and studying the difference in the results for these questions will help determine if students and teachers are likely to use the widget.

At the end of the questionnaire, the students are given the opportunity to give direct feedback on their experience using the widget by giving textual answers to the following questions:

- Did you come across any problems with the widget?
- Are there any features that you felt were missing that should have been included in the widget?
- Do you have any other comments?

The widget is currently a prototype, and there is room for improvements. These questions were added to identify if there were any issues with the widget and if the widget had any obvious short-comings that should be fixed before rolling out a full-fledged version of the widget. Any additional comments, such as feature suggestions and such, will also be helpful in deciding how to further develop the widget.

An alternative method of data collection, where the student would be guided through the different tasks in the practical test and then interviewed about their experience with the widget rather than filling in a questionnaire, was considered. This method has the potential to yield more reliable results than a simple questionnaire would. However, this method was rejected for two reasons. The first is that Gløshaugen campus was closed off to students due to the Covid-19 pandemic, making it impossible to perform the test and interview in person, which would have been best. However, video conferencing could have been used to perform the test and the interview. This was also rejected because the period allotted to testing was during the spring exam period, which is the second reason this method was rejected. In order to increase the likelihood that students would choose to participate, it was decided that it should be possible to participate in the testing remotely whenever the participants had time available to perform the test. This also allowed participants to perform the test simultaneously.

3.2.2 Testing with course staff

While students have the most direct interactions with the widget, it is also important to gather information about how CS1 course staff perceive and experience the widget. To collect this information, an interview over video-chat was set up with a former science assistant for the ”Information technology, introduction” course at NTNU. This science assistant has experience using Jupyter Notebook as part of an exercise system. Before the interview started, the former science assistant downloaded the test repository used in the student test, and downloaded and installed the widget.

The interview started with a guided demonstration. The former science assistant explored the user interfaces created by the widget and made a few attempts at solving the exercises. While the student assistant tested the widget, the functionality of the widget was explained. The student assistant was shown the files containing the exercise programs for task 2 and task three, and the hole detection and user interface generation processes was explained to the former science assistant.

After the demonstration, the former science assistant was asked some questions. The interview was performed as a semi-structured interview, being more conversationally, and allowing for more detailed and comprehensive answers and additional comments. The questions the former science assistant was asked where following:

Q1: Over all, what’s your impressions of the widget as it is now?

Q2: Would you have used the widget in the "Information technology, introduction" course as the widget is right now?

Q3: Is it desirable to be able to decide the number of failed attempts required before showing the correct output to the student, rather than to use a hard coded value for this?

Q4: A feature that has been considered for the widget is the possibility of giving the students general hints about how Python works related to the exercise and about the exercise topic. This would be imported into the widget through a variable like "file_path". Is this a desirable feature?

Q5: Right now the widget only supports individual exercises, and is instantiated with a string value of the file path to the exercise. A different way of doing this is to use setup files with all the instantiation values. This would allow the widget to support multiple exercises per widget instance, and the same user interface would be reused for each exercise. The widget would then be instantiated with the string value of the path to this setup file. Would it be applicable for you to create and use such a file?

Q6: One drawback with this single setup file is that it's not possible to show the exercise description as in the tasks in the user test. The description would rather have to be included as comments in the exercise program file, which would be included in the generated user interface. Would this be a problem?

Q7: Do you have any suggestions for improvements on the widget?

Q1 will be useful in answering RQ2.1. The former science assistant has experience working with students at a beginner's level in Python programming, and their opinion of the widget will likely tell if they expect the widget to be useful to the students.

The former science assistants answers to Q2 will give a good indication to whether or not the widget might be used in introductory Python programming courses. While not currently a science assistant in "Information technology, introduction", the former science student may recommend using the widget to current staff in the course, making their answer to Q2 a good indication of whether or not the widget may be used in the future.

The remaining questions are centered around possible improvements and features for the widget, and help answer RQ2.4. Q3-Q5 propose improvements and features that are intended to improve the experience of using the widget both for the students and the teachers, and thereby making it more likely that teachers would use the widget. Q6 points out a potential weakness of the widget, and attempts to determine if this potential weakness would deter teachers from using the widget. Q7 is an open question where the former science assistant can come with any suggestions for how the widget could be improved. The answers to this question will identify what the widget is currently lacking and which improvements should be made to make it more likely that teachers will use the widget in their exercise programs. Together, Q1-Q7 will provide a basis for discussing RQ1.

As the interview with the former science assistant is semi-structured, there is room for discussion other than the questions presented above. Any additional comments the former science assistant gives that are useful to the study will be written down and used to evaluate the widget and discuss the research questions.

3.2.3 Recruitment Methods

Before testing the widget prototype, test subjects had to be recruited. To recruit test subject, three recruitment techniques where used; self-selection sampling, convenience sampling and snowballing sampling. The population studied when testing the prototype, was students at NTNU who participated in the ”Information technology, introduction” course the fall semesters of 2019 and 2018. While this is the target population, the students who participated in the course during the fall semesters of 2017, 2016 and 2015 could be used as a substitute if necessary.

Self-selection sampling

The first recruitment technique use was self-selection sampling. The study and the need for test subjects were advertised in a social media group for 5th-students participating in the Computer Science master's degree programme. Although this group is part of the substitute population for the study, there was unfortunately no way of advertising the study directly to the target population. Recruiting these students was attempted by contacting the professor responsible for a different 1st-year introductory programming course with a significant overlap in participating students with the fall semester introductory programming course. However, the professor was unwilling to forward the advertisement to their students because the study was not directly related their course. Due to NTNU being shut down because of the Covid-19 pandemic during the testing period, the only students that could be recruited without violating the students' privacy where the 5th year students, as previously mentioned. This may have a considerable impact on the results of the prototype testing.

One significant challenge with this method is that the prototype was tested late in the semester when students prepare for their exams and finish up semester projects. This significantly affects the likelihood of people choosing to participate, and it cannot be expected that many students will choose to participate. Because of this, other methods of recruitment had to be put to use to ensure some participation.

Convenience sampling

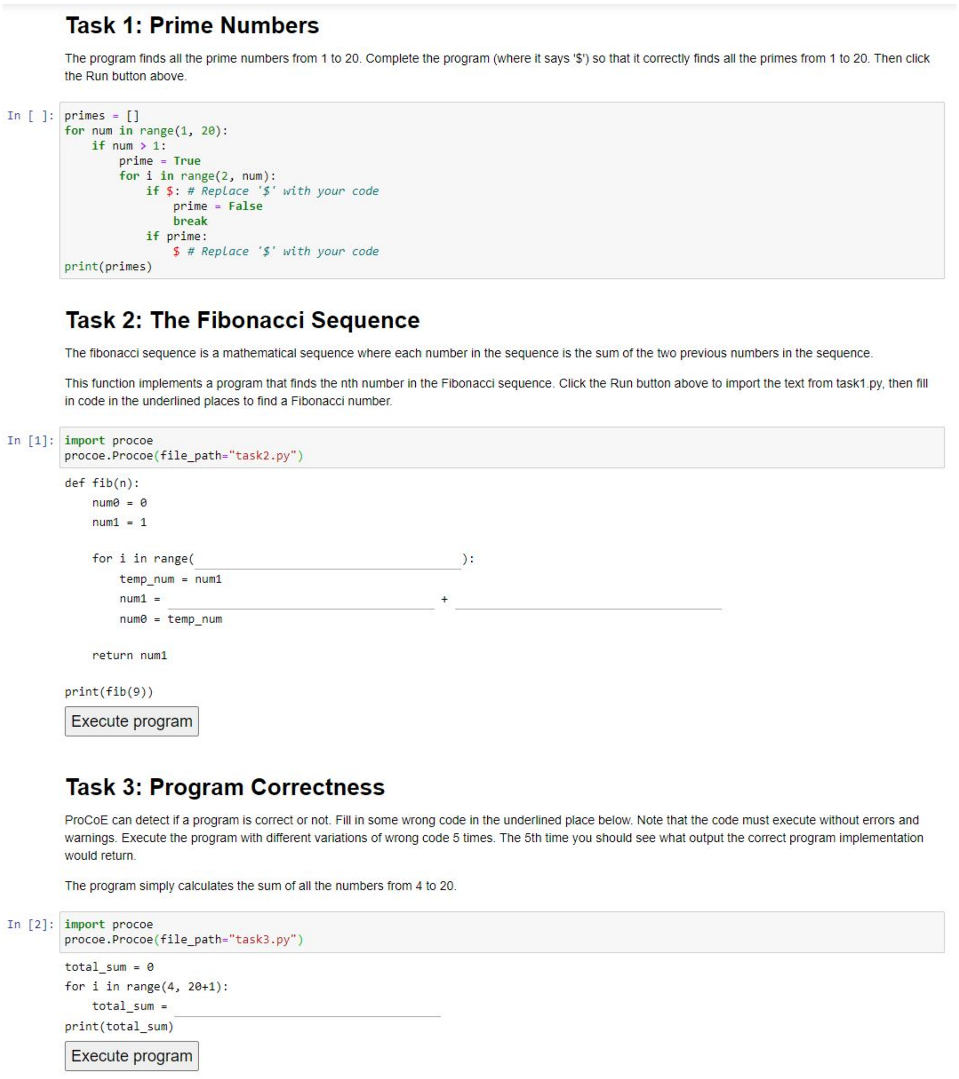
The second recruitment technique used was convenience sampling, where test subject were chosen because they were conveniently available and likely to be willing to participate. These test subjects consists of previous class mates and acquaintances that have previously participated in the ”Information technology, introduction” course. This method was chosen in order to augment the number of participants recruited by the self-selection sampling method. The participants recruited using this method belonged to the substitute population, and the result of the prototype testing may be impacted by this.

Snowballing sampling

The third and final recruitment method used was snowballing sampling, were recruited participants were asked for suggestions of other students who would be likely to participate in the testing. This method built upon convenience sampling, where the students who were directly recruited to participate were also asked for suggestions of other test subjects, and was chosen in order to augment the number of participants recruited by convenience sampling.

Cluster sampling (not used)

One other method of recruitment considered was cluster sampling, which uses the geographical proximity of instances of the target population to recruit participants to the study. In the case of this study, this would involve setting up a testing facility in one of the study halls at Gløshaugen (one of NTNU's campuses) and recruit participants from the students present in that study hall. While the chosen methods are non-probabilistic, this method provides a more representative selection of the target population and would provide results more suitable for statistical analysis. This method of recruitment had to be rejected due to the fact that the campus was closed off to students because of the Covid-19 pandemic during the testing period.



Task 1: Prime Numbers

The program finds all the prime numbers from 1 to 20. Complete the program (where it says '$') so that it correctly finds all the primes from 1 to 20. Then click the Run button above.

```
In [ ]: primes = []
        for num in range(1, 20):
            if num > 1:
                prime = True
                for i in range(2, num):
                    if $: # Replace '$' with your code
                        prime = False
                        break
                    if prime:
                        $ # Replace '$' with your code
        print(primes)
```

Task 2: The Fibonacci Sequence

The fibonacci sequence is a mathematical sequence where each number in the sequence is the sum of the two previous numbers in the sequence.

This function implements a program that finds the nth number in the Fibonacci sequence. Click the Run button above to import the text from task1.py, then fill in code in the underlined places to find a Fibonacci number.

```
In [1]: import procoe
        procoe.Procoe(file_path="task2.py")
```

```
def fib(n):
    num0 = 0
    num1 = 1

    for i in range(________________):
        temp_num = num1
        num1 = ________________ + ________________
        num0 = temp_num

    return num1

print(fib(9))
```

Execute program

Task 3: Program Correctness

ProCoE can detect if a program is correct or not. Fill in some wrong code in the underlined place below. Note that the code must execute without errors and warnings. Execute the program with different variations of wrong code 5 times. The 5th time you should see what output the correct program implementation would return.

The program simply calculates the sum of all the numbers from 4 to 20.

```
In [2]: import procoe
        procoe.Procoe(file_path="task3.py")
```

```
total_sum = 0
for i in range(4, 20+1):
    total_sum = ________________
print(total_sum)
```

Execute program

Figure 3.2: Jupyter Notebook used during the user test

Chapter 4

Program Completion Exercise Widget

Part of this Master's Thesis is the creation of a prototype of the Program Completion Exercise Widget (ProCoE)[1], a custom Juptyer Notebook widget that creates user interfaces for program completion exercises. This chapter describes and explains how the widget works, the design choices made, the technologies used to create the widget, and the architecture of the widget.

4.1 Architecturally Significant Requirements (ASRs)

4.1.1 Functional Requirements

Functional requirements state what the widget must do, and how it must behave and react to run-time stimuli (Bass et al., 2013). The functional requirements for the widget, described in Table 4.1, are based on the requirements presented in Sandberg (2019). Each requirement is given an ID (FR#) and a priority of 1-10, where 1 is the lowest priority and 10 is the highest priority.

4.1.2 Quality Attribute Requirements

Quality attribute requirements, also known as non-functional requirements, are qualifications of functional requirements or of the overall product (Bass et al., 2013). The following sections explain the quality attribute requirements for the architecture of the ProCoE widget.

[1]The repository for the prototype is available at https://github.com/DSandberg93/program-completion-exercise-widget.

ID	Description	Priority (1-10)
FR1	The user should be able to fill in the missing parts of the program	10
FR2	The user should be able to submit the completed program	8
FR3	The widget should take as input a Python program as a string	8
FR4	The widget should read the incomplete program and the solution program from a file	6
FR5	The widget should detect the incomplete parts of the program	9
FR6	The widget should show the incomplete program to the user with text inputs, replacing the incomplete parts of the program	9
FR7	The widget should execute the completed program	8
FR8	The widget should display the result of the execution of the completed program to the user	7
FR9	The widget should detect the correctness of the user's program and display it to the user	7

Table 4.1: Functional requirements

Modifiability

The ProCoE widget is a prototype and will likely be changed during development as more features and requirements are identified. Therefore, it is important to make sure that both the architecture and the widget itself are flexible, and that adding, changing and removing is easy, and inexpensive with regards to time and effort.

Availability

The core feature of the ProCoE widget is that it can generate user interfaces for many different exercises. It is imperative that the widget reliably generates correct user interfaces without errors in the user interfaces.

4.1.3 Business Goals

Business goals are the main reason for building a system (Bass et al., 2013). Many such business goals impact the architecture directly, resulting in specific quality attribute requirement, or indirectly.

Research Project

The ProCoE widget is developed as part of a research project, and some of the other parts of the project, such as testing the widget, depends on the widget being implemented before moving on to the next part. Also, in order to ensure that the other parts can be performed properly, the widget prototype needs to work well. The widget prototype is also itself a part of the research project, and changes may be made to the widget as the testing of the widget may identify the need or opportunity for improvements, requiring the widget and the architecture to be modifiable.

Schedule

The master's thesis the ProCoE widget is part of is due for delivery on June 10th. Because the widget is a part of a research project, the time available to spend on developing the widget and the related architecture is limited, and time has to be allocated for the other parts of the research project.

4.1.4 Constraints

Constraints are design decisions with no degree of freedom, meaning they are design decisions that are already made (Bass et al., 2013).

Jupyter Notebook The ProCoE widget was created specifically for use in Jupyter Notebooks, making Jupyter Notebook a requirement for using and developing this widget.

JavaScript The graphical part of custom widgets for Jupyter Notebook are created using JavaScript. The user interface of the ProCoE widget one of the main features of the widget, making the use of JavaScript a requirement.

Python Jupyter Notebook for python utilizes an IPython kernel which is run on the server running the Jupyter Notebook. The ability to execute strings containing Python code is instrumental to the ProCoE widget, making the use of Python for the back-end of the widget a requirement.

4.2 Design Choices

In the design and creation of the ProCoE widget prototype, several design choices were made as to how the prototype should look, how it should work, and which features it should have. This section will explain these choices and why they were made.

4.2.1 User Interface

The primary function of the ProCoE widget is to provide a graphical user interface for program completion exercises. Therefor, design choices regarding the user interface have major impact on how well the widget works. It is essential that the purpose of each part of the user interface is communicated well to the user. It is particularly important for user interfaces with interactive elements to have predictable behaviour when interacted with, as to not cause confusion. For the user interface of the widget prototype, there are three elements of particular importance; the text input fields, the program execution button, and the feedback area.

Text input fields

The text input fields are integral to the design of the user interface. These fields have a heavy impact on the functionality of the user interface and need to communicate that they are both interactive and that they are part of the text they are placed in. Text input fields in web applications are by default bordered boxes. The input fields used in the prototype were changed to only have a border at the bottom. Changing the border of the input fields like this may reduce the sense of interactivity, but the input fields' presence alone indicate the need for interaction. The border change also improves the sense of unity with the rest of the program.

The border that shows up when an input is active was also removed to improve the feel of integration, and the font of input fields was changed to be the same as the rest of the program. While these two changes have a smaller impact on the quality of the user interface, the cost of making these changes were virtually zero, and they improve the sense of integration of the input fields.

Another important part of the text input styling is that they were given a specific minimum width. The same minimum length is given to all the input fields. The reason for giving them this minimum width was to make sure that the user is given no inadvertent hints as to how much code to write into the input fields. In addition to the minimum length, the width of the input fields also changes to fit the length of the text written into the field should it exceed the minimum length. The adaptive length was restricted to 70 characters for the prototype to keep the fields from exceeding the width of the code boxes in Jupyter Notebook. This limit also restricts the number of letters the user can write in each field to 70 as well. It is unlikely that program completion exercises at the beginner's level would contain holes that requires code that exceeds this limit, however, this limit is likely to change as the widget is developed further.

Program execution button

The program execution button is an element of the user interface that is distinctly separate from the rest of the interface in terms of purpose and function. To properly communicate this difference, it is important to make sure it sticks from the rest of the interface, and that it is clear to the user that it is a clickable. To maintain the inherent interactive nature of the default button provided the ipywidgets package, this default button was used for the program execution button with very few changes. The font size was increased to make the purpose of the button clearer and the internal padding between the button text and the edges of the button was increased. While the button by nature sticks out from the rest of the interface, these small changes increase this effect.

Feedback area

The feedback area is the area where messages are shown to the user depending on their interaction with the widget, and is the only area where the widget can give information directly to the user. It is therefor important to make sure that the messages shown are formatted and styled in an unambiguous manner.

Position

Position has a significant impact on what parts of the user interface belongs together. It is intuitive to place the feedback area at the bottom of the user interface, and this placement helps separate the feedback from the exercise program. Keeping the program execution button between the two areas also helps with this separation. In addition, many IDEs and code editors place feedback areas, such as the terminal, at the bottom by default, so placing the widget feedback area at the bottom builds on the user's experience from using such IDEs and code editors, if they have such experience.

Text color

Text color, when used correctly, can have a significant positive impact on the user's perception of the text itself. Four colors were used when color coding the messages in the feedback area; green, yellow, red and black. Green, which is typically associated with positive messages and success, is used for the response message when the user solves the exercise correctly, as shown in Figure 4.7. Red, which is often associated with failure and danger, is used for the response message when the user solves the exercise incorrectly, as shown in Figure 4.8. Yellow, which is often associated with warnings, is used when the response message is a warning or error, as shown in fig. 4.9. Black, which is generally seen as a neutral color, is used for all other messages.

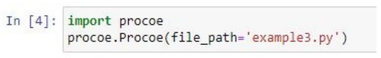
The choice of using red and yellow as described above could have been reversed, because programs that cause errors or warning are in general more critical than programs that are incorrectly implemented but runs without errors or warnings, and red messages seem more critical than yellow messages. However, in the context of program completion exercises, incorrect implementations are more critical because the purpose is to learn how to write correct programs, making an incorrect implementation a wrong answer and red the natural color to use for incorrectly implemented programs. Also, large parts of the programs are already given, with smaller parts left out, reducing the risk of execution errors significantly, while the chances of the user solving the exercise incorrectly remains quite high.

4.2.2 Instantiation Properties

As mentioned in Section 4.7, the ProCoE widget is instantiated with the path to the file containing the exercise program. This is done by providing the file path as a string to the property $file\_path$, as shown in Figure 4.1. This property shows the widget where to find the exercise program, and it is the bare minimum required for the widget to run, and it is the only property included in the prototype.

Another property, $solution\_trigger\_count$, which would allow the teacher or the user to decide how many failed attempts would be required before the result of executing the solution program would be displayed in the user interface, was also considered as an instantiation property. However, this property was ultimately rejected for two reasons. First, exposing this property to the user would allow the user to set this property to 0, giving the user the result of the solution program after their first attempt at solving the exercise. This would defeat the purpose the the $solution\_trigger\_count$, as well as potentially reduce the user's learning. The second reason for excluding the property is that there was no need for this property in the prototype, and it made no difference to the testing of the widget. Although, adding this property would allow teachers to decide the number of failed attempts necessary, it was decided that the cons outweighed the pros at the time of development. Should a better way of instantiating the widget that hid this property from the user, it would be a good idea to add this property to the instantiation properties.

As mentioned, at the time of development and testing, only the $file\_path$ property was implemented, but as the widget is developed further, more properties are likely to be made available, and the instantiation properties may be significantly changed in the future. However, the most basic version of instantiation properties (the $file\_path$ property) should be kept available.



```
In [4]: import procoe
        procoe.Procoe(file_path='example3.py')
```

Figure 4.1: Widget instantiation

4.2.3 Parser

An essential part of the inner workings of the ProCoE widget is the parser. This component is responsible for parsing the program in the exercise file into the exercise program used for generating the exercise program part of the user interface and the solution program used to evaluate the user's solution program, and parsing the user's solution program into an executable program.

The design choice that had an impact on the parser was the decision to change how the holes in the exercise program were to be marked. Sandberg (2019) proposed using the symbol **$** to mark the holes. However, this meant that the parser could only detect the hole itself and not the solution, which is integral to the program correctness checking feature of the widget. Therefor, it was decided that the holes in the exercise program were to be marked with **$)**$\langle content \rangle$**$(**, where **$)** and **$(** mark the beginning and the end of the hole, respectively, and $\langle content \rangle$ is the correct code for the hole. This way, the parser can detect both the solution and the hole itself. It was also decided to use the parenthesis inverted in combination with the **$** symbol in order to make the markings more unique and less likely to be confused with normal Python code.

4.2.4 Incorrectness Tracker

The incorrectness tracker, as explained in Section 4.7, keeps track of how many failed attempts the user has made and shows the result of the solution program to the user when they have failed a specific number of times. Currently this number is set to five. There were two design choices that impacted how this feature works.

The first design choice was made when it was discovered that attempting to execute the exact same program multiple times in a row only resulted in the program getting executed the first time, while the remaining attempts where ignored. Instead of changing this so that the program was executed every time, this bug was made into a feature of the widget.

The reason for this was to prevent the user from circumventing the incorrectness tracker, so that only different incorrect solutions where tracked. This way the user has to execute 5 different incorrect programs (or 2 if the user alternates between two incorrect programs) before receiving the result of the solution program.

The second design choice regarded which programs to track as incorrect. Part of the core functionality of the widget is to show execution warnings and errors to the user when they occur. This prompted the question ”Should programs that cause warnings or errors upon execution be considered incorrect implementations?” It was decided that programs that cause execution errors or warnings would not be considered incorrect implementations, but rather as incomplete implementations. The errors and warnings were regarded as enough feedback to the user. Because of this decision the incorrectness tracker was implemented to not track these programs.

4.3 Stakeholders

End users

The primary end users for the ProCoE widget are students and others who are at a beginner level of Python programming, and need or want to get a good amount of practice with Python programming without spending a lot of time. Their concerns regard the ease of use and the stability of the widget, and that the widget is a sufficiently better than the alternatives for solving program completion exercises.

Developers

The developers creating the ProCoE widget are naturally interested in the quality and the success of the widget, wanting the widget to be used by the end users, and that the widget is an improvement on the alternative methods for solving program completion exercises in Python.

Teachers

Teachers and educators, and their assistants, teaching Python programming at a beginner’s level are interested in the quality and stability of the widget, how convenient it is to set up and make exercises for the widget, and that the widget improves the experience of solving program completion exercises for their students.

4.4 Architectural Tactics

Architectural tactics are techniques used to achieve the quality attributes required for the system (Bass et al., 2013). An individual tactic affects the achievement of a single quality attribute. This section will describe the tactics used to achieve the quality attributes required of the ProCoE widget.

4.4.1 Modifiability

Reduce Module Size

One group of tactics used to achieve modifiability is to reduce the size of the system modules when they become large. Although the widget is a prototype, having tactics for reducing module sizes in place can help maintain the modifiability of the system during development.

Split module As module sizes grow, the cost of modifying is likely to increase. This cost can be reduced by splitting large modules into smaller modules, which should reduce the average cost of making changes. This is very useful in the process of developing the widget. As new features are added and functionality is improved and expanded, splitting the modules of the widget into smaller modules as they grow is important.

Increase Cohesion

Module cohesion is about making sure that the module's responsibilities belongs to that module. Increasing cohesion means moving responsibilities to and from modules until each module has the responsibilities that belong to it and not responsibilities that belong to other modules. Moving responsibilities from one module to another should be done if reduces the likelihood that the side effects of changes to modules affecting the other responsibilities of the original module.

Increase semantic coherence Increasing semantic coherence is about making sure that the responsibilities of a module serve the same purpose. If a module has a responsibility that is better suited to another module, then moving that responsibility to the second module increases the semantic coherence of those modules. This is very useful when developing the widget prototype. As new features are added, making sure that the modules of the widget stay coherent by adding the features to the modules they naturally belong to is important.

4.4.2 Availability

Detect Faults

To ensure the availability of a system it is important to have strategies in place for detecting or anticipating problems in the system. Because the prototype is not a continuously running service, it does not lend itself well to tactics using continuous monitoring, so it is important to use tactics that can be performed manually.

Sanity checking Sanity checking is used to check the validity of operations and outputs of components. It is often used on interfaces to examine specific information flow. Applying sanity checking to the prototype will allow developers to examine the different information flows in the widget, such as the communication between the front-end and the back-end of the widget.

Recover from Faults

Once a fault in a system has been detected it is important to have tactics in place for fixing the fault.

Rollback The rollback tactic is a preparation-and-repair tactic that allows a system to revert to a known previous good state when a failure is detected. Although this tactic is often used in systems that utilize active or passive redundancy tactics, it can be used in the widget by using older releases that are known to work well when a fault is detected. Because the widget is not a continuous service, the older version has to be introduced manually.

Software upgrade The software upgrade tactic is a preparation-and-repair tactic that intends to upgrade a system without affecting the service provided by a system when a fault in the system has been detected. In the widget this can be achieved by applying patches to the faulty parts of the widget.

4.5 Architectural Patterns

An architectural pattern establishes a relationship between a recurring situation (context) that gives rise to a problem, the problem arising from the recurring situation, and a successful resolution (solution) to the problem (Bass et al., 2013).

4.5.1 Model-View-Controller

The Model-View-Controller (MVC) pattern divides a system's functionality into three components; a model, a view and a controller. The model represents the system state, the view provides the user with a representation of the model and/or some user inputs, and the controller mediates the interaction between the model and the view. This is very useful for the ProCoE widget because the user interface has several user inputs, and changes are made to the user interface based on the user's interaction with these inputs. Although the user interface for the widget is simple, potentially making MVC an overkill, the user interface may change and become more complex as new functionality and features are added. It is a good idea to have the MVC pattern in place before the complexity of the user interface is increased.

4.6 Architectural Views

An architectural view is a complete or partial representation of an architecture from a specific perspective. Architectural views are typically expressed through diagrams.

4.6.1 Process View

The process view describes the activities and processes performed when using the widget. The process diagram shown in Figure 4.2 shows the main process explained in Section 4.7, which consists of a set of activities and sub-processes.

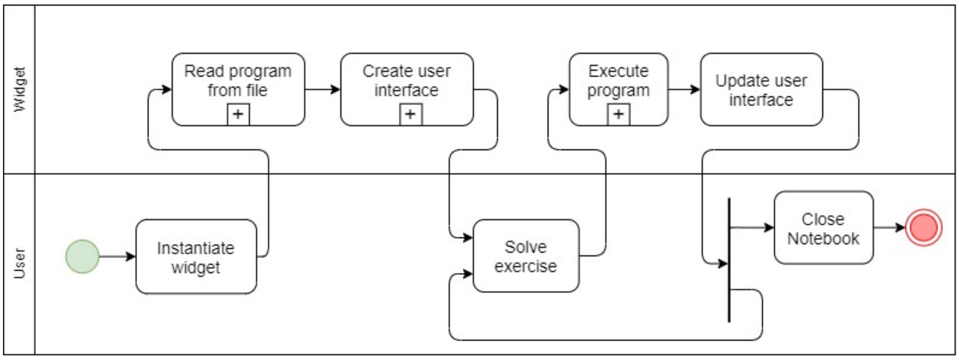


Figure 4.2: Main process

Figure 4.3 shows the process of parsing the program contained in the exercise file into two different programs; the exercise program and the solution program. The parser accomplishes this by searching each program line of the original program for the holes marked by **$)**$\langle content \rangle$**$(**. The parser adds the line with the content in the hole to the solution program, and adds the line without the content of the hole to the exercise program, as explained in Section 4.2.3. The parser loops this process until the all the lines of the original program has been searched.

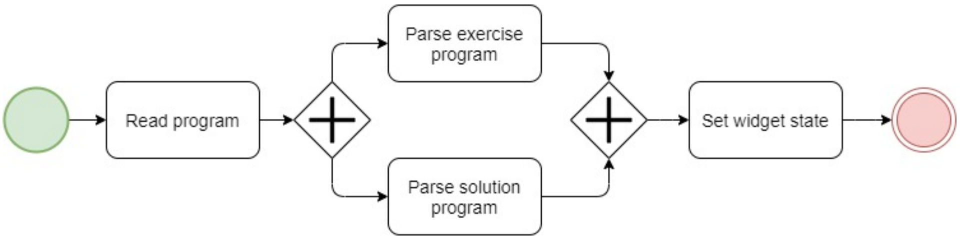


Figure 4.3: Read program from file

Figure 4.4 shows the process of generating the user interface for the exercise program. This is done by first building the core user interface with the exercise program container, the execution button and the feedback area. Then each of the program lines in the exercise program are parsed, replacing the holes in the lines with text input fields for the user to write their code in, and adding these lines to the user interface. This process is looped until there are no more lines left.

Figure 4.5 shows the process of evaluating the user's completed program. This is done by first parsing the user's program back into an executable program, and then executing the user's completed program. Once executed, the result of the user's program is checked

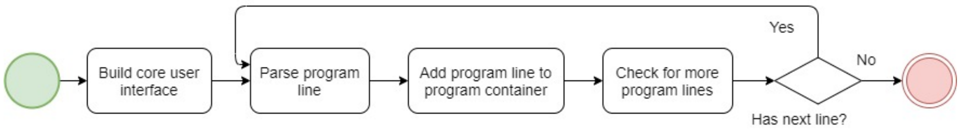

Build core user interface
Parse program line
Add program line to program container
Check for more program lines
Yes
No
Has next line?

Figure 4.4: Generate user interface

to see if the program executed correctly or resulted in an error or a warning. If the program execution resulted in an error or a warning, the result is saved to the widget state. If the user's program executes correctly, then the solution program is executed and the result is compared to the result of the user's program. If these results are equal to each other, the result of the user's program is stored as the correct answer in the widget state. If the results are not equal, the incorrect program counter is incremented, and the user's program is stored as the incorrect program in the widget state.

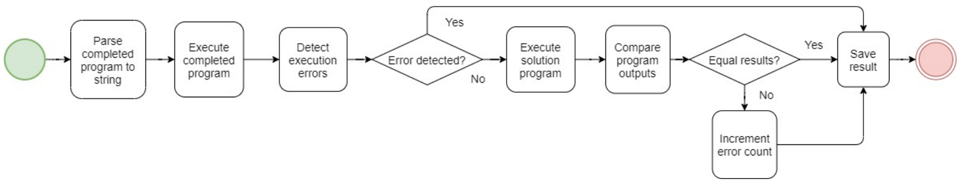


Figure 4.5: Evaluate user program

4.7 How it works

The ProCoE widget is a Jupyter Notebook widget that creates graphical user interfaces for program completion exercises in Python. This section will explain how the widget works, and will be illustrated with examples.

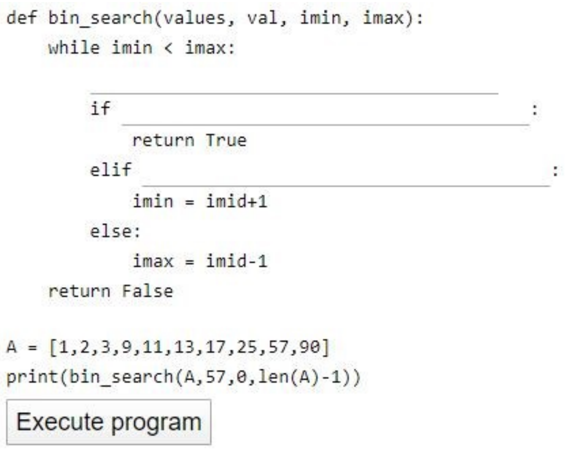
The widget is instantiated by the user by importing the package into a notebook and creating an instance of the widget with the path to the desired exercise relative to the location of the notebook. This triggers the widget to read the program contained in the exercise file, such as the program shown in Listing 4.1. The widget then parses the contained program, extracting both the exercise program used to later generate the user interface, and the solution program used to detect the correctness of the solution implemented by the user, and stores these two programs in its state.

Listing 4.1: Exercise example

```
def bin_search(values, val, imin, imax):
    while imin < imax:
        $)imid = (imin+imax)//2$(
        if $)val == values[imid]$(:
            return True
        elif $)val > values[imid]$(:
            imin = imid+1
        else:
            imax = imid-1
    return False

A = [1,2,3,9,11,13,17,25,57,90]
print(bin_search(A,57,0,len(A)-1))
```

Once the widget has parsed the exercise file, it builds the user interface, which consists of two parts; the exercise program, and the feedback. The exercise program part of the user interface contains a the exercise program itself where the incomplete parts are replaced with text inputs, and an execute button used to execute the user's program. Figure 4.6 shows the exercise program part of the user interface that is generated from the example exercise shown in Listing 4.1. Figure 4.7, Figure 4.8 and Figure 4.9 show user interface including the feedback part with the different possible feedback messages.



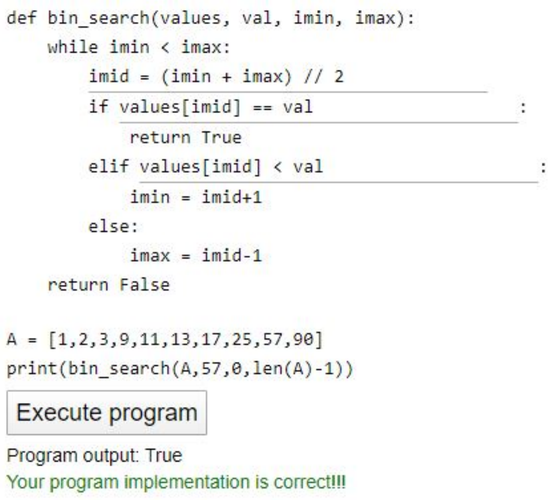
```
def bin_search(values, val, imin, imax):
    while imin < imax:
        ______________________________
        if ______________________________:
            return True
        elif ______________________________:
            imin = imid+1
        else:
            imax = imid-1
    return False

A = [1,2,3,9,11,13,17,25,57,90]
print(bin_search(A,57,0,len(A)-1))
```

Figure 4.6: User interface for example exercise

When the user has solved the exercise and clicked the execute button, the widget collects the user's program and parses it into an executable program. The widget then executes both the user's program and the solution program and compares the results of the two program to evaluate the correctness of the user's program. Once the user's program is executed and evaluated, the result of both the execution and evaluation of the user's program is returned to the user interface and displayed to the user.

Figure 4.7 shows a correct implementation of the example exercise, with the feedback message for correct implementation shown to the user. Figure 4.8 shows an incorrect implementation of the example exercise, with the feedback message for incorrect implementation shown to the user. If executing the user's program results in any warnings or errors, the user interface shows these to the user instead of the results of the evaluation of execution of the user's program. Figure 4.9 shows an implementation of the example exercise resulting in an error, with the error message shown to the user.

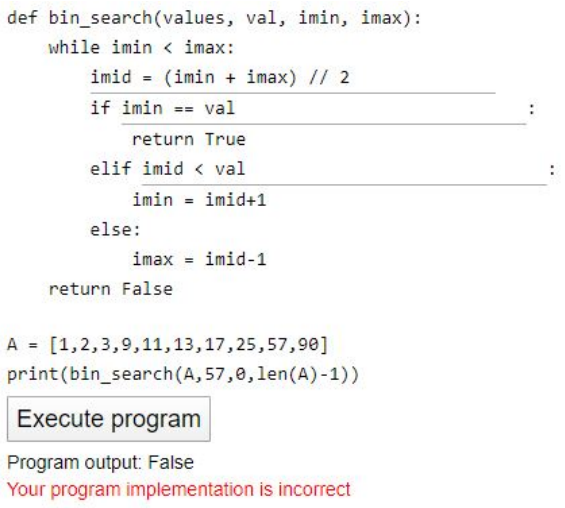


```
def bin_search(values, val, imin, imax):
    while imin < imax:
        imid = (imin + imax) // 2
        if values[imid] == val                :
            return True
        elif values[imid] < val               :
            imin = imid+1
        else:
            imax = imid-1
    return False

A = [1,2,3,9,11,13,17,25,57,90]
print(bin_search(A,57,0,len(A)-1))
```

Program output: True
Your program implementation is correct!!!

Figure 4.7: Feedback for correct implementation



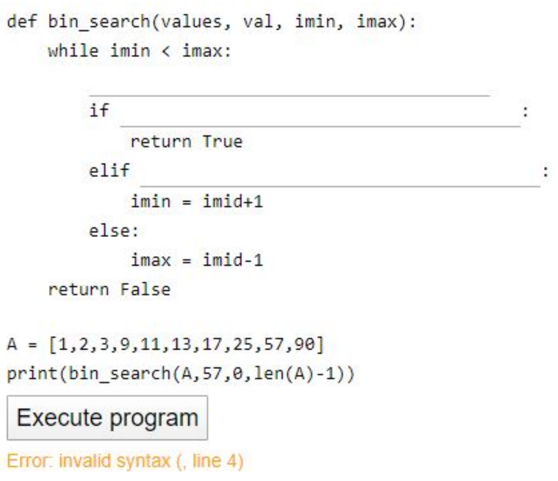
```
def bin_search(values, val, imin, imax):
    while imin < imax:
        imid = (imin + imax) // 2
        if imin == val                        :
            return True
        elif imid < val                       :
            imin = imid+1
        else:
            imax = imid-1
    return False

A = [1,2,3,9,11,13,17,25,57,90]
print(bin_search(A,57,0,len(A)-1))
```

Program output: False
Your program implementation is incorrect

Figure 4.8: Feedback for incorrect implementation

The widget tracks the number of incorrect programs the user executes, not including the cases when the program execution results in a warning or an error. If the user executes



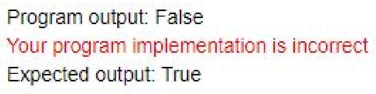
```
def bin_search(values, val, imin, imax):
    while imin < imax:
        ____________________
        if ____________________:
            return True
        elif ____________________:
            imin = imid+1
        else:
            imax = imid-1
    return False

A = [1,2,3,9,11,13,17,25,57,90]
print(bin_search(A,57,0,len(A)-1))
```

```
Error: invalid syntax (, line 4)
```

Figure 4.9: Feedback for implementation causing error/warning

5 different incorrect programs, the widget shows the result of the solution program to the user to help the user solve the exercise. Figure 4.10 shows the feedback message for the 5th incorrect implementation of the example exercise.



```
Program output: False
Your program implementation is incorrect
Expected output: True
```

Figure 4.10: Feedback for the 5th incorrect implementation

The user can at any point make changes to the code they wrote in the text inputs and execute the program again with these changes, even after the program has previously been executed.

Chapter 5

Results

5.1 User Testing

During the user test of the widget prototype, where students tested the widget then answered a questionnaire, 5 students participated. The following sections will show the results of the different parts of the questionnaire.

5.1.1 Background questions

The following list shows the result of the Background questions

Q1: Have you participated or are you currently participating in an introductory Python programming course?

- A1.1: Yes, I am currently participating in an introductory Python programming course

  Count: 0
- A1.2: Yes, I have previously participated in an introductory Python programming course

  Count: 5
- A1.3: No

  Count: 0

Q2: What level would you say your Python programming skills are?

- A2.1: Beginner

  Count: 0
- A2.2: Intermediate

  Count: 2
- A2.3: Expert

Count: 2

A2.4: Master

Count: 1

Q3: Are you currently studying or have you previously studied information technology or computer science at a college or university?

A3.1: Yes, I am currently studying information technology/computer science

Count: 3

A3.2: Yes, I have previously studied information technology/computer science

Count: 1

A3.3: No, I am currently studying something else

Count: 1

A3.4: No, I am not currently studying anything

Count: 0

A3.5: I don't want to say

Count: 0

5.1.2 SUS schema

Table 5.1 shows the results of the SUS schema. The answers range from 1 to 5, where 1 is strongly disagree and 5 is strongly agree. Using the calculation method from Jordan et al. (1996) on the answers in Table 5.1 results in an average SUS score of 88.5 out of 100.

S#	**Statement**	**1**	**2**	**3**	**4**	**5**
S1	I think that I would like to use this system frequently	0	1	3	1	0
S2	I found the system unnecessarily complex	4	1	0	0	0
S3	I thought the system was easy to use	0	0	0	1	4
S4	I think that I would need the support of a technical person to be able to use this system	5	0	0	0	0
S5	I found the various functions in this system were well integrated	0	0	0	2	3
S6	I thought there was too much inconsistency in this system	3	2	0	0	0
S7	I would imagine that most people would learn to use this system very quickly	0	0	0	0	5
S8	I found the system very cumbersome to use	4	0	1	0	0
S9	I felt very confident using the system	0	0	1	2	2
S10	I needed to learn a lot of things before I could get going with this system	4	1	0	0	0

Table 5.1: SUS schema results

The test subjects were also asked to answer three questions about their experience using the widget. The following list shows the results. The answers to the follow-up question (Q1.1) were text answers.

Q1: Compare task 1 and task 2. Did you prefer solving the task without the widget (task 1) or with the widget (task 2)?

- A1.1: I preferred using the widget

  Count: 3

- A1.2: I preferred not using the widget

  Count: 1

- A1.3: I don't have a preference

  Count: 1

- Q1.1: Why? (Optional)
  - For introductory programming learning, the widget gives a smoother introduction
  - The immediate feedback increased my confidence in the solution
  - Feels more natural and less constrained
  - When your job is to insert a very short line of code in a specific place like this task, it was a nicer user experience to use the widget
  - Depends on the intention of use. If its for learning purposes and i was a beginner, then i would prefer the widget'ed one as it specifies which part should be changed and which parts should not be changed, this removes some of the "complexity", when looking at the code. i.e. it is less claustrophobic

Q2: Task 3 in the test demonstrated the widget's program correctness feature. Did you find this feature useful?

- A2.1: Yes

  Count: 5
- A2.2: No

  Count: 0
- A2.3: I don't know

  Count: 0

The test subjects were given 2 scenarios, as described in Section 3.2.1, and asked questions about the scenarios. The answers to each of the questions were given on a scale from 1 to 5, where 1 is very likely, and 5 is very unlikely. Table 5.2 shows the results of the questions regarding scenario 1 and Table 5.3 shows the results of the question regarding scenario 2.

At the end of the survey the test subjects were given the opportunity to answer feedback questions. The following list shows the answers they gave. The answers were given as text answer.

Question	**1**	**2**	**3**	**4**	**5**
How likely is it that you would do these exercises if you could not use the widget, like in task 1?	2	1	1	1	0
How likely is it that you would use the widget when solving these exercises if you had to install Jupyter and the widget on your pc, like in the test?	2	1	2	0	0

Table 5.2: Scenario 1 results

Question	**1**	**2**	**3**	**4**	**5**
How likely is it that you would register for remote access in order to use the widget to solve the practice exercises mentioned above?	3	2	0	0	0

Table 5.3: Scenario 2 results

Q1: Did you come across any problems with the widget?

- Inserting the correct ”magic number” in task 3 resulted in the widget approving the task, while it is not the ”correct” code. Perhaps the widget could check the program with another parameter (in the background) to check that the actual code is as intended?
- Nope
- No

Q2: Are there any features that you felt were missing that should have been included in the widget?

- Nope
- No

Q3: Do yo have any other comments?

- These comment text-boxes are too small, as I can’t see my entire comment at once...
- I feel like a tighter integration with Jupyter, with code for making the widget work being hidden, is an important step - though obviously this is prototype
- Personally, i do not like the way it looks, the input field, personally i would prefer the task 1’s view of the code (i.e. actual python code), where the parts that should not be changed could have been made unchangeable instead, i feel this would increase familiarity with the language and how it looks. Especially considering that the language is made to look nice, where indentation, spacing etc has meaning.

5.2 Testing with course staff

The results of the interview with the former science assistant are listed below. The interview was conducted in Norwegian and has been translated into English.

Q: Over all, what's your impressions of the widget as it is now?

A: As a student, it's great. The important different between the widget and the traditional approach is that you get a prompt that is more than just errors. Because the problem is, for example, in this situation where my program executes, and I'm not getting any errors, but it's not correct either. This is a situation many end up in. In these situations this feedback is very good. You get more feedback than if you code an exercises in the traditional incorrectly, but it still executes.

Q: Would you have used the widget in the "Information technology, introduction" course as the widget is right now?

A: Yes, I would. It seemed very easy to set up the exercises. I could see this being used at primary school level. You could use it on quite young pupils, as well as the students who come to us at the university. Also, the "Information technology, introduction" course has a quite large exercise bank which is reused. It would likely be easy to one night sit down and turn these exercises into program completion exercises by adding the parsing symbols. Once I've done this, I would have provided my students with a lot of practice without it costing a lot. And I think it's easier to generate code that the students have to fill in, than it is to generate a good exercise that they have to write code for. Especially when testing this kind of stuff.

Q: Is it desirable to be able to decide the number of failed attempts required before showing the correct output to the student, rather than to use a hard coded value for this?

A: Yes. I think having a relatively high default value is necessary, and then students could change the value themselves after a while. Though, there will always be students who would set the value low. That's an interesting problem. What would students do if they had the choice? It might be possible to hide the widget instantiation so that students can't change the value.

Q: A feature that has been considered for the widget is the possibility of giving the students general hints about how Python works related to the exercise and about the exercise topic. This would be imported into the widget through a variable like "file_path". Is this a desirable feature?

A: It's very relevant. It helps the student contextualise their problem with a method of debugging things. They would have to go back and think about what the topic of the exercise actually is and how the relevant Python concepts actually work, which is a very good way of solving problems.

Q: Right now the widget only supports individual exercises, and is instantiated with a string value of the file path to the exercise. A different way of doing this is to use setup files with all the instantiation values. This would allow the widget to support multiple exercises per widget instance, and the same user interface would be reused for each exercise. The widget would then be instantiated with the string value of the path to this setup file. Would it be applicable for you to create and use such a file?

A: Yes. It seems to me that this is typically a practice database for the students. So I think this would be very useful. And my experience with subject teachers is that the easier it is for them to set things up, the more likely is it that it's used. This is very applicable. Students would like this.

Q: One drawback with this single setup file is that it's not possible to show the exercise description as in the tasks in the user test. The description would rather have to be included as comments in the exercise program file, which would be included in the generated user interface. Would this be a problem?

A: No. The exercise description isn't very long. Including the exercise description as comments would be doable, especially when it's this kind of exercises with a maximum of 10-15 lines of code where the point is to get practice. Program completion exercises aren't very useful for exercises with programs that are longer than this.

Q: Do you have any suggestions for improvements on the widget?

A1: Since executing the program without making any changes doesn't do anything, there should probably be some prompt or feedback that tells the student to make changes. A very new student would probably be afraid things aren't working if they clicked the execute button and nothing happened. They wouldn't necessarily think that they would have to make changes, but rather think that it's stuck and they'd have to call their student assistant.

A2: A hint button, where the students get to see all the code parts that are missing from the program when they click it. This way the students would have the code they need, but would have to figure out where the code belongs. Because that's easier for the students than having to come up with the code themselves. These hints could maybe be automatically generated since every time a hole is created in the user interface, the code that should be in that hole is extracted. This would similar to drag-and-drop exercises without actually dragging and dropping anything. The missing code parts could shown in a random order to the student. The subject teacher could also possibly add distractors, which is a pedagogical tool. By distractors, I mean code that is incorrect but could technically could have been in the holes.

Q: Additional comments.

A: A problem I have seen, especially in the context of the "Information technology, introduction" course, is that sometimes the distance between what we train the students to do in the exercises and the exam, because on the exam there are always code understanding tasks, but we don't have a good way of training the students in this. All the exercises have only the description and the students write the entire code. Program completion exercises with the widget is better for this kind of practice because the students are exposed to code that they have to understand and make changes to.

A: I had two students last year that were coders, and they set up the Jupyter server for the ”Information technology, introduction” course as part of a summer job last year. Both were second year students.

Chapter 6

Discussion

This chapter will discuss the results, how significant they are and how they answer the research questions.

6.1 Weaknesses and limitations of the study

Determining how significant the results of a study is important. This section will discuss how significant the results of the user test and the interview with the former science assistant are, how they will be interpreted, and how well they will answer the research questions.

6.1.1 Low number of participants

The results of the user test, where students tested the widget and then answered a questionnaire, show that only 5 students decided to participate. There are a few reasons why so few participated. The first is, as mentioned in Chapter 3, that few students could be recruited directly without violating their privacy. Because of this, recruitment methods that are less likely to yield many participants had to be used, and therefor a reduced number of participants was expected, but unavoidable.

Another reason is that the testing period overlapped with the exam preparation period, making it even less likely that students would choose to participate in the test. This was also unavoidable, because of the development of the widget prototype and the preparations for the testing.

The third reason is that all schools in the Norway, including the Gløshaugen campus, were closed due to the Covid-19 pandemic. This made it impossible to recruit test participants on the campus and had a severe impact on the number of participants.

This low number of test participants has a strong impact on the significance of the gathered results. Because there were so few participants, the results will have no statistical significance, and performing statistical analysis of the results would not be useful. But while a conclusion with statistical significance based on the results gathered in this study is not possible, the results will still indicate answers to the research questions.

6.1.2 No beginners participated

In the questionnaires the test participants answered, they were asked what level they considered their Python programming skills to be. Two participants answered intermediate, two answered expert, one answered master, and none answered beginner. Because the widget and the type of exercises the widget was made for are designed for beginner programmers, the answers given by the test subjects that answered expert or master are less likely to be representative of beginner programmers. These participants have previously been beginners, and can imagine how a beginner would experience widget, so while their answers are unlikely to be representative of beginners, they can still give indications of how beginners would experience the widget.

The two participants who answered intermediate on the question about skill level, on the other hand, are more likely to give answers that would likely be more similar to beginners' answers. While they are not complete beginners, they are closer and can likely better understand how a beginner would experience the widget, and their answers to the questionnaire would likely be more accurate than those of the other participants.

6.1.3 Few test tasks

The number of tasks in the user test was limited to only three. There were multiple reasons for this, as stated in Chapter 3. This limited number of test tasks has an impact on the significance of the results gathered in the questionnaire, because the test subjects had limited opportunity to use the widget. Had there been more test tasks, the test subjects would have gained more experience with the widget and a more complete understanding of the purpose of the widget and how it works. While the answers given by the test subjects in the questionnaire may not have changed, each answer would be stronger, and evaluation based on those answers would be more secure.

6.1.4 Short questionnaire

The number of questions included in the questionnaire was also limited in order to keep the test times short and attract students to participate in the test. However, this means that questions that could have provided useful data were left out. Additional questions that could have been included in the study were discovered after the user testing was completed. Still, the questions in that were included in the questionnaire should give the data necessary to answer the research questions.

6.1.5 One test per test subject

Another weakness of this study is that the user testing had to be limited to a single test per test subject. Observing how the students use the widget over time was not possible, and therefor it is not possible to determine if students who start using the widget will keep using it throughout the duration of the course. Nor was it possible to observe the students' learning outcome from using the widget over time.

6.1.6 Interview with a former CS1 course staff member

In addition to testing the widget on students, an interview of a former science assistant for the ”Information technology, introduction” course at NTNU was conducted. While the results of a single interview with a former CS1 course staff member are less significant than the results of multiple CS1 course staff members from different schools and universities, the interview conducted with the former science assistant yielded useful and interesting results. And because of the experience the former science assistant has as a former CS1 course staff member, and their experience working with beginner Python programmers, the results of the interview are still significant and are likely to provide good answers to the research questions.

A more comprehensive study, where the widget is introduced into a CS1 Python programming course, such as the ”Information technology, introduction” course, would likely yield more reliable results because the number of observed subjects would be all the students participating of the entire course, which could be in the hundreds. Observing the students use of the widget for a few weeks would provide a strong basis for determining whether or not beginner students would use the widget, and it would provide the opportunity to analyse how the number of students who use the widget changes over time. Studying the learning effects of the widget could also be in such a study. The results from such a study would be far more representative because subjects being observed in the study would for the most part be beginner programmers.

6.2 Will students use the widget?

One of the research questions in this study is whether or not the widget prototype will be used by students in CS1 Python programming courses if they had access to it. This section will discuss this by first discussing the user-friendliness of the widget and then discussing if students would prefer using the widget over the traditional method of solving program completion exercises.

6.2.1 User-friendliness to students

To provide a basis for determining the user-friendliness of the widget, the test subjects were asked to fill out a SUS schema. The answers they gave resulted in an average SUS score of 88.5, which is considered an excellent score (Bangor et al., 2009), and indicates that the widget prototype is very user friendly. However, as previously discussed, the small number of participants means that, while the SUS score may be accurate, it is somewhat unreliable.

The answers to each of the questions in the SUS schema are, for the most part closely grouped. For 7 out of the 10 questions, all students answered either with a degree of agreement (answered 4 or 5) or a degree of disagreement (answered 1 or 2). Two of the remaining questions (S8 and S9), while not as closely grouped, still show a significant unity, with the average answer being either a degree of agreement or a degree of disagreement. While these results may be flukes due to the low number of participants, this low answer variance indicate that the SUS score might not be entirely random.

One question, on the other hand, did not yield the same unified agreement or disagreement as the other questions. The answers to S1 were grouped in the middle (answered 2, 3 or 4), making this answer stick out and affect the SUS score quite a bit. The reason why the answers to S1 stick out is that the widget is intended to be used by beginner programmers to learn programming and gain experience, while the test subjects were intermediate, expert and master programmers in Python. While an intermediate Python programmer might want to use this widget to get some practice and improve their Python programming skills, an expert or a master is far less likely to want to use this. Because 3 out of 5 participants were expert of master Python programmers, the result of this statement is particularly unreliable.

With that said, the results of the SUS schema do indicate that the Widget prototype is user-friendly. And while the SUS score is somewhat unreliable, it is also unlikely to be a complete fluke.

6.2.2 Students' preference

In the questionnaire the test subjects were asked if they preferred using the widget or the traditional approach when solving program completion exercises. In addition to answering this question, they were asked why they preferred one approach over the other.

The answers to the preference question were varied. Three of the test subjects said they preferred the widget, explaining that ”For introductory programming learning, the widget gives a smoother introduction”, ”The immediate feedback increased my confidence in the solution” and ”When your job is to insert a very short line of code in a specific place like this task (task 2), it was a nicer user experience to use the widget”. These answers list some of the reasons why the widget was developed in the first place, and shows that the widget has worked as intended for these three test subjects.

One of the test subjects answered that they had no preference between the widget and the traditional approach. They explain that it ”Depends on the intention of use. If it's for learning purposes and I was a beginner, then I would prefer the widget'ed one as it specifies which part should be changed and which parts should not be changed. This removes some of the ”complexity”, when looking at the code. I.e. it is less claustrophobic.”. While the test subject originally had no preference, they would prefer to use the widget had they been a beginner, which is as expected. An expert or a master would not necessarily see much point in using this widget themselves, but neither are they a part of the target group for the widget. Although the test subject is an expert Python programmer and can only recall how it was to be a beginner, their answer still indicates that using the widget is the preferable approach for beginners when solving program completion exercises.

The last test subject answered that they preferred the the traditional approach over the using the widget. Their explanation was that it ”Feels more natural and less constrained”. This response is not surprising because the test subject is a master programmer in Python, and is likely used to writing programs with the freedom to change any part of any existing code. Program completion exercises are by nature constrained, however the traditional approach allows for changes to the original code to be made. The widget on the other hand removes this possibility, resulting in a more constrained version of the exercise. While this was meant to be helpful to beginner programmers, more experienced programmers may have issues with this. As the test subject is a master Python programmer, and did not offer any preference or explanation from the perspective of a beginner, their answer is less likely to be representative of the preference of beginner programmers.

While the test subjects were not beginners, their preferences and their explanations for those preferences still indicate that beginner students would likely prefer to use the widget over the traditional approach when solving program completion exercises.

6.3 Will teachers use the widget?

Another of the research questions is whether or not teachers in CS1 Python programming courses will use this widget in their exercise program. This section will discuss this, by first discussing how useful teachers expect the widget will be to their students, then discussing how practical it would be for teachers to create exercises for the widget, and finally discuss the effort teachers can expect to spend when using the widget in their exercise program.

6.3.1 Usefulness in teaching beginner programming to students

In the interview with the former CS1 science assistant, several topics were discussed. One of these was how helpful the widget is expected to be to beginner students. The interviewee was positive and expected that the widget would be quite helpful to beginner students, saying that ”You get more feedback than if you code an exercise in the traditional incorrectly, but it still executes” and ”... on the exam there are always code understanding tasks, but we don't have a good way of training the students in this. ... Program completion exercises with the widget is better for this kind of practice because the students are exposed to code that they have to understand and make changes to”. They also indicated that the widget could be useful in teaching Python programming to younger people, saying that ”I could see this being used at primary school level. You could use it on quite young pupils, as well as the students who come to us at the university.”

The interview indicates that the widget is likely to be useful. To conclusively determine the widget's usefulness, it would have to be use in a CS1 Python programming course for a longer period of time, and data collected throughout the testing period.

6.3.2 Practicality for teachers

Another topic that was discussed during the interview was how practical the widget would be for teachers. It was expected that creating exercises for the widget would be quite practical, since the teacher would simply have to write a program and then add the parsing symbols ('**$)**' and '**$(**) to the program where they wanted the holes to be. This was confirmed by the former science assistant, who said that ”I think it's easier to generate code that the students have to fill in, than to generate a good exercise that they have to write the code for”. But interestingly, the former science former science assistant expected this process to be even easier than expected, saying that ”Also, the ”Information technology, introduction” course has a quite large exercise bank which is reused. It would likely be easy to one night sit down and turn these exercises into program completion exercises by adding the parsing symbols.”. It is not unlikely that other CS1 Python programming courses also have similar banks with exercises that could be turned into program completion exercises for the widget. This makes it easier for teachers to get started with the widget because they can use existing exercises in the beginning. Once the existing exercises have been converted, the teachers would have to create exercises for the widget from scratch.

6.3.3 Expected effort from teachers

Teachers can expect to put in a bit of effort in order to implement the widget as part of their exercise program. Some of this effort would need to be spent on creating exercises for the widget. Since the widget allows students to solve several exercises in a short amount of time, it would be up to the teachers to create as many exercises as they believe the students would need. The effort required to make all these exercises could be significantly reduced if they have access to a bank of exercises that can be converted into program completion exercises, as suggested by the former science assistant.

The effort spent on creating exercises for the widget can also be expected to be lower than the effort in creating exercises that the students have to write the code for. This is due to the practicality of creating program completion exercises, as the former science assistant stated in the interview.

The widget was specifically designed to be used primarily for voluntary practice exercises, so getting students to do these exercises is likely to require some effort. Teachers can expect to have to set up a Jupyter server with the widget installed that the students can use to do the practice exercises remotely without downloading and installing anything. The results from the user test indicates that, while some students are likely to do voluntary practice program completion exercises if they have to download the exercises, others are unlikely to do them, even if they can use the widget. The likelihood of students doing the voluntary practice exercises increase significantly when all they have to do is register an account on a website in order to get access to both the exercises and the widget. So putting in the extra effort of setting up a server is likely to be worth it.

The teachers can however reduce the effort in setting up a Jupyter server by employing students with programming experience to set up the server for them. This has been successfully done in the ”Information technology, introduction” course, where two second year students set up the Jupyter server currently in use in the exercise program in the course.

Teachers will likely have to put in some effort to utilize the widget in the exercise program. This effort can be reduced significantly if they can convert existing exercises to program completion exercises, and by employing students to set up a Jupyter server for them.

6.4 Improvements and new features

Identifying improvements and new features is important in determining how the widget should be developed in the future in order to improve it's functionality, usefulness and user-friendliness, and thereby increase the likelihood that the widget will be used. This section will discuss the different improvements and new features that were found in the user test and the interview. The features proposed to and by the former science assistant are not supported by the widget.

6.4.1 User test

At the end of the questionnaire in the user test, the test subjects were asked if there were any features that they felt the widget was lacking and if they had any other comments to the widget. While no features seemed to be lacking, they had two suggestions for areas that could be improved. The first suggestion was that the widget should be integrated more with tightly with Jupyter, with the code for the widget work behind the scenes. This is unfortunately not possible on the widget side, as the widget needs to be instantiated with some values in order to run. Making the widget code work behind the scenes would have to be done inside the notebook rather than the widget, so exploring the possibility of hiding the code block used to instantiate the widget may accomplish this.

The second area of improvement is how the user interface created by the widget looks. One test subject said that ”Personally, I do not like the way it looks, the input fields.I would prefer the task 1's view of the code (i.e. actual python code), where the parts that should not be changed could have been made unchangeable instead. I feel this would increase familiarity with the language and how it looks. Especially considering that the language is made to look nice, where indentation, spacing etc. has meaning.”. This critique of the input fields is very useful. While the input fields were given a specific size in order not to give the students any hints as to how much code should be written in the fields, this seems to have caused the user interface generated by the widget to be too different from how the Python language looks. This could potentially be fixed by making the width of the input fields change to fit the content of the field, and setting the minimum size of the field to the width of a single letter. This would make the user interface created by the widget look more similar to the Python language, which is important.

The test subjects were also asked if they encountered any problems with the widget. While they didn't encounter any problems, one of the test subjects noted that it was possible to simply insert the correct number in the input field in task 3 and thus get a correct implementation. Though this is an interesting observation, it does not have anything to do with the widget specifically, but rather with the task itself. The parameter check proposed by the test subject is not relevant to the widget because the same problem would be present if the widget wasn't used when solving the task. It would be up to the exercise creator to create exercises that don't have such ”shortcuts”.

6.4.2 Interview

During the interview, some improvements and new features to the widget were suggested to the former science assistant. One of these improvements was to make the variable that determines how many failed attempts the students needs to make before they see the result of the solution exercise available to the teacher, so that they could decide how many attempts would be required. The former science assistant was interested in this feature, though they thought that this might lead the students to misuse the variable by setting it to zero so that the they would get the answer immediately. This might not be a problem if it was possible to hide the code window containing the widget instantiation. Finding a way to prevent the students from misusing this variable would be important for this improvement to be useful.

One of the features that were presented to the former science assistant was to give teachers the opportunity to give hints about the topic of and the programming concepts involved in the exercise. The former science assistant thought this was very relevant as this would help the students take a step back and think about the topic of the exercise and how the relevant Python concepts actually work. This response indicates that it would be useful to include this feature in the widget.

Another feature that was suggested to the former science assistant was to make it possible for teachers to create a single setup file that would contain all the instantiation values used by the widget, and hide these variables from the students. The science assistant found this feature interesting. The use of such a file would make it easier for teachers to set up notebooks with many program completion exercises by only having to change this single file rather than many notebook code windows. The teacher could also use future features more easily, as these features would use setup file when relevant. It would also allow the widget to reuse the user interface when the student has solved one exercise and then moves on to the next, thereby improving the student's experience using the widget.

The addition of the setup file feature would mean that the exercise descriptions could no longer be shown to the user as they were in the user test, which might make teachers less likely to use the widget. When the former science assistant was asked if they thought this would deter teachers from using the widget, they said no. Since the exercise description wouldn't be long, it could just be added as a comment in the exercise program. Based on this and the positive feedback the former science assistant gave to the feature itself, the set up file feature should most likely be one of the features included in the widget in future development.

In addition to the suggested improvements and new features, the former science assistant had some suggestions of their own. The first suggestion was to improve the feedback the student gets when executing the same program multiple times. The prototype gave no feedback when the same program was executed repeatedly, which the former science assistant pointed out might confuse a new student and cause them to think something is wrong. The former science assistant suggested improving this by displaying a prompt that tells the student to make changes before executing the program. This improvement is important for the widget because the purpose of the widget is to improve the student's experience when solving program completion exercises, and any confusion caused by the widget should be fixed quickly. Fixing this issue should be prioritised in future development.

The former science assistant also suggested implementing a hint button that allows the student to see a shuffled list of all the parts that are missing from the exercise program. This is a very interesting feature that adds a drag-and-drop-like aspect to the widget. If a student gets stuck on an exercise, the hint button would provide helpful hints without giving the exact answer. The hints shown to the students could be automatically extracted from the program in the exercise file when it gets parsed. Considering how simple this would be to accomplish and the possible benefit students would have from this feature, it should probably be implemented during future development of the widget. The feature would however require testing in order to determine how its use should be restricted.

6.5 The widget vs. the traditional approach

The final research question of this study is whether or not using the widget is an improvement over the traditional approach when solving program completion widget. The results of the user test indicate that beginner Python programmers may prefer using the widget over the traditional approach, though some improvements on the user interface may be warranted. The former science assistant was positive and pointed out that the widget provides better feedback than the traditional approach.

The widget is also a piece of software and can be improved with new features, such as the ones discussed above. This ability to add more features requires the use of a software and is not possible with the traditional approach. Although the widget requires more effort in the ways installation and setup, it would still likely be an improvement over the traditional approach because of current and future features that are not available to the traditional approach.

Chapter 7

Conclusion

In this study the creation and testing of the ProCoE widget prototype has shown a custom Jupyter widget that creates user interfaces for program completion exercises in Python. Although it is impossible to conclusively determine with any statistical certainty if this widget will be used by hundreds or thousands of beginner Python programmers based on the results of testing the widget on five non-beginner Python programmers, the results still indicate that beginners are likely to find the widget user friendly and that beginners may be likely to prefer using the widget over the traditional approach when solving program completion widgets. These indications combined with the possibility of solving program completion exercises with the widget remotely, rather than downloading and installing Jupyter notebook and the widget, indicate that students may be likely to use the widget if it was made available to them.

In addition to testing the widget prototype on students, an interview with a former science assistant for the ”Information technology, introduction” course at NTNU was conducted in order to determine if teachers in CS1 Python courses are likely to utilize the widget in their exercise programs. While the results of this single interview isn’t enough to conclusively determine if the widget will be used by many teachers in different CS1 Python courses, the results of the interview indicate that the widget may be useful in teaching beginner programming to students and that the creation of exercises for the widget is quite practical. Although teachers can expect to spend quite a bit of effort setting up a Jupyter server and making exercises for the widget, this effort can be greatly reduced by employing students with good programming experience to set up the server for them, and by converting existing exercises into program completion exercises. Combining the previous result based indications and the reducible effort in using the widget, with the inclusion of the features suggested and identified in the interview, indicate that teachers in CS1 Python courses may be likely to utilize the widget in their exercise programs.

Because the widget is a software, and can be continuously improved with new features that are unavailable to the traditional approach, the widget is likely to be an improvement over the traditional method of solving program completion exercises. The results of the user test and the interview with the former science assistant also indicate that the widget may be an improvement over the traditional approach.

Bibliography

Bangor, A., Kortum, P., Miller, J., 2009. Determining what individual sus scores mean: Adding an adjective rating scale. Journal of usability studies 4 (3), 114–123.

Bangor, A., Kortum, P. T., Miller, J. T., 2008. An empirical evaluation of the system usability scale. International Journal of Human–Computer Interaction 24 (6), 574–594. URL `https://doi.org/10.1080/10447310802205776`

Bass, L., Clements, P., Kazman, R., 2013. Software Architecture in Practice, 3rd Edition. Addison-Wesley.

Bergin, S., Reilly, R., 2005a. The influence of motivation and comfort-level on learning to program. In: Proceedings of the 17th Workshop of the Psychology of Programming Interest Group, PPIG 05. University of Sussex, Brighton, UK, June 2005. Psychology of Programming Interest Group, pp. 293–304.

Bergin, S., Reilly, R., 2005b. Programming: Factors that influence success. SIGCSE Bull. 37 (1), 411–415.

Ellis, M. E., Hill, G., Barber, C. J., 2019. Using python for introductory business programming classes. QRBD, 237.

Gibbs, G., Jenkins, A., 1992. Teaching Large Classes in Higher Education: How to Maintain Quality with Reduced Resources. Kogan Page, Ch. 2, pp. 23–36. URL `https://books.google.co.uk/books?id=pUA9AAAAIAAJ`

Gomes, A., Mendes, A., 01 2007. Learning to program - difficulties and solutions. In: International Conference on Engineering Education. pp. 283–287.

Guerra, H., Gomes, L. M., Cardoso, A., 2019. Agile approach to a cs2-based course using the jupyter notebook in lab classes. In: 2019 5th Experiment International Conference (exp.at'19). pp. 177–182.

Hamrick, J. B., 2016. Creating and grading ipython/jupyter notebook assignments with nbgrader. In: Proceedings of the 47th ACM Technical Symposium on Computing Science Education. SIGCSE '16. Association for Computing Machinery, New York, NY,

USA, p. 242.
URL https://doi.org/10.1145/2839509.2850507

Hevner, A., R, A., March, S., T, S., Park, Park, J., Ram, Sudha, 03 2004. Design science in information systems research. Management Information Systems Quarterly 28, 75–105.

Jenkins, T., 2001. The motivation of students of programming. In: Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education. ITiCSE '01. ACM, New York, NY, USA, pp. 53–56.
URL http://doi.acm.org/10.1145/377435.377472

Jenkins, T., 2002. On the difficulty of learning to program. In: Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences. Vol. 4. Citeseer, pp. 53–58.

Jordan, P. W., Thomas, B., McClelland, I. L., Weerdmeester, B., 1996. Usability evaluation in industry. CRC Press, Ch. 21, pp. 189–194.

Lutz, M., 2013. Learning Python. O'Reilly Media, Inc.

Luxton-Reilly, A., 2016. Learning to program is easy. In: Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education. ITiCSE '16. Association for Computing Machinery, New York, NY, USA, p. 284–289.
URL https://doi.org/10.1145/2899415.2899432

Merriënboer, J. J. V., Krammer, H. P., 1990. The "completion strategy" in programming instruction: Theoretical and empirical support. In: Research on Instruction: Design and Effects. Educational Technology Publications, Inc., pp. 46–60.

Merriënboer, J. J. V., Kreammer, H. P., 09 1987. Instructional strategies and tactics for the design of introductory programming courses in high school. Instructional Science 16 (3), 251–285.

Nelson, G. L., Xie, B., Ko, A. J., 2017. Comprehension first: Evaluating a novel pedagogy and tutoring system for program tracing in cs1. In: Proceedings of the 2017 ACM Conference on International Computing Education Research. ICER '17. Association for Computing Machinery, New York, NY, USA, p. 2–11.
URL https://doi.org/10.1145/3105726.3106178

Ozgur, C., Colliau, T., Rogers, G., Hughes, Z., Bennie, E., 07 2017. Matlab vs. python vs. r. Journal of data science: JDS 15, 355–372.

Petersen, A., Craig, M., Zingaro, D., 2011. Reviewing cs1 exam question content. In: Proceedings of the 42nd ACM Technical Symposium on Computer Science Education. SIGCSE '11. Association for Computing Machinery, New York, NY, USA, p. 631–636.
URL https://doi.org/10.1145/1953163.1953340

Robins, A., Rountree, J., Rountree, N., 2003. Learning and teaching programming: A review and discussion. Computer Science Education 13 (2), 137–172.

Sandberg, D., 2019. Program completion widget for jupyter notebook: A litterature study.

Trautwein, C., Bosse, E., 2017. The first year in higher education—critical requirements from the student perspective. Higher Education 73 (3), 371–387.

Winograd, T., 1996. Bringing design to software. ACM.

Winograd, T., 1997. The Design of Interaction. Springer New York, New York, NY, Ch. 12, pp. 149–161.
URL `https://doi.org/10.1007/978-1-4612-0685-9_12`

Winslow, L. E., 1996. Programming pedagogy — a psychological overview. In: SGCS. pp. 17–22.

Zastre, M., 2019. Jupyter notebook in cs1: An experience report. In: Proceedings of the Western Canadian Conference on Computing Education. WCCCE '19. Association for Computing Machinery, New York, NY, USA, pp. 1–6.
URL `https://doi.org/10.1145/3314994.3325072`

Zingaro, D., Petersen, A., Craig, M., 2012. Stepping up to integrative questions on cs1 exams. In: Proceedings of the 43rd ACM Technical Symposium on Computer Science Education. SIGCSE '12. Association for Computing Machinery, New York, NY, USA, p. 253–258.
URL `https://doi.org/10.1145/2157136.2157215`