The Development of Design Guidelines for Educational Programming Environments

Bedour Alshaigy

OXFORD BROOKES UNIVERSITY

A thesis submitted in partial fulfilment of the requirements of the award of Doctor of Philosophy

School of Engineering, Computing and Mathematics
Oxford Brookes University

September 2017

Abstract

Introductory programming courses at university are currently experiencing a significant dropout and failure rate. Whilst several reasons have been attributed to these numbers by researchers, such as cognitive factors and aptitude, it is still unclear why programming is a natural skill for some students and a cause of struggle for others. Most of the research in the computer science literature suggests that methods of teaching programming and students' learning styles as reasons behind this trend. In addition to the choice of the first programming language taught.

With the popularity of virtual learning environments and online courses, several instructors are incorporating these e-learning tools in their lectures in an attempt to increase engagement and achievement. However, many of these strategies fail as they do not use effective teaching practices or recognise the learning preferences exhibited by a diverse student population. Therefore this research proposes that combining multiple teaching methods to accommodate different learners' preferences will significantly improve performance in programming.

To test the hypothesis, an interactive web based learning tool to teach Python programming language (PILeT) was developed. The tool's novel contribution is that it offers a combination of pedagogical methods to support student's learning style based on the Felder-Silverman model.

First, PILeT was evaluated by both expert and representative users to detect any usability or interface design issues that might interfere with students' learning. Once the problems were detected and fixed, PILeT was evaluated again to measure the learning outcomes that resulted from its use. The experimental results show that PILeT has a positive impact on students learning programming.

Relevant Publications

Quintin Cutts, Peter Donaldson, Elizabeth Cole, **Bedour Alshaigy**, Mirela Gutica, Arto Hellas, Edurne Larraza-Mendiluze, Robert McCartney, Elizabeth Patitsas, and Charles Riedesel. 2017. Searching for Early Developmental Activities Leading to Computational Thinking Skills. In Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '17). ACM, New York, NY, USA, 393-393. DOI: https://doi.org/10.1145/3059009.3081332

Alshaigy, B. 2017. Evaluation of PILeT: Design guidelines, usability and learning outcomes results. Global Engineering Education Conference (EDUCON), IEEE, 2017.

Dennis Bouvier, Ellie Lovellette, John Matta, **Bedour Alshaigy**, Brett A. Becker, Michelle Craig, Jana Jackova, Robert McCartney, Kate Sanders, and Mark Zarb. 2016. Novice Programmers and the Problem Description Effect. In Proceedings of the 2016 ITiCSE Working Group Reports (ITiCSE '16). ACM, New York, NY, USA, 103-118. DOI: https://doi.org/10.1145/3024906.3024912

Alshaigy, B., Kamal, S., Mitchell, F., Martin, C. and Aldea, A., 2015, November. Pilet: an interactive learning tool to teach python. In Proceedings of the Workshop in Primary and Secondary Computing Education (pp. 76-79). ACM, 2015.

Contents

Chapter One: Introduction

1.1 Background of Research Problem

With the rapid growth of internet technologies and its application, there has been an exploding demand in the industry for graduates with computing expertise to fill those numerous employment opportunities. However the number of applicants for CS degrees has plummeted dramatically by 28.7% for undergraduate courses compared to 10 years ago (Universities UK 2015) and by 5% from 2016 to 2017 (Higher Education Student Data 2017). This results in a small number of qualified graduates and a shortage in the industry for information technology skills. Upon further investigation those numbers have been attributed to students' negative attitudes and aversion to programming courses in general. Although several factors, while not exhaustive, were responsible for the problem, such as the students' failure to acquire rudimentary programming skills, understanding the syntax and semantics of a programming language (Robins et al 2003) and critical cognitive problem solving skills (T. Beaubouef et al 2001) to name a few, we are far from fully understanding the underlying reasons behind different progression rates amongst them. Evidence from multinational studies (Lister et al 2004, McCracken et al 2001) and literature review strongly implicated teaching techniques adopted whilst teaching programming (J. Allert 2004), students' learning preferences (Lahtinen et al 2005), in addition to the choice of the first programming language taught (A. McGettrick et al 2005a) and usability issues found in e-learning tools (Ardito et al 2004). Whilst these reasons were discovered almost a decade ago, the findings are still relevant today as more and more studies are reporting the same causes without definitive solutions (Bosse and Gerosa 2017, Gomes et al 2012, Özmen and Altun 2014)[1].

1.2 Thesis Statement and Objectives

With the prevalence of mobile devices and e-learning, many instructors are in favour of using innovative courseware, massive open online courses and virtual learning environments in teaching. Some of these approaches have been proven effective in improving student' retention and engagement such as using games to teach a concept (Eagle et al 2008) or providing a user friendly environment with less cryptic error messages and feedback to support the students in programming (Murphy et al 2008). However many of these tools are either too complex to use due to design configurations or do not take into account the learning differences found in a diverse cohort of students. As a consequence, students grapple with understanding programming theories and practice, exhibit decreased enthusiasm towards the subject, and fail exams. Therefore this thesis states that

Combining multiple teaching methods to accommodate different learners' preferences will significantly improve performance in programming

To test this hypothesis, an interactive web based learning tool to teach Python programming language (PILeT) was developed. The tool's novel contribution is that it offers a combination of pedagogical methods to support the student's learning style. Therefore, each programming concept will be explained using videos, reading material, examples, exercises and puzzles independently, or in combination with other approaches. Additionally, multiple choice questions are available at the end of each lesson to assess the students' understanding of the taught concept. This way, each student can

[1] For this reason, throughout the dissertation, seminal papers were used as references dating back to 2005. In instances where new findings are being reported, the latest research is cited.

learn from the teaching technique they are most comfortable with, or use a mixture of several methods to support their learning (Pollock and Harvey 2011a).

An extensive literature review preceded the development stage to determine essential design guidelines for building the pedagogical tool. The areas covered:

1. Predominant programming problems.
   1.1. Teaching methodology.
   1.2. Students' learning styles.
   1.3. Choice of first programming language taught.
2. Evidence of existing relationships between learning styles and teaching practices.
3. Analysis of existing pedagogical software and environments.
4. Students and instructors use of pedagogical tools.

Following this process, a clear set of objectives was devised to aid with the hypothesis. These objectives are:

1. Identify essential design guidelines for the development of pedagogical tools.
2. Develop the interactive tool (PILeT) based on those guidelines.
3. A heuristic evaluation of PILeT to detect any usability problems associated with the interface design by expert reviewers.
4. Perform another usability test from the perspective of end users with the aim of evaluating their overall interaction and experience with the tool.
5. Evaluate the tool pedagogically by measuring the learning outcomes of students.

1.3 Motivation and Contribution

This research is driven by the author's ambition to contribute to the huge body of research on teaching introductory programing (Bruce and Bruce 2004, Pears et al 2007, Robins et al 2003) specifically the pedagogical tools used to support the students learning (Gomez-Albarran 2005). Several of these studies were concerned with the effectiveness of certain instruments' features on the learning outcome, for instance the benefits of code visualisation and execution for conceptual understanding (Sorva et al 2013), using games or puzzles to increase the levels of engagement amongst learners (Bayliss 2009, Curtis 2005a), or activities dedicated towards enhancement of problem solving skills. A group of those studies observed the changes in behavioural traits in students under the tool's influence. While the results of these experiments seem promising, they are in danger of being unreliable. A methodological analysis by Randolph (Randolph and J. 2007) of 352 published studies revealed that a huge number of experiments did not follow any methodological frameworks which throw into questions the validity of the results. The researchers discovered that around a third of the studies did not recruit participants; the few that did were small scale experiments that did conform to sampling techniques or follow the right guidelines in recruiting volunteers. As for scientific results, many failed to statistically analyse the data and relied instead on speculations and loose interpretations of questionnaire responses without supporting evidence. In addition, over half of the experiments did not sufficiently specify their investigation methods, and a quarter of those did not enclose the research questions or conduct a literature review before starting a new study.

The second area of concern which motives this thesis is the lack of usability testing, standard design guidelines for the development of educational software, or evidence of tool evaluation in terms of learning outcomes. As a matter of fact, some educators make the switch to digital tools to follow the latest educational trends without realising the impact of their decision on their students or themselves.

A collection of significant outcomes from published studies were as follows:

- Most of the research on pedagogical tools measured the effectiveness of a single unique feature, interactivity for example, on learning programming without taking into account other contributing factors that hinder comprehension such as cognitive skills.
- Several studies measured the success of a tool by heavily relying on quantitative results such as final exam marks without considering the students' reaction and level of satisfaction with the tool. A portion of those studies were not repeated to verify the accuracy of their results.
- Only a few of the studies in the field of educational technology inspected the teachers' and students' use of pedagogical tools. For instance, what is preventing teachers from adopting these tools to deliver the course or monitor the students' progress (Levy et al 2007)? How do students actually interact with pedagogical software (Stern, Markham, Hanewald, et al 2005)? Is it possible that they are missing out on the actual learning objectives?
- The failure of several pedagogical tools is attributed to the absence of standardised design guidelines and principles for developing those tools.

In light of these findings, this research's original contribution is to improve the teaching and learning of programming by devising a set of 11 essential design guidelines for the development of educational programming environments. These guidelines were derived by combining the results of an extensive literature review on educational software with established design guidelines for websites. In order to evaluate them, an interactive learning tool was developed which conformed to the guidelines. The tool was then tested by combining usability tests with measurements of conceptual understanding of programming knowledge in students.

1.4 Thesis structure

The overall structure of the dissertation takes the form of seven chapters. This first chapter gives a brief overview of the thesis by starting with an introduction into the background of the research problem, followed by the thesis statement and objections, thesis motivations and contribution to knowledge.

Chapter 2 contains an in depth literature review that covers common difficulties exhibited by first time students, successful programming pedagogies, learning preferences, the influence of the first programming language taught in addition to students' and instructors' use of pedagogical tools. The chapter includes evidence of an existing relationship between learning styles and teaching methods.

Chapter 3 examines and evaluates popular programming environments used by novices based on functionality and educational impact. The findings were used to develop a set of design guidelines for the development of PILeT.

Chapter 4 details the essential design guidelines for developing programming tools. It also includes a description of PILeT in addition to documentation of the system architecture and interactive features. The chapter concludes with an outline of how it complies with most of the design guidelines and learners on the Felder-Silverman Spectrum.

Chapter 5 describes the heuristic evaluation and user based testing of PILeT as conducted by expert and representative users respectively, this includes the evaluation method and results.

Chapter 6 reports the process and results of the learning outcomes evaluation of PILeT.

Finally Chapter 7 concludes with a reflection on the research’s objectives along with threats to validity and recommendations for future work.

Chapter Two: Literature Review

2.1 Chapter Overview

This chapter examines the literature on predominant programming problems commonly exhibited by first year university students, and examples of successful methods of teaching introductory programming courses. This is coupled with a number of recent studies, in computer science education, suggesting an association between students' learning styles preferences and teaching methodologies. Finally the chapter concludes with the significance of the first programming language taught at university on students' in addition to instructors' and students' use of educational tools.

2.2 Problem Description

Introductory programming courses are an integral part of computer science education; they develop logical and reasoning skills and improve problem solving abilities. Programming is customarily taught to students in their first academic year in the UK. One of the expected learning outcomes of the course is for students to create solutions that satisfy a set of requirements in a chosen programming language. While some students succeed, many others fail at achieving that goal.

The difficulty and complexity of teaching programming is not exclusive to one institute but rather universally acknowledged amongst educators (Milne and Rowe 2002a). Despite the abundant literature and recommendations on teaching introductory programming, specifically curricula and pedagogical consideration comprehensively summarised in the publication by (Pears et al 2007), the effectiveness of these strategies are debatable. And with the absence of a widely recognised pedagogical framework, it is challenging to implement these recommendations in a computer science course.

The difficulties that novice programmers experience while learning have been extensively explored over the years (Dale 2006, Lahtinen et al 2005, Tan et al 2009). These challenges have contributed to the high failure and attrition rates for a very long period (Andrew McGettrick et al 2005, Nikula et al 2011). Students' perceived attitudes towards programming have a negative effect on comprehension of concepts. There is strong anecdotal evidence, supported by the literature (Lister et al 2004, McCracken et al 2001, Robins et al 2003), suggesting that the average student lack rudimentary programming skills by the time of graduation.

For some time, there has been an increasing interest in students' learning activities and behaviour when solving programming problems (Carmo et al 2007, Tie and Umar 2010). The most recent learning analytics data from Stanford Computer Science Department and Graduate School of Education (Blikstein et al 2014) offered a new insight into the preferences that students exhibit whilst learning new programming ideas, and the influence of the teaching strategy used at the time. These findings suggest a strong relationship between students' learning style and teaching methods.

Combined together, these findings highlight the significance of identifying appropriate pedagogies to address common challenges faced by a diverse group of students learning programming.

2.3 Predominant Programming Problems

One of the greatest challenges that have been frequently reported by first year computer science students is learning to program. Programming requires an exhaustive comprehension of abstract concepts in addition to advanced logical skills in the domain of problem solving (Robins et al 2003, Winslow and E. 1996). This problem is not new; this difficulty has also been acknowledged by teachers who are struggling to instil those imperative skills to students over the years (J Carter and Jenkins 2010, A. McGettrick et al 2005b). A commonly recognised issue is novices' inability to translate a particular programming problem expressed in English to the corresponding solution in programming code. This is largely due to students struggling with decomposing a problem into sub-problems and implementing a solution to each part.

Many of the conclusions on studies exploring programming difficulties provided significant information on cognitive and comprehension skills in novice students especially in the domain problem solving (T. Beaubouef et al 2001) and deriving a solution strategy (Cutts et al 2006). Researchers observed that whilst were able to articulate a problem and suggest solutions verbally; they were not able to express it in code. This is not different from abstraction (Bouvier et al 2016, McCracken et al 2001), which is extracting rules and concepts from a problem or example. In addition, those studies revealed that students lack fundamental problem solving techniques that requires the identification of basic problem elements, the relationship between them, and the operation and steps necessary for constructing a solution. They also fail to apply the same problem solving techniques on similar problems (Shute 1991). While these challenges are not applicable to all learners, it is not uncommon, especially in a programming course, to end up with two students' group at the end of the academic year, the "experts" and the "novices" (Bornat and Dehnadi 2008, Robins 2010, Winslow and E. 1996).

In an attempt to discover prevalent learning difficulties found in mixed ability student groups, several studies investigated the role of personal attitudes and individual traits manifested by students on their programming abilities. Those attitudes include self-efficacy; one's personal belief in succeeding in a task and mental models; a description of thoughts and ideas as they are represented in the real world (Hamilton et al 2008, Ramalingam et al 2004), in addition to other factors such as motivation to learn and capability of programming (Carbone et al 2009), engagement and interest in the topic (Corney *et al.*, 2010), encouragement and support by instructors and peers (Brenda Cantwell Wilson and Shrock 2001) and the incompatibility of teaching methodologies with learning styles (Thomas et al 2002a, Zander, Thomas, Simon, Murphy, McCauley, et al 2009).

A different group of studies shifted their focus towards diagnosing learning difficulties that students encounter whilst learning to program such as understanding the syntax and semantics of a programming language and combining them into meaningful programming codes (Robins et al 2003). Other challenges include:

- The difficulty of particular programming concepts such as: inheritance and polymorphism (Goldman et al 2008).
- Development of programming misconceptions related to the language construct for example classes and objects in Java (Kaczmarczyk et al 2010a).
- Code reading and tracing especially found in multiple choice questions in which students predict the right answer (Lister et al 2004).
- Debugging and error finding whether it is the students' own code or others (Sue Fitzgerald et al 2008).
- Difficulty of a programming language over another one (Mannila and de Raadt 2006).
- Programming paradigm; Object Oriented first vs. functional programming (Bruce 2004).
- In addition to inadequate lecture notes, textbooks (Lahtinen et al 2005), and curriculum (Pears et al 2007).

Authors	**Mental Model**	**Learning Styles**	**Teaching Methods**	**Programming Language Choice**	**Program Comprehension**	**Programming Concepts**	**Cognitive Skills**
(James Allert 2004)	x	x	x	x			
(Theresa Beaubouef and Mason 2005)			x	x			x
(Lahtinen et al 2005)		x	x		x	x	x
(A. McGettrick et al 2005b)		x	x	x			
(Milne and Rowe 2002a)	x				x	x	
(Robins et al 2003)	x	x	x		x	x	x
(Tan et al 2009)		x	x	x	x		x
(Zander, Thomas, Simon, Murphy, Ren, et al 2009)		x		x		x	
No. Times Outlined	**3**	**6**	**6**	**5**	**4**	**4**	**4**

Table 2.1 Common Programming Problems Reported by Students in Literature

To summarise, Table 2.1 presents a list compromising of the most seminal literature on learning difficulties found in first year undergraduate students as acknowledged by the CS Education experts, along with the most common programming problems reported in those publications.

Students' mental model, program comprehension, programming concepts and cognitive skills were constantly reported as causes of failure of students in programming courses in their first year. Interestingly, learning styles and teaching method were repeated the most (6 times) followed by the choice of the first programming language taught (5 times). What stands out in the table is that the choice of first programming language, along with programming comprehension and concepts are specific to the domain of computer science whereas cognitive skills, mental models, learning styles and teaching methods are associated with pedagogy. This suggests that those challenges should not be examined independently but as a whole since it combines pedagogy specific issues along with computer science concepts.

2.3.1 Teaching Method

Not long ago, teaching programming involved lecture notes and textbooks taking place in static lecture halls or computer labs where content is passively delivered by the tutor and an example to practice with throughout the duration of the lecture or lab. The problem with this approach is that students cannot be expected to learn syntactical expressions of programming irrelevant to everyday occurrences using traditional teaching methodologies. Textbooks are not sufficient without adequate explanations supported by clear examples. Students can only master programming by rigorous practice and persistence. It was evident that there was a pressing need to revamp the way it was taught.

In order to improve teaching programming, many lecturers experimented with several teaching theories and methodologies. An example is adopting an object first paradigm vs. procedural first, which is still a controversial subject in computer science education circles that is yet to reach a consensus on the decision as some feel strongly about the benefits of introducing objects first to students rather than later (Astrachan et al 2005, Bruce 2004). Other teaching strategies stepped away from those paradigms all together and focused on learners' engagement by incorporating games; either by introduce programming concepts and ideas whilst playing the game or writing game assignments (Cliburn and Miller 2008), however this method is not without its drawbacks, as students reported spending more time understanding the homework requirement compared to traditional exercises (Sunt al 2011), or they get invested playing the game itself whilst inadvertently forgetting the assignment requirements, and in some cases missing out on the learning objectives or failing to transfer and apply the learning outcomes into real life problems or situations.

The literature on teaching introductory programming is vast (Pears et al 2007). The process of finding the appropriate teaching methodology and implementing it on a diverse group of learners with different abilities has potentially grave implications towards either the success or failure of students. The best teaching practices in introductory programming courses should consider the perspective of both the lecturer and the learner. Lecturers should address common difficulties experienced by novices such as the syntax and features of a programming language, problem construction and solution formulation, and misconceptions surrounding programming concepts as a result of misunderstanding or failure to build mental models. The teaching methods itemised below targets several programming problems reported by students and instructors alike. Results of these methods showed an increase in the success rate of students learning the subject.

2.3.1.1 Syntax free Approach

Adopting a syntax-free approach (Fidge and Teague 2009, Heliotis and Zanibbi 2011) improves programming competence and develops critical thinking skills (Tasneem 2012). Students define, examine and suggest possible solutions to similar problems without any consideration to the programming language. Once the answer is chosen, students start constructing a formal algorithmic scheme written in pseudo-code then convert it to the equivalent syntax of the programming language. This technique enhances generic problem solving abilities as it requires a profound understanding of the question, detecting key terminologies (e.g. goal, conditions) and strategically planning a solution.

2.3.1.2 Examples and Exercises

Examples form an indispensable part of the learning process; they illustrate new ideas and motivate students into learning. Excellent programming examples should demonstrate the language structure, style, and apply appropriate algorithms for problem solution (Borstler et al 2010). As such these examples must follow a consistent pattern that clearly exhibits the guidelines and rules that students are expected to replicate in their code in order to develop good programming habits. Additionally, examples must be carefully designed to prevent students from forming their own misinterpretations and misconceptions of programming concepts, or to lead them to poor learning tendencies (Carbone et al 2001). Furthermore, examples should correspond with the taught concept specifically without being too abstract or too complex (Katherine Malan and Halland 2004). Once the student fully understands the principle, connections could be made with other examples to show how a learned concept could be applied or combined with other concepts in different examples. This will allow the student to incrementally build his knowledge and improve his problem solving techniques.

Follow up exercises (Parsons and Haden 2006, Shuhidan et al 2011) are just as equally important; they effectively implement learned constructs and assess the level of comprehension to allow timely intervention. Assignments should encourage students to think creatively of new solutions to different situations and test different programming statements. Various question types (e.g. write piece of code, find bugs in the code) permits students to think of different perspectives of the programing language as opposed to large code writing tasks that overwhelms students struggling with writing programs involving more than one concept (Zingaro et al 2012). A number of studies have investigated the factors that interest students in assignments. Research by Hansen and Eddy (Hansen and Eddy 2007) found an existing relationship between the level of frustration and engagement in programming tasks, this means that students enjoy assignments that present some level of difficulty that increases confidence once solved. Laymen et all (Layman et al 2007) suggest different characteristics that increases students' engagement in assignments such as usefulness and niftiness. Cliburn and Miller (Cliburn and Miller 2008) argue that given a choice, student favour specific assignments over open ended tasks. Therefore, careful consideration must be given to the learning outcomes that will be assessed before selecting assignments for students.

2.3.1.3 Puzzles

Using puzzles to teach programming places less emphasis on syntax and concentrates on problem solving techniques (Curtis 2005b, Ross 2002). Puzzles have been recently employed in introductory computer programming courses (Cha et al 2007, Kawash 2012) to attract students who are initially intimidated by programming and to exemplify abstract concepts. These puzzles aim to develop the fundamental reasoning and logical skills necessary to systematically solve problems and transfer these vital skills into applicable real life situations. They can also be used as a vehicle to imperceptibly instil programming principles such as searching and sorting, recursion, and solution design. Results of an experiment conducted by (Merrick 2010) positively associated puzzles with increased motivation and participation in beginners. Puzzles in programming can take many forms from word puzzles (crossword) to game like and logical puzzles (sliding tiles, Tower of Hanoi).

2.3.1.4 Visualisation

Studies into novice programmers learning unveiled that common programming misconceptions are mainly behind programming problems. These misconceptions include: mistaking objects for classes, objects as records for storing values, and methods are a form of assignments (Holland et al 1997). Furthermore, students do not understand object behaviour once methods are invoked; their allocation in memory, or the sequence of method execution. Visualisation tools are instrumental in conceptualising these aspects by animating the dynamic interaction between different object elements and functions within different program states upon execution (Brusilovsky et al 2006). They can also be utilised to augment different programming concepts at different stages. And yet, studies have showed that the use of visualisation software is not widespread (Levy and Ben-Ari 2007) despite their proven effectiveness. Lecturers are hesitant toward adopting these tools because not only do they require practice in order to use them effectively, they also involve adapting the teaching pedagogy and curriculum to the tool which is time consuming. There is also the possibility that student might not rely on these visualisation instruments because of the technical difficulties they face whilst running the tool.

There is growing evidence to support the success of visualisation tools to teach programming with promising results. The key to their successful utilisation is to provide the necessary training and support for educators who decide to adopt and integrate this technology in the classrooms, and encourage the students' use of these tools by incorporating interactive functions to increase engagement and motivation when learning.

2.3.1.5 Implication of teaching methodology

Overall, the goal of each of the programming methods listed so far is to foster critical thinking and reasoning skills in order to improve problem solving and decision making abilities in students. Additionally these methods have shown to be successful in boosting learners' confidence and persistence when faced with frustrating problems and abstract concepts, and increase their motivation and level of engagement with the content they have learnt.

While those methodologies focus on different teaching theories and use different instruments to deliver the concepts (syntax free, examples and exercise, puzzles, visualisation), they all aim to improve students' performance in programming. And while a pedagogical technique might be successful in one group of learners, it could result in negative consequences such as loss of interest and demotivation in others. Combining those pedagogies would reconcile the differences in learning styles found in a group of leaners. To this end, lecturers must consider those factors when designing and developing programming courses.

2.3.2 Learning Styles and Preferences in Students

Recently, there has been a major increase in the number of students enrolling in computer science courses worldwide (Crump 2004), especially international students. In addition to adjustment and language issues caused by the transition, they also experience a number of challenges because of social and cultural differences in their learning. This results in a diverse student population with different learning needs.

Learning styles are identified as "cognitive characteristics, affective and psychological behaviours that serve as relatively stable indicators of how learners perceive, interact with and respond to the learning environment" (Keefe 1979).

Students of different age groups, skill and gender absorb and assimilate new information differently. For example some learners experiment and try things out as a means of learning while others are very cautious about their decisions and prefer reflective thinking before taking a course of action. Some favour written explanation to instructions and others respond better to graphical explanations in the form of pictures, drawings and charts.

Generally, a student's learning profile is based on previous experiences in which the adopted learning approach worked best for them. This preference could change however as the students' progress in their learning or face a new problem in which another learning style would be more suitable to adopt. The diversity of students' preferences has led many lecturers to reconsider the design and delivery of programming courses to reduce experienced difficulties amongst learners, and bridge the knowledge gap between them.

There are several existing learning theories since it gained popularity in the 1970s. The literature so far identifies 71 learning styles models (Coffield et al 2004), with VARK (visual, aural, read/write, kinaesthetic) and Kolb learning style model being the most employed strategies in several scientific fields namely mathematics and chemistry (Chin and Brown 2000, Pashler et al 2008). Similarly, there has been a growing interest in the value and application of learning theories in computer science education context (Haden et al 2004). The main aim of those studies was to help the students recognise their own individual learning style in order to devise teaching strategies to support students in a mixed ability classes, and improve their academic achievements. Although the validity and reliability of those theories are still questionable, a considerable amount of research has reinforced the positive impact of matching teaching strategies with learning styles. In most of those studies, the Felder-Silverman learning style was model was applied.

2.3.2.1 Felder - Silverman Learning Style Model

The Felder - Silverman learning style model (Felder and Silverman 1988) is a synthesis of several learning theories which includes Kolb's Learning style Model (Kolb 1984), the Myers-Briggs Type Indicator (Briggs Myers and Myers 1980) and the Hermann Brain Dominance Instrument (Hermann 1982). This model has been used to design teaching activities in both traditional and technology enhanced learning.

The Felder - Silverman learning Style Model divides students into four dimensions (Felder and Brent 2005) based on their responses to four questions:

1. What kind of information does the learner prefer processing: sensory (visions, noises, physical feelings) or intuitive (memories, ideas, views)? **Sensing learners** are factual, practical and partial towards traditional well established procedures to problem solutions. **Intuitive learners** are drawn to concepts, innovative and creative problem solvers.
2. What kind of sensory information is mostly suitable to the learner: visual or verbal**? Visual learners** favour pictures, diagrams, flow charts whereas **verbal learners** favour written or spoken instructions and explanations.
3. How does the learner choose to process and understand new information: actively (by physical engagement in discussions and activities) or reflectively (by way of reflection and self-examination). **Active learners** learn by experiencing things out, working with other group members and on the other hand **reflective learners** learn by thinking first and prefer to work individually.
4. How does the learner naturally develop their knowledge: sequentially (gradual development by processing one piece of information at a time) or globally (looking at the "big picture")? **Sequential learners** learn by incremental acquisition of knowledge in a systematic manner whereas **global learners** learn by acquiring knowledge randomly and making huge steps in learning by understanding how does the material they already know relate to each other until they see the big picture.

There are several advantages to the Felder – Silverman model; while most of the other learning style models classify learners based on their abilities, the Felder-Silverman model primarily focuses on the preferred learning style (Felder 1996). A learning style profile highlights the areas of strength in students and the potential practices that could hinder their learning. Additionally, although other models assign learners into either overly restrictive categories or a few groups, the Felder – Silverman model organise learners into balanced dimensions classified as (active-reflective, sensing-intuitive, visual-verbal and sequential-global). Consequently, if students exhibit a preference towards a dimension, then they will easily learn in a teaching environment that supports that dimension or experience difficulties otherwise.

Although other learning style models argue that the content design of the teaching course should be altered to accommodate diverse learners, Felder argues that it is perfectly adequate to combine a variety of teaching methods.

2.3.2.2 The Index of Learning Style Questionnaire

The Index of Learning Style Questionnaire (Felder and Soloman 1997) is a validated and reliable research instrument (J. Allert 2004, Felder and Spurlin 2005) that compromises of 44 multiple choice questions to determine the learning preference of students based on the four dimensions identified in the Felder – Silverman model. The test is freely available online, and can easily be administered by teaching faculty, or used by students independently to help them understand and improve their learning experience as its simpler to understand and score than other tests such as the Myers-Briggs test.

The index was first developed to understand learning style differences in engineering students but was later applied in a broad range of disciplines especially in computer science education research and learning technologies (Alaoutinen and Smolander 2010, J. Allert 2004, Carmo et al 2007, Thomas et al 2002b, Tie and Umar 2010, Zualkernan et al 2006).The findings from those studies have established a relationship between the preferred learning style and positive performance in programming even though the model is not relatively new. For example, results from an experiment carried out by Thomas et al in an introductory programming course showed that reflective learners scored better grades than active learners in their exams, and verbal learners achieved higher grades than visual learners (Thomas *et al.*, 2002). These results were corroborated by similar experiments carried out by Allert (James Allert 2004) and Zualkernan et al (Zualkernan et al 2006). However in the latter experiment, the sequential learners outperformed the global learner in one case study whereas the global learners outperformed the sequential learners in another case study.

2.3.2.3 Relationship Between Learning Styles and Teaching Method

There are a growing number of studies that confirms the association between learning styles and pedagogies. Their findings report a significant improvement in students' programming skills and comprehension when matched with a teaching methodology that compliments their preference. However students preferences are subject to change either over time, or due to cognitive development, or through experiences (Graf et al 2007). This might result in complete rejection of other teaching methods by students, or cause discomfort in their learning as they lack the mental skills to understand and benefit from other teaching approaches. Therefore, it is advisable to create a heterogeneous teaching environment that fulfils different learning requirements and support students' choice. This could be achieved by devising teaching strategies that benefit the majority of learners and providing supplementary material and resources for those disadvantaged by other pedagogies.

For example, in order to appeal to sensing and intuitive leaners, the teaching material should include factual information complimented with abstract concepts. It also helps to provide specific examples for sensors and interpretations and theories for intuitive learners. Visual learners could be presented with animation of code execution, which could benefit visual, sequential and active learners, accompanied by summaries or descriptions for verbal and intuitive learners.

In order to engage both active and reflective learners in lectures, classroom discussions and problem solving activities would benefit the former group and frequent breaks for thinking and content reflection would be useful to the latter group. Finally sequential learners are already at an advantage as most of any subject material is sequentially ordered and gradually develops during the course allowing students ample time to absorb and process the information. Global learners could be assisted by providing in advance an overview of the lesson and the learning outcomes to enable them to see the big picture and make connection with other content.

It might be difficult at first to plan and design teaching strategies to accommodate several students' needs due to several constraints including the heavy workload and different responsibilities of the academic staff, however these techniques could be implemented with the aid of educational technologies to disseminate the content and support students learning.

Recent developments in the field of educational technologies and hypermedia have prompted researchers to consider learning styles whilst building those tools. Studies investigating the influence of learning preferences in technology enhanced systems were increased as a result of the popularity of distance learning, and many adaptive systems were designed to provide courses that cater to different learners and fit their individual needs. Examples of those educational technologies include CS383 (Carver et al 1999), IDEAL (Shang et al 2001), MAS-PLANG (Peña et al 2002), TANGOW (Paredes and Rodriguez 2004), and AHA! (Stash et al 2006). Those technologies could be used to enhance the learning environment for students and increase their performance in programming courses.

2.3.3 Choice of first programming language taught

There are several programming languages taught at university in introductory courses (Ernie Giangrande 2007). The decision is usually influenced by the language characteristics, the choice of other universities (de Raadt et al 2002, 2004), industry preferences (Dingle et al 2000) or popularity as measured by the TIOBE index (TIOBE 2016).

Students' first experience with programming usually involves writing code in Java or C++, however there have been several arguments about the suitability of those languages for introductory courses (Bruce 2004) in addition to research exploring the benefits of each language, or comparing them objectively in an educational context (Ernie Giangrande 2007, Mannila et al 2006, Mannila and Raadt 2006).

There are a number of possible risks associated with teaching Java and C++ such as syntactical difficulty and cognitive overload. These problems arise as result of concentrating on the constructs and features of those languages such as object orientation and polymorphism and digressing from the main objective of learning programming which is problem solving (Palumbo 1990). As a result of these concerns, the new trend of teaching Python as a first programming language is gaining momentum (EDU-SIG 2016), becoming a natural choice for many educators (Agarwal and Agarwal 2006, Leping et al 2009, Radenski 2006) and a favourable option amongst students(Sanders and Langford 2008, Stephenson 2009).

Python is an interactive programming language that has been developed as a teaching and learning instrument. The language has several desirable attributes such as expressive syntax and meaningful semantics akin to natural language that allows students to explore different expressions and functions and experiment with several problem solutions. It also satisfies the criteria of language features for programming languages used for teaching (Milbrandt 1995) which are:

- Simple and easy to use by students.
- Structured in its design.
- Powerful computing capacity.
- Simple syntax and meaningful names for keywords.

- Dynamic definition of variable types whilst typing.
- Strict format of input and output to foster good programming models and rules.
- Instant error feedback and powerful tools to encourage debugging and testing.
- Produces visually appealing graphical user interface components for small and large scale projects.
- Facilitates easy transition to different programming paradigms (Goldwasser and Letscher, 2008) and computer science courses (Agarwal and Agarwal, 2005).

2.3.4 Students' and instructors' use of educational tools

Several research projects have been dedicated towards the development of novice programming environments (Pears et al 2007). The main motivation behind these tools is to facilitate learning for students, in addition to supporting teachers in their roles (i.e. feedback, assessment and course management). Nevertheless, there are little to no signs of these tools being adopted by teachers and learners despite their wide availability. This could be attributed to the fact that most of these tools were designed to solve specific programming problems experienced by students in a university course (a notable example includes Greenfoot (Kölling and Michael 2010) designed to help students understand Object Oriented programming concepts). As such, it is no surprise that they are several tools developed by different universities that solve the same programming problem and with similar features. However, due to different universities' policies, the use of these tools might be exclusive. Another reason for the limited adaption is the absence of support or lack of contact point after the tool's development by researchers, as these tools are mostly prototypes or in beta release, making it difficult for teachers to customise these tools to meet the individual needs of their students.

Numerous studies have attempted to explain students' and teachers' use of these environments and their pedagogical effect (Knobelsdorf, Isohanni, & Tenenberg, 2012; Levy & Ben-Ari, 2007; Stern, Markham, & Hanewald, 2005). While the use of these environments is generally met with enthusiasm by teachers, they are reluctant about implementing them at all times for practical reasons. Setting up an environment for the first time is often tedious and complicated, and without the right technical support, it often leads to teachers abandoning these systems altogether because they feel that they have no control over them. Learning to use these environments is time consuming as well and requires hours of practice to benefit from the various interactive features they offer, in addition to looking for good examples, exercises and teaching resources that aligns with the environment's pedagogical approach, or suitable for a diverse cohort of learners. There are also a number of authors who dispute the educational value of these systems (Naps et al., 2003) or report mixed results (Gurka & Citrin, 1996; Hundhausen et al., 2002).

Students on the other hand are more keen on using programming environments and understand their significance on their learning achievement. However, they rarely use them independently outside of the lecture mainly because the majority of these systems are used as supplementary tools by teachers and therefore their use is often voluntary for students. Another reason that explains the inconsistent use of these tools is the reported difficulty of working with these environments as several of them were developed by experts and not user oriented. In an observational study investigating the use of novice pedagogical software carried out by Stern et al. (Stern et al., 2005), it was discovered that students create their own techniques for using software and these techniques inadvertently differ from the intended use by the developers. They also struggle with identifying the learning activities or interacting with advanced features without further assistance. This case is especially accurate when the instructional designs of the environment are ambiguous and interferes with their learning, or if the environment's teaching method is incompatible with their learning style. And while some students manage to use these tools successfully, they find it hard to transition to other IDEs afterwards. This results in learners losing interest in these tools altogether.

It is evident that there are a number advantages and disadvantages associated with the use of novice programming environments by both teachers and students. Consequently, it is important to anticipate any potential difficulties and issues that arise as a result of using these systems by learners, and consider additional costs on teachers in terms of installation, customisation and course management (McKimm, Jolie, & Cantillion, 2006). Although digital literacy might not be a problem for current university students, there should be a support system in place to assist them with the technical and educational aspects of the tool. Additionally, careful considerations must be taken regarding the design and features of the tool (i.e. visual and interactive elements) to make learning programming more engaging for students.

2.4 Summary

This chapter provide a brief description of the problem statement including literature relating to the difficulties that novice students' encounter whilst learning programming for the first time. Although several problems have been identified, both individual and language related, it is reasonable to conclude that the four main programming problems are teaching methodologies, learning styles, the choice of the first programming language taught and usability problems associated with e-learning tools.

These conclusions have a serious implication on the pedagogical design of programming courses and the usability issues of educational tools. In order to create a better learning environment, the teaching instrument must appeal to a diverse group of learners, supports several students' preferences, and is easy to use by both teachers and learners.

In order to teach several learners on the Felder-Silverman Learning Style spectrum (Active/Reflective, Sensing/Intuitive, Visual/Verbal, Sequential/Global), lecturers should consolidate the differences found in a group of learners. As such:

- Syntax free approach to appeal to (Active/Sensing /Sequential) learners.
- Examples and Exercises to be favourable by (Active/Verbal/Global) learners.
- Puzzles to attract (Reflective/Sequential) learners.
- Visualisations to cater to (Sensing/Visual) learners.

The presented methods were specifically chosen because they were proven to address several difficulties reported by new learners such as problem solving skills in addition to increasing performance and motivation in students.

The next chapter will explore a number of popular existing educational tools used at universities and their evaluation. The reported findings will be used to form the design guidelines which in turn were used to develop the interactive tool.

Chapter Three: Novices Programming Environments

3.1 Chapter Overview

This chapter examines the classification of programming environments for novice programmers along with examples of popular tools used to teach programming to first year students. This is followed by an evaluation of those environments based on functional and educational features. The findings were used to develop a set of guiding principles for the design the pedagogical tool central to this research, PILeT (Chapter 4).

3.2 Classification of Programming Environments for Novice Programmers

In recent years, a large number of programming tools and environments have been developed to make learning programming more accessible to a large audience of different ability levels. These tools often teach a comprehensive set of concepts, or offer debugging support and error detection for programming code. Another motivation for tools' development was to reduce the workload on teachers in the form of automated assessments and feedback, and content management and delivery (Ala-Mutka, 2005; Douce, Livingstone, & Orwell, 2005). These features are especially appreciated by students who value the importance of instant feedback and formative assessment as it allows them the freedom of experimenting with code and learning from trial and error.

There are over 50 programming environments dedicated to novice programmers, each with its own set of educational objectives and solutions. As such, several authors have attempted to organise these environments by suggesting classification taxonomies (Guzdial, 2003; Powers & Gross, 2005) with Kelleher and Pausch (Kelleher & Pausch, 2005) offering the most extensive classification for programming environments and languages for novice programmers (Figure 3.1).

According to Kelleher and Pausch, novices' environments are divided into two broad categories: systems that try to teach programming for the sake of programming, called teaching systems, and systems that assist the use of programming for the sake of another goal, called empowering systems. As this thesis is focusing on teaching and learning programming, empowering systems have not been explored further in this research, and thus all of its branches have been excluded from the diagram.

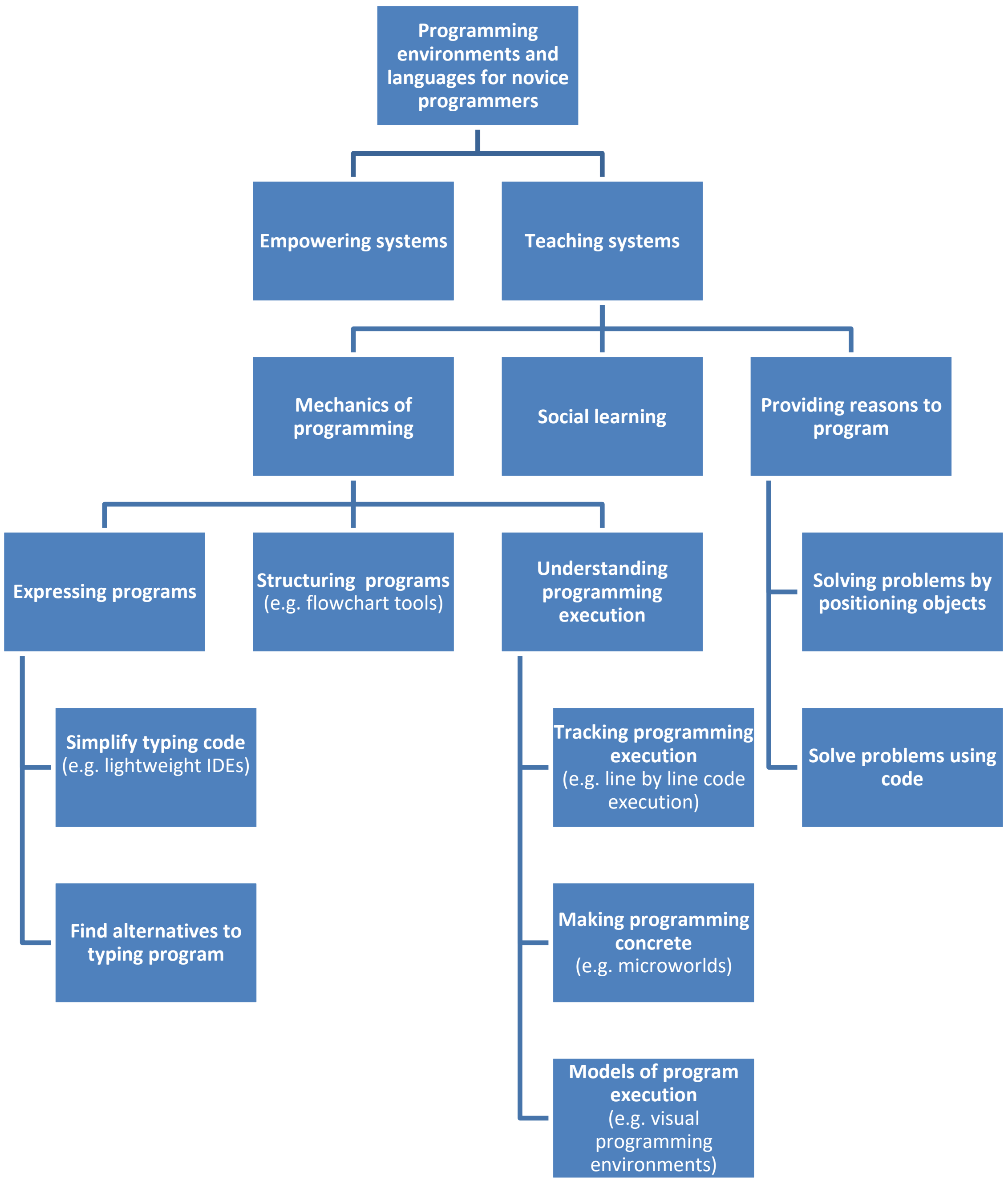

Programming environments and languages for novice programmers
Empowering systems
Teaching systems
Mechanics of programming
Social learning
Providing reasons to program
Expressing programs
Structuring programs (e.g. flowchart tools)
Understanding programming execution
Solving problems by positioning objects
Simplify typing code (e.g. lightweight IDEs)
Tracking programming execution (e.g. line by line code execution)
Solve problems using code
Find alternatives to typing program
Making programming concrete (e.g. microworlds)
Models of program execution (e.g. visual programming environments)

Figure 3.1 Classification System for Programming Environments And Language for Novice Programmers

Teaching systems are created to help students learn programming. Therefore, the majority of the programming environments in this group include simple programming tools that expose learners to fundamental programming concepts and the mechanics of programming. The other systems are designed to motivate student into programming either by allowing them to work together side by side or over a network, social learning, or by providing reasons to program for instance giving students specific tasks or a set of objectives as a starting point to solve a problem. Since the literature in Chapter 2 (Section 2.3) heavily implicated programming concepts and code cognition as learning obstacles, the emphasis will be on systems that aid with the mechanics of programming.

The first group in mechanics of programming, expressing programs, are systems that aim to make it easier for learners to express programming instructions in a specific programming language as with lightweight IDEs. The second group, structuring programs, focuses on the organisation and structure of the code (algorithm) instead of the syntax of the code as can be seen in flowchart tools. Finally the third group, understanding programming execution, aims to help the students understand what happens in the machine's memory at run time by visually tracking the execution of the program line by line and as a whole. Examples include microworlds, line by line code execution and visual programming environments. Below is a description of different environment types based on those classifications.

3.2.1 Understanding programming Execution

3.2.1.1 Line by line code execution

For students learning object oriented languages such as Java, it could be useful to experiment writing code in an interface that allows learners to assign values to variables, initialise objects and implement methods in the same way as a functional programming language for example Python. DrJava (Allen, Cartwright, & Stoler, 2002) is a lightweight pedagogic tool that enables students to run programs in an interactive incremental manner.

3.2.1.2 Microworlds

Microworlds are novice centred worlds that allows users to interact and manipulate the surrounding virtual environment by a set of programming code instructions. Microworlds usually employ storytelling techniques or game like adventures to facilitate learning programming. Examples of microworlds are Karel the Robot (Pattis & E., 1981) that teaches Karel programming language, and Greenfoot programming environment (Kölling & Michael, 2010) that teaches Java programming language.

3.2.1.3 Visual programming environments

There are various forms of visual programming tools: some use graphical representations of the programming code to allow learners to build algorithms by dragging and dropping elements of the code in the right order. These elements could be represented as pictorial icons such as RoboLab (Karoulis & Athanasis, 2006) or text code such as JPie (Goldman, 2003). Another group of visual programming tools either offers static visualisation of programming elements during execution, such as BlueJ (Kölling, Quig, Patterson, & Rosenberg, 2003), or dynamic visualisation, such as Jeliot3 (Moreno, Myller, Sutinen, & Ben-Ari, 2004), or both, such as jGRASP (James H, Hendrix, & Umphress, 2010). Each of these systems have their own strengths and weaknesses (Ragonis & Ben-Ari, 2005).

3.2.2 Structuring Programs

3.2.2.1 Flowchart tools

Flowchart tools enable learners to construct small parts of code graphically by using templates and manipulating flowcharts. The objective of these tools is to teach learners the flow and direction of functions rather than fixating on the syntax of the programming language. These tools also allow student to change the connection and direction between different program elements. Example of these tools includes RAPTOR (Giordano & Carlisle, 2006).

3.2.3 Expressing Programs

3.2.3.3 Lightweight integrated development environments

Expert programmers use sophisticated software development kits (SDKs) and integrated development environments (IDEs) to compile and execute programs. Some of these IDE's, such as NetBeans, are used to teach introductory programming courses, however, these tools were proven to be difficult to use by novices as they require complicated operations. Lightweight IDEs are simple, easy to use programming editors used in educational environments with helpful features such syntax highlighting and autocomplete. An example of a lightweight IDE is IDLE ("25.5. IDLE — Python 3.6.0 documentation," 2017).

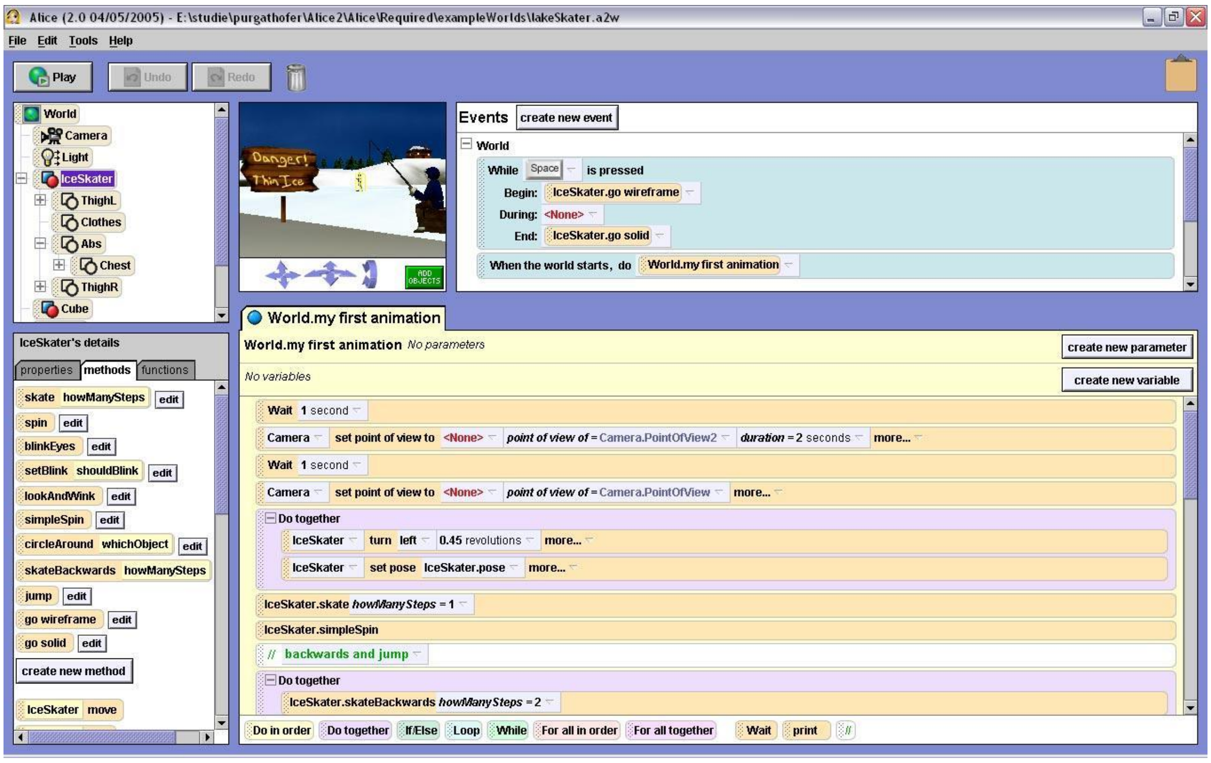
3.3 Examples Programming Environments for Novices

There are several options of educational technologies to choose from. Though many share similar features and characteristics, each of one these tools offers its own distinct support type (Powers & Gross, 2005). This section presents a limited selection of some of the widely recognised pedagogical environments that have been heavily investigated by the computer science education community. Alice (Cliburn, 2008), BlueJ (Kölling et al., 2003), Jeliot3 (Moreno et al., 2004), RoboLab (Karoulis & Athanasis, 2006) and RAPTOR (Giordano & Carlisle, 2006) are educational environments that have been used to teach introductory programming at university level. These tools were specifically selected to exemplify the group of environments specified in section 2.2. [2]

3.3.1 Alice

Alice (Figure 3.2) is a popular microworld system that enables learners to build programs from scratch by manipulating objects in a 3D environment. Programming in Alice is implemented by dragging and dropping graphical blocks labelled with objects, methods and other command lines into a main window (called a main world). Once these blocks are arranged and dropped in the right order, a textual representation of those blocks unveils. Learners can also change the execution flow of programs by using control structures such as for and while loops. As the code in each block is fixed, i.e. uneditable, Alice encourages students to build different programs using essential concepts taught in introductory programming courses without worrying about producing syntactical errors.

[2] It should be noted that none of these environments teach Python. An existing tool, How to Think Like A Computer Scientist: Interactive Edition, has been reported to be used by several learners individually but never at an institutional level.



```
World
  While Space is pressed
    Begin: IceSkater.go wireframe
    During: <None>
    End: IceSkater.go solid
  When the world starts, do World.my first animation

World.my first animation  No parameters
No variables
  Wait 1 second
  Camera set point of view to <None> point of view of = Camera.PointOfView2 duration = 2 seconds more...
  Wait 1 second
  Camera set point of view to <None> point of view of = Camera.PointOfView more...
  Do together
    IceSkater turn left 0.45 revolutions more...
    IceSkater set pose IceSkater.pose more...
  IceSkater.skate howManySteps = 1
  IceSkater.simpleSpin
  // backwards and jump
  Do together
    IceSkater.skateBackwards howManySteps = 2
```

Figure 3.2 Alice

3.3.2 BlueJ

BlueJ (Figure 3.3) is a Java IDE specially developed to teach an object oriented language to students. The environment allows learners to write, edit, compile, test and debug their code. BlueJ graphically represents the relationship between several objects and classes in the form of a UML (Unified Modelling Language) diagram. This feature allows the students to understand the class hierarchy of objects (i.e. superclass, subclass) and other objected oriented concepts such as inheritance. Users can alter the state of classes by typing in the syntax in the editor, or clicking on the objects.

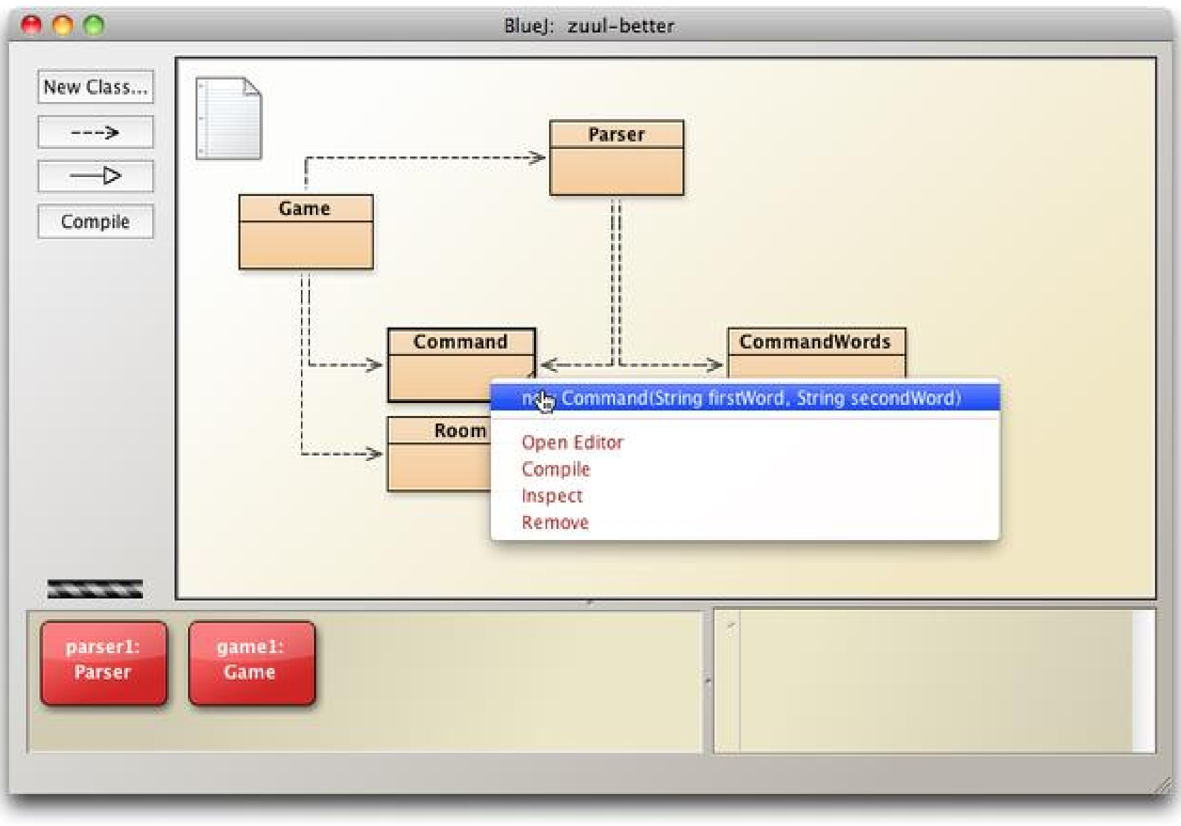
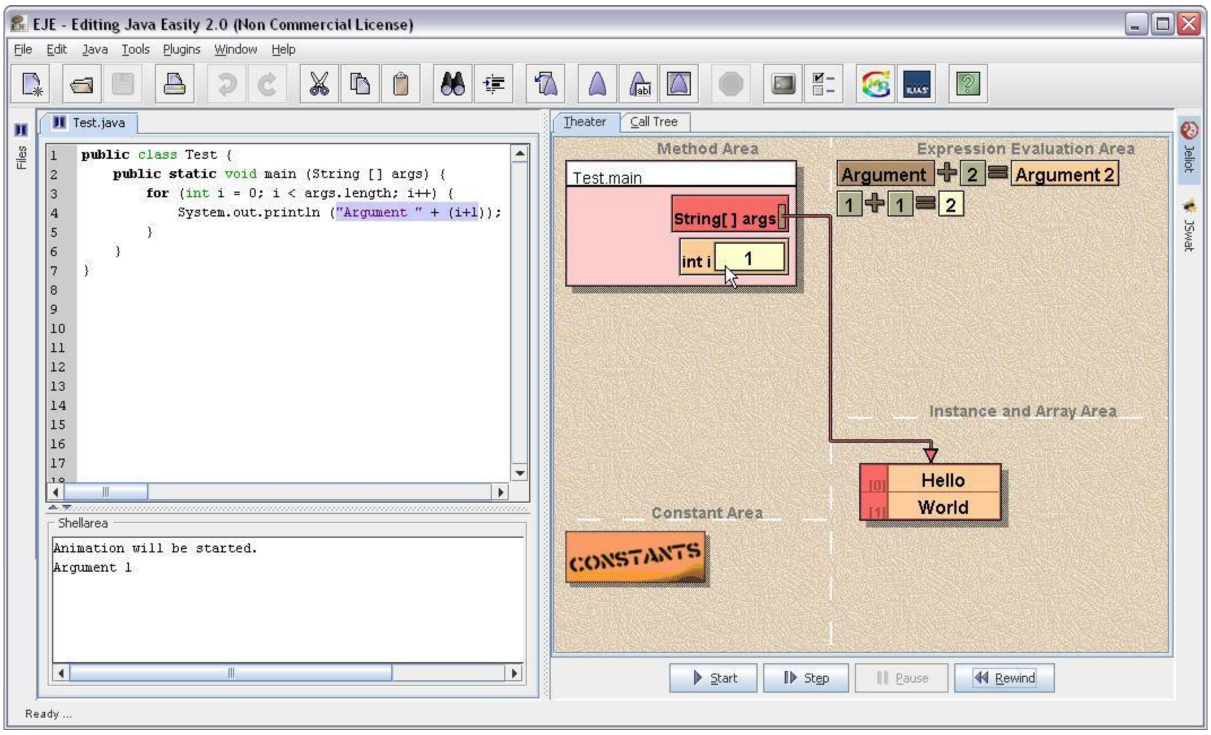


Figure 3.3 BlueJ

3.3.3 Jeliot3

Jeliot3 (Figure 3.4) is a program visualisation tool developed to help beginner students learn object oriented programming and procedural programming. The user interface contains a code editor and a visualising space that incrementally animates each line of source code during execution. If there are any compilation errors, the program highlights the line where the error occurred and asks the user for the correct input. This feature encourages learners to experiment with code and learn from their mistakes.



```
public class Test {
    public static void main (String [] args) {
        for (int i = 0; i < args.length; i++) {
            System.out.println ("Argument " + (i+1));
        }
    }
}
```

```
Animation will be started.
Argument 1
```

Figure 3.4 Jeliot3

3.3.4 Robolab

Robolab (Figure 3.5) is a LEGO programming environment that uses assembly language to program and control robots. In order for the robot to operate, learners first create the program by assembling programmable bricks and icons that represent functions and instructions in the interface then download the program to the robot via an infrared transmission. This environment teaches programming by allowing students to construct their knowledge while modifying and testing their robots and thus motivating them into learning.

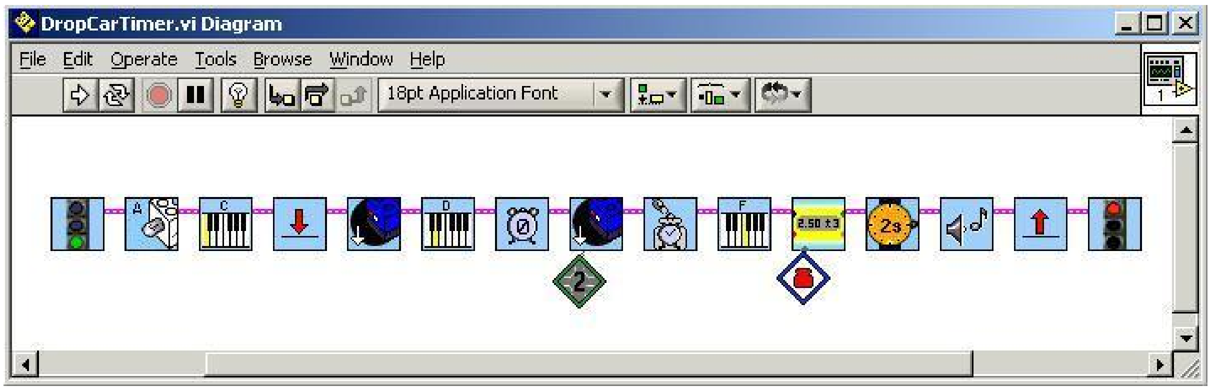
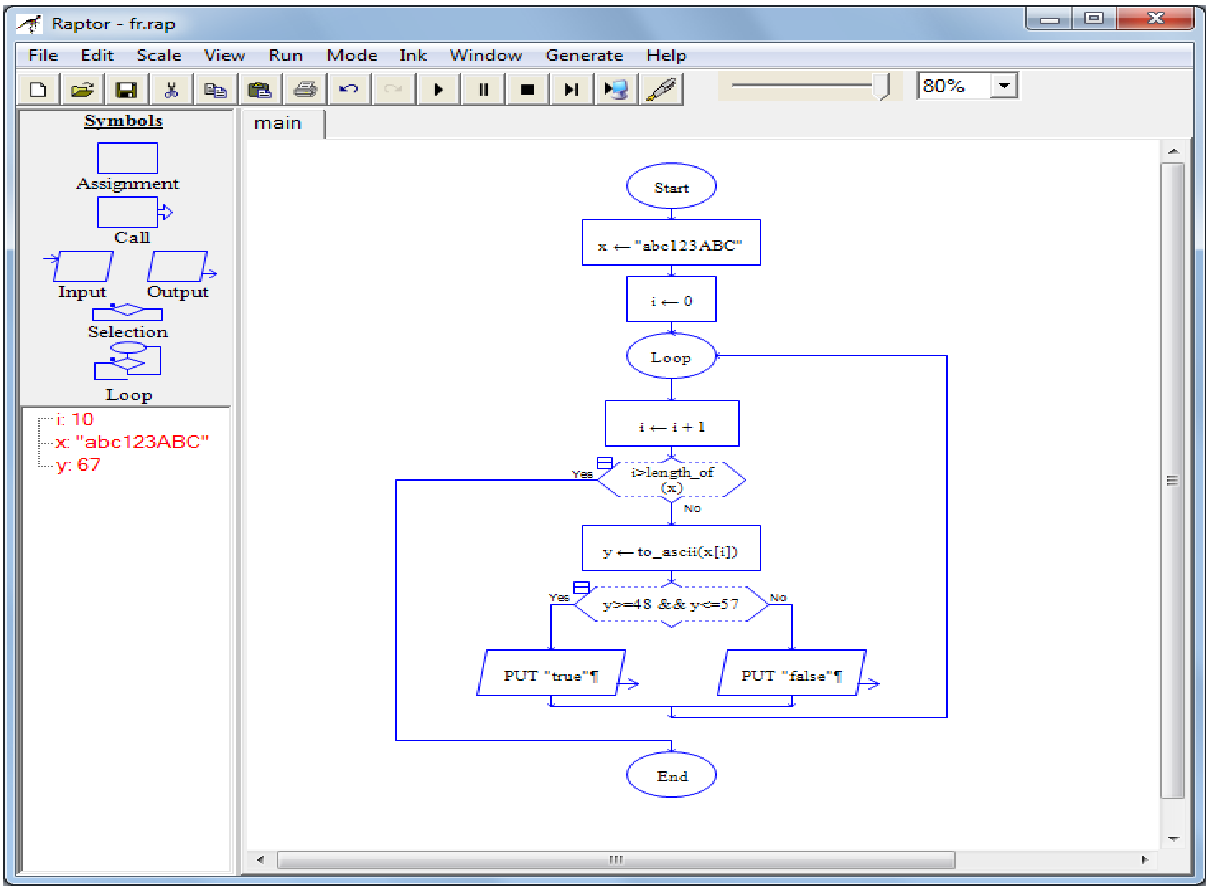


Figure 3.5 RoboLab

3.3.5 RAPTOR

RAPTOR (Figure 3.6) is a learning environment that was developed to teach students algorithm development by drawing flowchart diagrams. Each stage of the program is built by selecting different flowchart symbols from the interface representing variable assignment, user input, program output, loops, procedure calls and selections. Additionally, students are allowed to manually add more information to symbols which are automatically checked for syntactical correctness to prevent any errors. The programming elements of RAPTOR are similar to C and Pascal languages.



i: 10
x: "abc123ABC"
y: 67

Figure 3.6 RAPTOR

3.4 Evaluation of Novice Environments

In order to evaluate programing environments for novices, a rubric was designed with a set of standardised criteria to measure two dimensions: the functional features of each tool and the educational impact on novice learner (Appendix 2). The functional features evaluation rubric was distilled from the works of Kelleher and Pausch (Kelleher & Pausch, 2005) and Gross and Powers (Gross & Powers, 2005).

3.4.1 Functional Features Evaluation

The table in Appendix 2 presents a comparison of the different functional features of each environment from the following aspects: programming style, programming concepts taught, code representation in the tool, code construction, additional support to understand programming concepts, and syntax error handling.

It can be seen from the table that developers of those tools have tried to incorporate different approaches to teach programming concepts. They have also included several features to make programming easier and more accessible to students. And while some novice environments succeeded at simplifying programming ideas by providing alternatives to typing in code, like Alice and RAPTOR for example, they can hardly compete against tools that actually teach the mechanics of writing programs, like BlueJ and Jeliot3. However, these tools reportedly have a steep learning curve, and make it difficult for students to transition into using IDEs.

Therefore, in order to support students in their learning, it is necessary to create an environment that focuses both on the syntactical aspects of the programming language and problem solution. This could be accomplished by developing an integrated environment with a built in IDE for code formulation and a space for algorithm visualisation. Additionally, the tool should reinforce the taught programming concepts by offering supplementary material along with examples, exercises and error feedback to address the gap in their knowledge and improve their programming skills.

3.4.2 Educational Impact Evaluation

Several strategies were used to evaluate the educational impact of novice programming environments. These techniques were adopted from the works of Hundhause et al. (Hundhausen, Douglas, & Stasko, 2002) in which they used anecdotal, analytical and empirical evidence to assess students' learning outcomes after using these tools for a fixed period. Both quantitative and qualitative data were collected, ranging from overall grades to questionnaires, to support the evaluation. A summary of the results is listed in Table 3.1.

Programming environment	**Educational impact**
Alice	Using Alice increases the performance and retention of struggling novices whilst having little or no impact on other groups. Nevertheless, it was reported to increase the level of confidence and attitude towards programming on all students (Bishop-Clark et al 2007, Sykes 2007).
BlueJ	A link was established between using BlueJ and creating programming misconceptions in students. Though some of these misunderstanding may be alleviated with the right intervention methods, they could lead to new problem in program comprehension (Chudowsky et al 2001).
Jeliot3	Jeliot3 must be used for a prolonged period by students in order to benefit from the different functions of the tool. Additionally, only struggling students fully benefited from using the tool animation feature (Maravić Čisar et al 2011).
Robolab	Leaners reported difficulties using Robolab as some aspects of the environment are very difficult to grasp. It also did not have a positive impact on their learning because it takes a very long time to complete a programming exercise (Fagin 2002).
RAPTOR	RAPTOR enhances students' comprehension of learning algorithms whilst it fails to teach the students the syntax of programming (Carlisle et al 2005).

Table 3.1 Comparison of Programming Environments for Novices (Educational Impact)

While it is evident that some learners benefit from novice programming environments, those benefit are limited to a small set of students for various reasons mostly relating to functionality and usability. Therefore it is important to investigate students' use of pedagogical tools and teachers' utilisation of these systems to teach programming. It is also worth exploring whether a relationship exists between the system design and learning outcomes.

3.7 Summary

With the prevalence of programming environment for novice learners, several teachers are in favour of using them in introductory programming courses at university. While some of these tools have been proven effective in improving student' retention and engagement, many have failed to address the difficulties that students experience whilst learning. A systematic evaluation of these tools concluded that most of these problems were attributed to design and usability issues as opposed to the quality of the teaching resources or examples. The As a result, a set of design principles for educational programming environments were developed in the next chapter to help with the design of PILeT.

The next chapter includes a description of PILeT in addition to documentation of the system architecture and features.

Chapter Four: Design Principles of Programming Environments

4.1 Chapter Overview

This chapter presents the design guidelines for building educational environments. This is followed by the process of developing PILeT which includes a layout of the system architecture and a description of the features. The chapter concludes with an outline of how it complies with design guidelines and learners on the Felder-Silverman Spectrum (Chapter 2).

4.2 Guiding Principles for Educational Programming Environments

Despite the growing popularity of programming environments for novices, there does not exist a set of standardised design guidelines or guiding principles to help with the development of these systems. Therefore it is no surprise that while some of these educational environments were found to be useful in assisting students with learning programming, many fell short of fulfilling basic learning objectives due to design or usability issues. These environments also ignore the individual differences and needs existing in large student groups such as comprehension skills, previous knowledge and learning styles (Ford & Chen, 2001). To this end, a number of guiding principles were created to fill the gap in existing systems, and inform the design of the programming tool central to this research (PILeT). These principles were distilled using thematic analysis (Braun and Clarke 2006) of existing literature on designing learning environments, combined with the evaluation of current programming tools. Next those principles were reviewed by two design experts for feedback and evaluation. As a result of this process, 11 design principles were developed.

1. Ease of use:
ISO 9241 (Quesenbery, 2001) defines usability as "The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use". In order for a product to be deemed usable, it must meet the five characteristics: effective, efficient, engaging, error tolerant and easy to learn. In the context of programming environments, the tool should be easy to use by novice learners, and allows them to achieve a specified goal without getting stuck.

2. Discoverability:
The tool should be designed to allow the students to locate the needed information and functions to complete a specified task in a short time. This could be achieved by considering the interface elements in terms of text size, colour, information layout, order and flow in addition to consistency.

3. Interactive functionalities:
Studies on teaching programming found that active learning techniques have been proven more effective than passive learning activities (Walker, 2004; Wulf, 2005). Therefore the tool should allow learners to interact with the learning material and exercises in order to increase engagement with the content and increase motivation in learning. This could be achieved by using attractive animation and graphics for example. However, those features should be designed carefully as to not distract the students from meeting the learning outcomes.

4. Customisation:
The tool should support diverse learners with specific learning characteristics. Using hypermedia in learning environments facilitates adaptive instructions and allows the use of resources to match the individual needs of each of them (Carver, Howard, & Lane, 1999). This could be achieved by offering

customised teaching strategies and exercises to appeal to many students (i.e. using videos and visual diagrams for visual learners, puzzles for active learners).

5. Error handling:
Generally, novice learners have a difficult time dealing with runtime errors that have complicated messages. This leads to students' frustration with programming (Marceau, Fisler, & Krishnamurthi, 2011; Murphy et al., 2008). Therefore the tool should offer a helpful method of handling errors in students' code. This encourages students to experiment with different syntax of the programming language and incrementally build larger programs.

6. Automated assessment and feedback:
Automated assessments and feedback offer a quick and useful way for student to check the correctness of their solution, gauge their understanding of the taught concepts and fill any gaps in their knowledge. The tool should offer formative assessment to improve students' comprehension and support them learning programming.

7. Visualisation:
In order to alleviate the complexity of programming, the tool should provide visual representation of code execution at run time and illustrate programming concepts (for example function calls) in order to help the students comprehend programming constructs and structure.

8. Improving problem solving techniques:
The tool should place more emphasis on the problem domain and less focus on the programming syntax. This could be achieved by teaching the students strategies for algorithm formulation and different solution techniques in a stepwise manner.

9. Minimise cognitive overload:
There are several recommendations in the computer science education literature for reducing cognitive overload for students. For example, using self-paced instructions for teaching, reducing the amount of text for reading or replacing words with images, incorporating relevant examples to demonstrate programming concepts and providing hints for solutions to name a few.

10. Cover core programming concepts taught in introductory programming course:
This includes threshold concepts (Boustedt et al., 2007; McCartney & Sanders, 2005; Zander et al., 2008) which are considered to be "troublesome" by some students. Although introductory courses are different at each university, for the purpose of this research, the essential concepts were identified by:

- Consulting the module leaders of introductory programming courses at the university this research is taking place.
- Reviewing the ACM curricula recommendation for computer science (The Joint Task Force on Computing Curricula, 2013).
- Literature review on difficult programming concept (Chapter 2).
- Student feedback. A 5 point Likert scale questionnaire was distributed to first year student at the end of the course asking them to rate each concept based on the level of difficulty. This method has been validated and used by (Carlson, Chandler, & Sweller, 2003) and (Marcus, Cooper, & Sweller, 1996).

Based on these resources, the core concepts in introductory programming courses are: variables (type and assignment), selection statements (if statement), repetition (for and while loop), functions (definition, structure, return value and call) and lists.

11. Dependency:
The tool should allow the student to write custom programs in the same way as they are created in IDEs. Currently most of the novice environments allow the student to construct code by either dragging and dropping code segments or blocks, drawing diagrams, or manipulating input values in existing code. To reduce the risk of dependency, the tool should behave in the same way as programming IDEs and present the student with standard compiler messages at execution time.

Table 4.1 presents a mapping of the identified design requirements against existing pedagogical tools (Chapter 3). It can be inferred from the table that none of the novice programming environments meet all of the eleven guiding principles. Whilst none of the tools support customisation and automated assessment and feedback (requirements 4 and 6), Jeliot3 and RAPTOR appears to meet most of the requirements followed by BlueJ. However, other than BlueJ, they do not cover all of the essential programming concepts taught in introductory programing courses. Therefore, there presents a need for an educational tool that conforms to all of the guidelines.

Programming environment	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Alice	X	x	x								
BlueJ					x		x		x	x	x
Jeliot3			x		x		x	x	x		x
Robolab			x		x						
RAPTOR	X	x	x		x			x	x		

Table 4.1 Design Principles Met By Existing Programming Environments

4.3 PILeT

PILeT (Python Interactive Learning Tool) is a web based application that combines multiple teaching methods to accommodate various learning needs found in a diverse group of students As such, each programming concept is explained using videos, reading material, examples, exercises and puzzles adequately on their own, or in combination with other approaches. This way, student can learn from the teaching technique they are most comfortable with, or use a mixture of several approaches to support their learning (Pollock and Harvey 2011b). The selected methods were carefully chosen as they have been proven to improve comprehension of programming concepts and increase motivation in student (Chapter 2).

4.3.1 System Architecture

PILeT was built using Runestone Interactive Tools (Runestone 2012). Runestone is a free open source authoring tool for creating interactive computer science textbooks online such as Fundamentals of Web Programming (Miller 2015) and CS Principles: Big Ideas in Programming (Guzidial 2016). The advantage of using Runestone over writing a web based application in HTML from scratch is that the user can use and customise pre-existing directives, which are explicit blocks of markup language, for building different components of the application. This method enables the creation of the interactive textbooks in a short period. The user can also build their own directives to create new features for their website.

PILeT architecture consists of two main layers: the content development layer and PILeT server (Figure 4.1). The content development layer is responsible for creating all of the learning material content including text, images, videos, code visualisation, tracing, exercises and external links. The output from this layer is then passed to PILeT server.

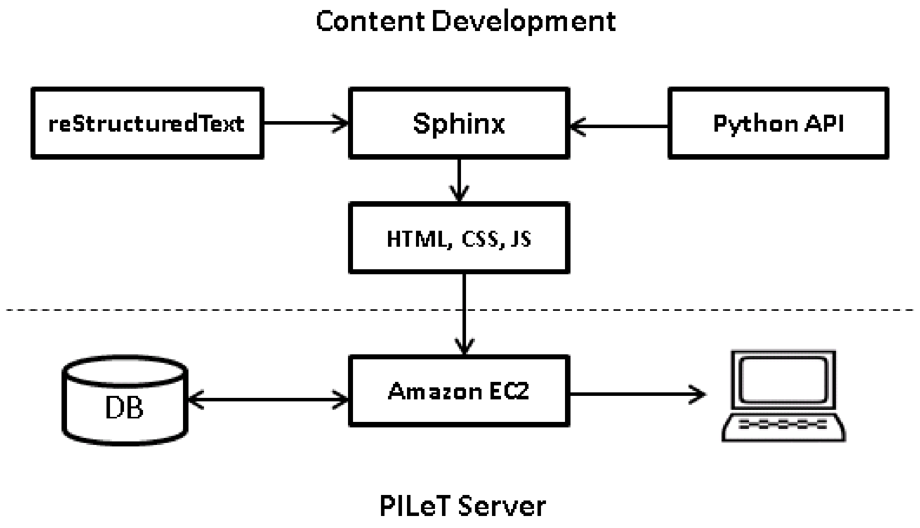


Figure 4.1 PILeT Architecture

PILeT is based on Sphinx (Brandl 2008) which originally was designed as an automatic tool for generating documentation for Python source code. Currently it is used in various applications for instance documenting software projects in different programming languages and creating e-books. Sphinx works by converting reStructuredText, its lightweight markup language (Goodger 2002), into different formats such as .epub and HTML websites. Sphinx uses Python docutils package to process the documentation.

The PILET server is an Amazon EC2 instance (Amazon Elastic Compute Cloud virtual server). The server's main responsibility is to deliver the content to the student via the web browser. Other responsibilities include:

- Data storage and collection for PILeT content.
- Storing log in details for students.
- Recording activities generated by the students.
- Saving assignment marks.
- The server also includes an interface for the course leader to grade the solutions submitted by the students and leave feedback.

In order to write textual paragraphs in reStructuredText files, normal text is used which could be separated by spaces, punctuations and line breaks for new paragraphs just as writing a document. Special inline markup is used to format the text such as *example text* to produce *example text* and **example text** to produce **example text**. However to add videos, images and other interactive features into PILeT, several directives were used to create the final look and feel of PILeT.

Directives are usually written in blocks of code and embedded within a reStructuredText document. Directives include special commands that enable Sphinx to convert the code in the block into a specified output. Each directive include a name, arguments (required and optional), and content. The directives used in PILeT are described below and include YouTube, activecode, codelens, Parson's programming puzzles, and multiple choice questions.

4.3.1.1 YouTube Directive

The YouTube directive is used to embed videos into PILeT using reStructuredText. The videos are uploaded on YouTube first, and then the video id is used to locate and stream the video on the page. The name of the directive is (youtube) followed by a unique identifier which is the video id on YouTube. height, width and align are required arguments to position the video within the page.

In order to execute the directive in Sphinx, a class is created in Python which inherits from docutils.parsers.rst.Directive (Elza et al 2012). Sphinx compiles the reStructuredText into HTML. Finally, the YouTube video is embedded on PILeT as a thumbnail image that plays once clicked (Figure 4.2).

4.3.1.2 Activecode Directive

Activecode directive is used to allow the execution of Python code within the PILeT webpage instead of using an external IDE such as IDLE. The activecode box could be left blank for users to construct their own code or it could include a worked example that could be edited and compiled again.

The name of the directive is (activecode) followed by a unique identifier which links the submitted code to the grading page in PILeT. The next line contains a sample code which could be modified and executed. Sphinx compiles the reStructuredText into HTML (the generated code is too long to be included). Figure 4.3 shows an implementation of activecode in PILeT.

The activecode directive depends on Skulpt (Graham 2010) and CodeMirror (Haverbeke 2011), both which are open source projects. Skulpt is a built-in implementation of Python code in the browser window. When the code is executed, Skulpt renders the source code into JavaScript then runs it in the JavaScript virtual machine on PILeT page. The advantage of this method is that it allows the execution of code in PILeT offline.

CodeMirror is a rich and customisable text editor which uses JavaScript in its implementation. It is used for modifying code in the browser and contains several useful features such as code auto completion, syntax highlighting and auto indentation. The last two features help new users to get familiar with Python code structure and syntax.

4.3.1.3 Codelens Directive

The codelens directive is used to visualise a step by step execution of the Python code. It also displays the different values of variables throughout the various stages of the program.

The name of the directive is (codelens) followed by a unique identifier to differentiate it from other codelens directives. Showoutput is an optional argument to show the output from a print statement on the webpage. The lines that follow contain the Python code that will be visually traced. The difference between activecode directive and codelens is that the source code in codelens cannot be modified and must be syntactically correct for it to be executed.

The codelens directive is based on Philip Guo Python Tutor (Guo 2013). It works by running the source code through a Python interpreter. Next a series of stack frames are produced which match with each line of the code as it is stepped through. The stack is then converted into a JavaScript Object Notation format which is saved in a PILeT page and then delivered to codelens upon execution (Figure 4.4).

4.3.1.4 Parson's Programming Puzzles Directive

Parson's programming puzzles (Parsons and Haden 2006) is an interactive tool that allows users to drag and drop fragments of code in the right order into the solution area.

The name of the directive is (parsonprob) followed by a unique identifier to differentiate it from other Parson directives. The following line is a question text which is separated from the rest of the source code with single dashes. The equal sign is used to encapsulate each line of code in a block. Once it is implemented, the code blocks are randomly mixed in order.

Parson's programming puzzles directives is created using Hot Potatoes authoring tool (Potatoes 2001). The tool is used to create various online exercise including drag and drop problems in JavaScript. Figure 4.5 shows an implementation of Parson's Programming Puzzles in PILeT.

4.3.1.5 Multiple Choice Questions Directive

The multiple choice questions directive is used to create questions either with one correct answer (radio buttons) or multiple answers (checkboxes). It also allows the addition of feedback with both wrong and right answers.

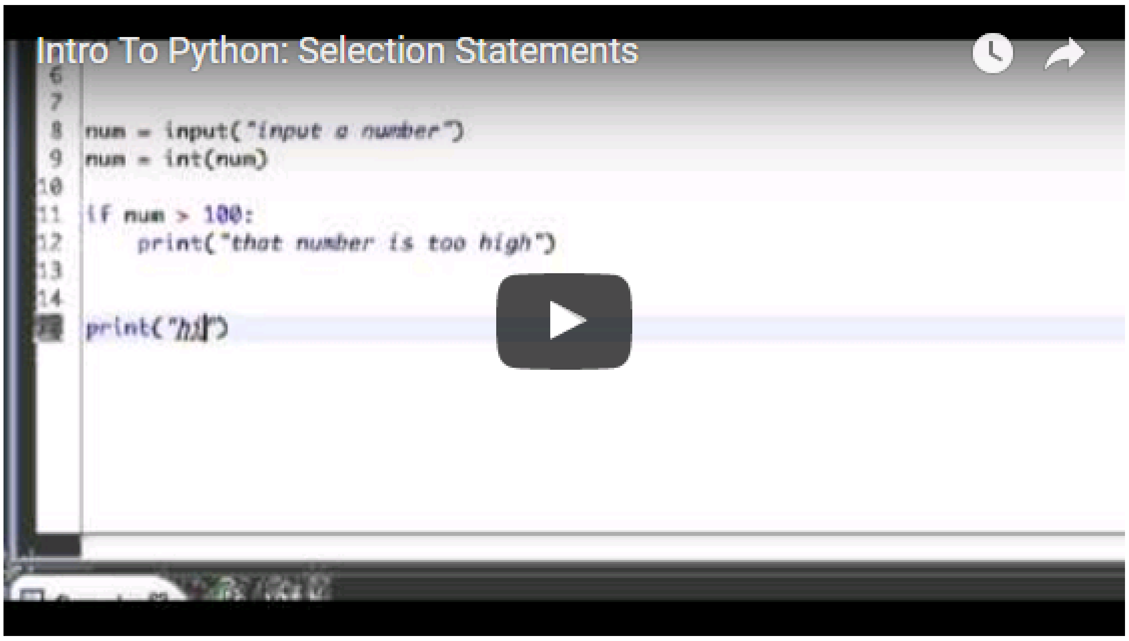
The name of the directive is (mchoicemf) followed by a unique identifier which links the submitted answer to a grading page in PILeT. To create checkboxes (mchoicema) is used. answer_a and correct are required arguments where the value after correct is listed as one of the answers. feedback_a, feedback_b, feedback_c and feedback_d are optional arguments which correspond to each selected answer. The line that follows the question text. In case there is more than one correct answer (checkboxes) the values after correct are separated by commas. Figure 4.6 shows an implementation of multiple choice questions in PILeT.

4.3.2 PILeT Features

A number of interactive features were developed and embedded in PILeT. These features are in line with the created design principles and appeal to learners on the Felder-Silverman spectrum (Chapter 2).

4.3.2.1 Embedded Videos

Whilst it is difficult to find good substitutes to traditional textbooks, many leaners will rely on other modes of learning than read textual material. This could be attributed to many reasons such as comprehension problems or reading difficulties (Paul Carter and Paul 2009, David J Malan 2007). PILeT embedded videos provide a suitable alternative to explain Python concepts, gradually work through programming examples, or even support textual material. And while videos might be a favourable option even for textual learners for their ease of use, it is unfeasible to convert an entire course book content into video, additionally, the attention span for videos is rapidly shrinking (Koohang and Harman 2007).



```
num = input("input a number")
num = int(num)

if num > 100:
    print("that number is too high")


print("hi")
```

Figure4. 2 YouTube Video in PILeT

Videos in PILeT are short, ranging from seven to 10 minutes. The videos are embedded at the beginning of each chapter and cover fundamental principles corresponding with the programming concept taught in that chapter (Figure 4.2). Sometimes, there are additional videos for other examples and exercises. The videos are similar to tutorials; they start with an introduction by the instructors who explains the objective of the lesson followed by a live demonstration of the code in a Python IDE. Learners have the option to pause the videos and attempt to do the exercises on their own before the final answer is revealed, or type in the source code line by line along with the video demonstration. Overall, video instructions are beneficial to different learners as it allows them to progress at the pace that is most appropriate to them.

4.3.2.2 Embedded Code Compiler

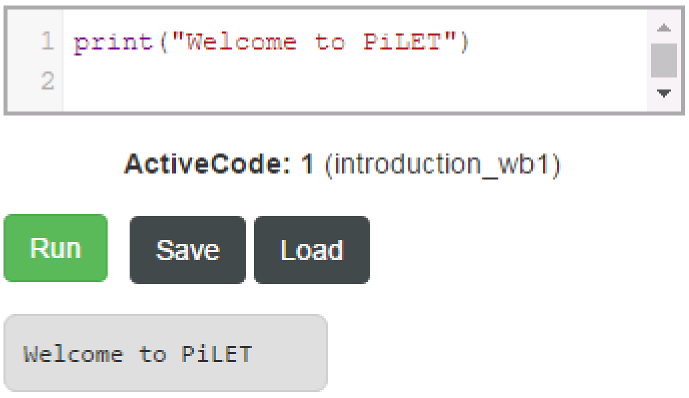
Programming is one of the computer science disciplines in which practice is imperative to a learners' success. Novice students are constantly encouraged to experiment with their code and learn from their mistakes. However, many students lose interest in the subject when they encounter programming problems or runtime errors.

PILeT contains an embedded code compiler called activecode. It enables students to execute worked examples, modify them, or try out their own code on the web browser without switching between the webpage and an external IDE. Another advantage of activecode over other IDEs is that it presents the learner with user friendly error messages along with a description of the error and suggestions for fixing it.

Activecode consists of the following (Figure 4.3):

- Code editor with line numbers: the area containing the code.
- Execution window: displays the result of the execution.
- Run button: execute the code in the editor.

- Save button: once the student is logged into PILeT, they can save any changes they made to their code. Additionally, a copy of the source code is submitted to the instructor for grading and feedback.
- Load button: to recall the saved code.



```
print("Welcome to PiLET")
```

ActiveCode: 1 (introduction_wb1)

```
Welcome to PiLET
```

Figure 4. 3 Activecode in PILeT

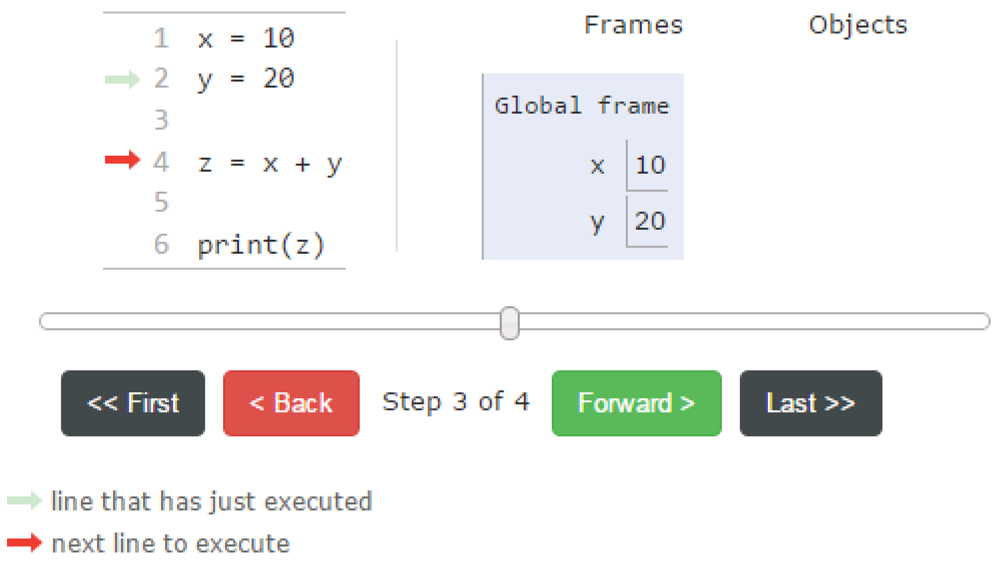
4.3.2.3. Embedded Visualisation

Code visualisation is a method to graphically represent the interaction between different elements of the program, for example variables and functions, in the memory while the program is running. It also shows a step by step execution of the source code. This method allows the learners to build metal models of their programs and understand the logic behind the execution process. Other benefits include reading and tracing complex code, error debugging, and understanding the flow (selection statements) and iterative development (loops) of programs (Naps et al 2002).

Codelens (Guo 2013) is an embedded interactive code visualizer that animates the execution of Python code line by line. It also tracks the different values of variables as they change during the program run. The student have full control over the speed of the code execution; the can move forward one step, backward or jump to the last line of the program. Teaching Python using codelens enhances comprehension of program algorithms and data structures.

Codelens consists of the following (Figure 4.4):

- Code window: the area containing the code along with two coloured arrows. The green arrow points at the line that has just been executed and the red arrow at the line to be executed.
- Result window: displays a graphical animation of variable values and interactions between them and other program elements.
- First button: jump to the first line of the source code.
- Back button: move backwards one line.
- Forward button: move forward one line.
- Last button: jump to the last line of the source code.



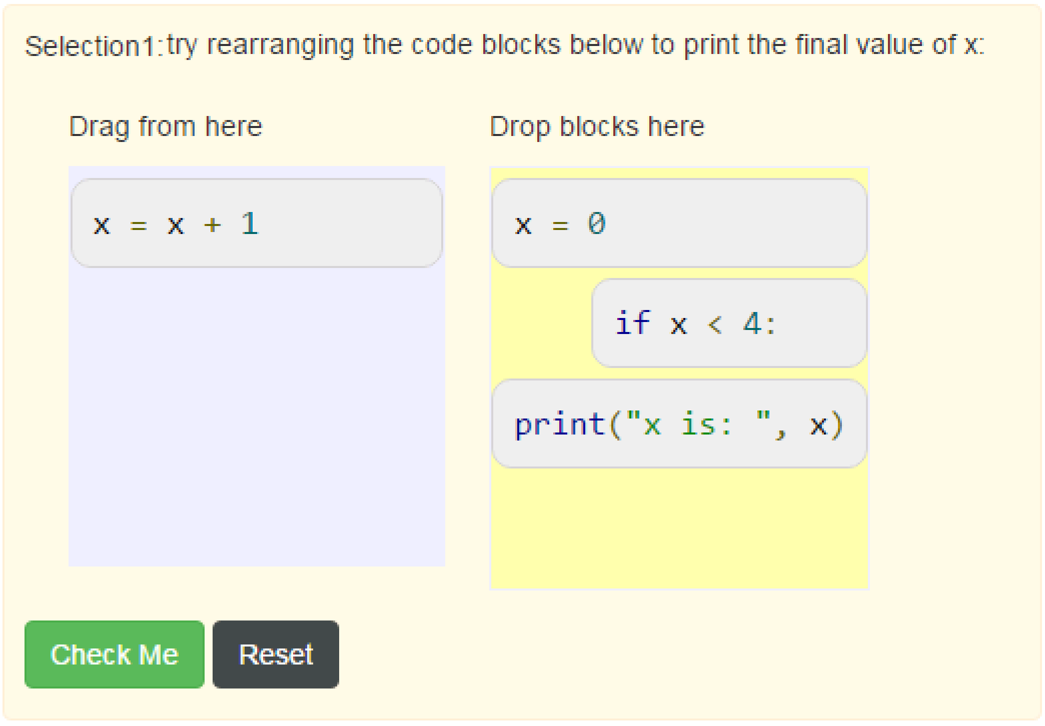
```
x = 10
y = 20

z = x + y

print(z)
```

Figure 4.4 Codelens in PILeT

4.3.2.4 Parson's Programming Puzzles

Parson's programming Puzzles (Parsons and Haden 2006) are simple interactive exercises that require students to build working programs by rearranging random code blocks into the right order via drag and drop. This kind of exercise encourages students to actively interact with the source code without the fear of producing syntactical errors as no typing is taking place. Using Parson's puzzles to teach programming minimises the cognitive load on learners and increases motivation. The puzzles have different difficulty levels which could be increased by adding distractors or extra unrequired code blocks. Not only does the student have to identify them, but they also must adhere to Python indentation rules when dragging the right blocks into place.

The left section of Parson's Programming Puzzles contains code blocks that must be dragged in the right order to the right section (Figure 4.5). Once the code assembly is completed, the student can check the correctness of their solution by pressing the check me button. If they want to start again they press the reset button.


Selection1:try rearranging the code blocks below to print the final value of x:

Drag from here

```
x = x + 1
```

Drop blocks here

```
x = 0
    if x < 4:
print("x is: ", x)
```

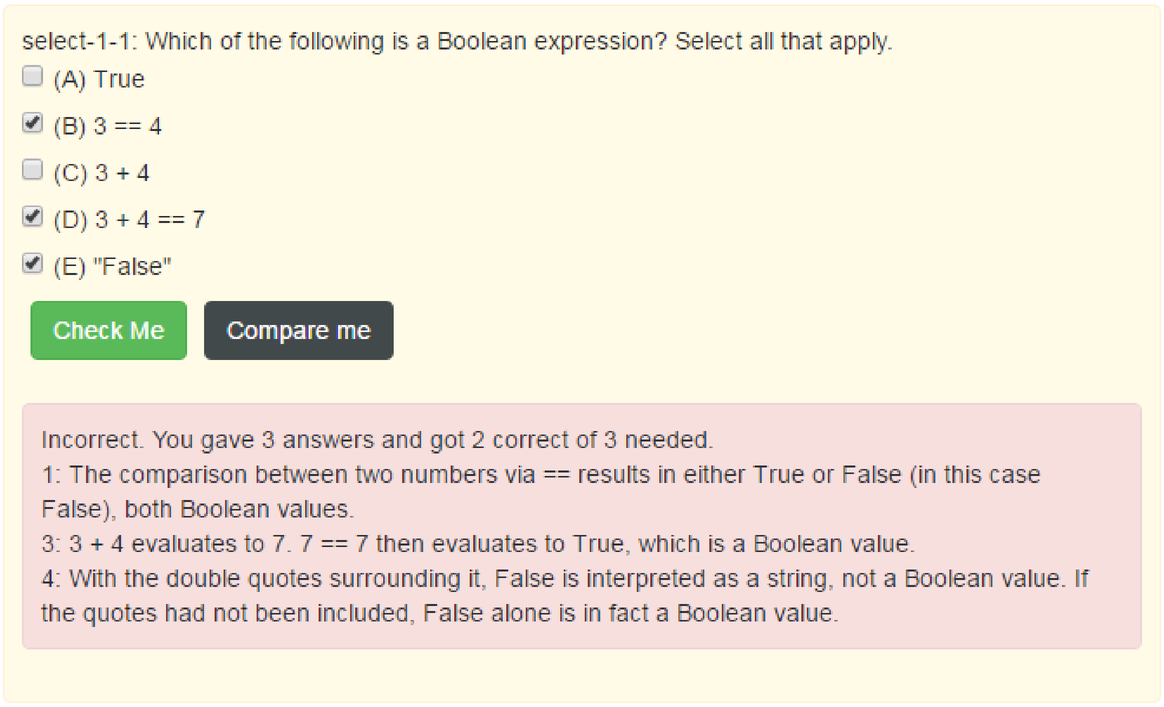
Figure 4.5 Parson's Programming Puzzles in PILeT

4.3.2.5 Automated Assessment and Feedback

PILeT contains several embedded exercises in the form of multiple choice questions and checkboxes. These questions are used to examine learners' comprehension of learnt concepts as a whole and as means to supplement the teaching material. For example, for students learning about selection statements in Python, the multiple choice questions are mostly focussed on the syntax, or keywords, that are used to construct the if statement, along with the direction of the flow of execution. For some students, learning through examples is considered a better alternative to learning passively through textual explanations as they provide a quick hands on way of gaining knowledge and retain learnt content.

Automated assessments in PILeT are also used to identify and correct misconceptions and previous ideas that some learners may have about certain programming constructs and provide instant feedback (Kaczmarczyk et al 2010b). This is achieved by checking the learners' submissions against classical misconceived answers and correcting it by delivering a clarified, detailed response (Hristova et al 2003).

Figure 4.6 shows an implementation of an automated assessment in PILeT. The student has selected (E) “False” as an answer which is different from (False). The detailed feedback explains why the solution is incorrect along with the right answer.


select-1-1: Which of the following is a Boolean expression? Select all that apply.

- [ ] (A) True
- [✓] (B) 3 == 4
- [ ] (C) 3 + 4
- [✓] (D) 3 + 4 == 7
- [✓] (E) "False"

Incorrect. You gave 3 answers and got 2 correct of 3 needed.
1: The comparison between two numbers via == results in either True or False (in this case False), both Boolean values.
3: 3 + 4 evaluates to 7. 7 == 7 then evaluates to True, which is a Boolean value.
4: With the double quotes surrounding it, False is interpreted as a string, not a Boolean value. If the quotes had not been included, False alone is in fact a Boolean value.

Figure 4.6 Automated Assessment in PILeT

4.3.2. 6 Additional Features

PILeT also includes additional features that cannot be characterised as interactive. For example, textual content is available in the interactive tool for the same reason they are available in textbooks; to explain programming concepts, present some examples, and provide exercises and the model solutions associated with them. However the interactive tool allows more freedom to improve the quality of the text by combining it with other media forms to enhance the learning experience.

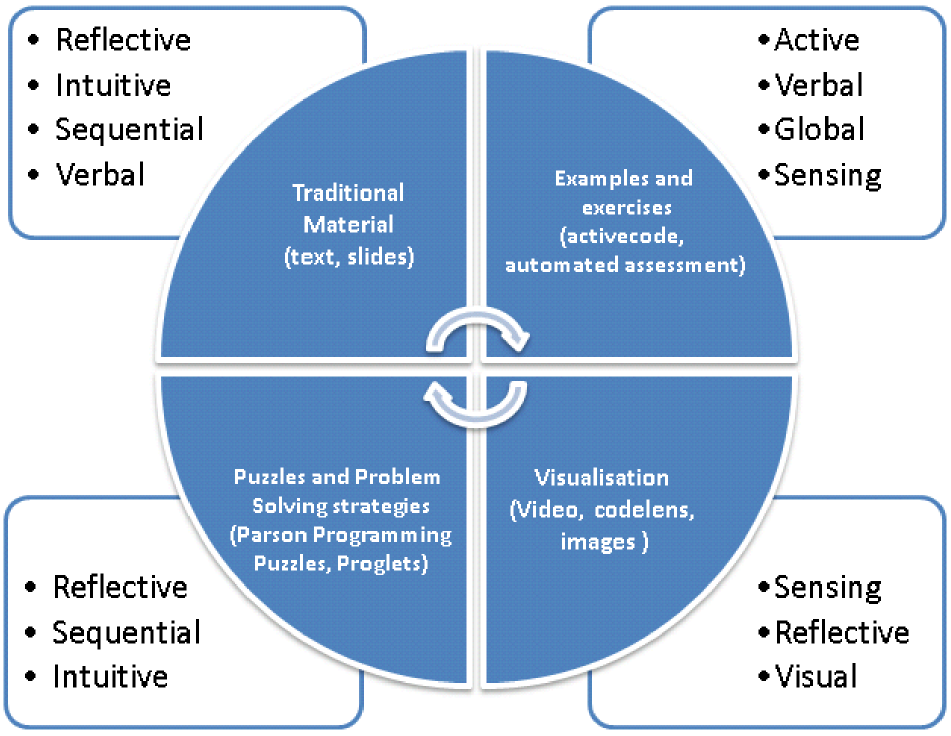
While traditional textbooks solely rely on textual paragraphs to explain ideas and rules, PILeT combines the text with embedded PowerPoint slides that are used for revising concepts. As such, the slides could be used as a suitable alternative to videos and textual explanation to learn about Python. Additionally, images, tables and GIFs (graphics interchange format) are used throughout PILeT to illustrate concepts. Hyperlinks to other content are used for additional reading material.[3]

4.4 Compliance with Felder-Silverman Learning Style Model

Figure 4.7 shows how PILeT support the diverse learners on the Felder-Silverman spectrum (Chapter 2).

- The traditional material in the form of text and slides appeal to (Reflective/Intuitive/Sequential/Verbal) learners.
- Examples and Exercises, both activecode and automated assessments, are favourable by (Active/Verbal/Global/Sensing) learners.
- Parson's Programming Puzzles and Proglets attract (Reflective/Sequential/Intuitive) learners.
- Videos, code visualisation and images cater to (Sensing/Reflective/Visual) learners.

[3] PILeT Demo Session can be found in Appendix 3.



Reflective
Intuitive
Sequential
Verbal
Active
Verbal
Global
Sensing
Traditional Material (text, slides)
Examples and exercises (activecode, automated assessment)
Puzzles and Problem Solving strategies (Parson Programming Puzzles, Proglets)
Visualisation (Video, codelens, images)
Reflective
Sequential
Intuitive
Sensing
Reflective
Visual

Figure 4.7 PILeT Supporting Felder-Silverman Learners

4.5 Compliance with Design Guidelines

A lot of consideration has gone into the design of PILeT. Both the interactive and non-interactive features that were included have been proven to improve comprehension of programming concepts, increase motivation in learners and improve problem solving skills. However, most of these features exist in isolation and the student is left with no option but to toggle between them which is distracting and inconvenient. PILeT has successfully combined all of these features in one interactive website, but in order to avoid any usability or design issues that might hinder the use of PILeT, a set of design guidelines were created. PILeT features comply with all of the design guidelines except for guidelines (1, 2 and 4)[4].

1. Ease of use

PILeT interactive features are very easy use as they require as few clicks as possible to reduce frustration. Additionally some of these features are already familiar to students (e.g. playing videos and slides, answering multiple choice questions). PILeT also contains a user manual at the beginning to guide the students through using the website. PILeT was evaluated for ease of use in (Chapter 5).

2. Discoverability

PILeT contains an intuitive interface which enables students to navigate through different pages without previous experience to locate content. Moreover, all of the pages are consistent throughout the website and follow the same structure and layout. PILeT was evaluated to identify any issues with the interface design (Chapter 5).

[4] The guidelines are evaluated in Chapter 5 and 6 respectively.

3. Interactive functionalities
Activecode, codelens, Parson’s Programming Puzzles and automated assessments are embedded interactive features that aim to increase engagement with programming without interfering with the learning process.

4. Customisation
The second version of PILeT supports customisation of teaching programming based on the students’ learning style preference as identified in the Felder-Silverman model. If the student is not comfortable with the presented teaching method, they have the option to switch to more suitable alternative. This guideline is evaluated in (Chapter 6).

5. Error handling
Activecode compiler messages are user friendly; it locates the error line, explains the type of error, and presents the user with helpful suggestion for fixing the error.

6. Automated assessment and feedback
PILeT provides swift assessment in the form of multiple choice questions. Additionally, instructors receive a notification once students submit their source code via PILeT which enables them to check the correctness of students’ solution, and provide extensive written feedback on their code. For some students, these questions serve as a way of self-regulating their own learning without having to read again all of the material (Bjork et al 2013).

7. Visualisation
Videos, GIFs, and codelens are different methods in which PILeT explains Python concepts visually. They also produce graphical representation of the programs’ different execution states and final output.

8. Improving problem solving techniques
Asking students to write lengthy programs whilst still new to programming concepts has negative impact on their problem solving abilities (Sweller 1988). In PILeT, learners are taught problem solving strategies by using exercises that require students to write small programs or proglets (Edmondson 2009). These little programs allow students to mainly focus on algorithm strategies without worrying about the quality of the source code at first. In addition, Parson’s Programming Puzzles in PILeT are used by students to solve programming problems in a stepwise manner.

9. Minimise cognitive overload
PILeT reduced cognitive overload by employing several worked examples in each chapter. These examples provide an opportunity for students to study the syntax and structure of Python. Teaching using worked examples is a good alternative to lengthy explanations (Brusilovsky 2001). Using images and tables in PILeT reduces cognitive overload in students as well as it minimises the amount of text they have to read.

10. Cover core programming concepts taught in introductory programming courses
PILeT consists of five chapters, they are:

- Variables and expressions.
- Selection statements (if statement).
- Loops (for loop, while loop)
- Functions.
- Lists.

Whilst these chapters do not teach programming in Python in its entirety, they cover all of the main Python concepts that are taught in introductory programming courses and recommended in the ACM curricula for computer science (The Joint Task Force on Computing Curricula 2013). They are also characterised as threshold concepts (Eckerdal et al 2006, Khalife 2006, Sorva 2010), as soon as they are mastered, students will be able to make connections with further programming ideas and progress positively in their learning.

11. Dependency
Embedding Activecode in PILeT allows students to write, modify and compile code in the same window as the programming question. They also execute code in the exact same way as external IDEs and produce error messages when syntax errors are detected.

4.6 Summary

Several learning environments fail to teach student programming due to usability issues associated with the overall design. As such, this chapter presented 11 design guidelines for the development of educational programming environments. These guidelines were based on thematic analysis of existing literature on designing learning environments, and with the evaluation of current programming tools. Results of the evaluation showed a need for a tool that fills the existing needs.

Therefore, a web based interactive learning tool to teach Python programming language (PILeT) to novice learners was developed. A description of the system architecture and features was included along with a demo in (Appendix 3). Finally the chapter concluded with a layout on how it complied with students on the Felder-Silverman model and most of the design guidelines except for three of them (ease of use, discoverability, and customisation).

The next chapter details the evaluation of PILeT based on ease of use and discoverability and the following chapter (Chapter 6) based on customisation.

Chapter Five: Usability and Discoverability Evaluation of PILeT

5.1 Chapter Overview

This chapter describes the process of evaluating PILeT by two groups: experts and users. As such this chapter is divided into two main sections: heuristic evaluation (Section 5.4) and user testing (Section 5.5). The expert group will evaluate PILeT based on design guideless 1: usability, and the user group will evaluate it based on design guideline 2: discoverability.

The heuristic evaluation section includes: usability heuristics, problems, recruitment process, evaluation method and results. The user testing section contains a detailed test plan which was used for evaluation, it includes: test artefacts, users profile and numbers, test environment and equipment, usability metrics, evaluation method and results.

The chapter concludes with a summary of both evaluations (Section 5.6).

5.2 Introduction to Usability Evaluation

Usability evaluation, usability testing and interface testing are all interchangeable terms that describe the process of identifying and recommending solutions for usability problems that arise as a result of a software design, websites or web applications (usability.gov 2013). The evaluation process involves collecting data that relates to the usability of a certain product by specific people for a specific activity within a specific work context (Preece et al 1994). Usability problems could encompass anything that interferes and hinders the user's ability to efficiently and effectively complete a required task in a satisfied manner (Karat et al 1992). According to Jakob Nielsen, usability is defined by a combination of five characteristics (Nielsen 2012):

- **Learnability:** how easy it is for a user who confronts an interface for the first time to perform a basic task.
- **Efficiency:** how fast can a user accomplish a task once they are familiar with the design.
- **Memorability:** how easy is it for a user to remember the interface and use it effectively after the first visit
- **Errors:** the number of errors that the user makes, the severity of those errors and recovering from those errors easily after committing them.
- **Satisfaction:** how satisfied is the user by the design of the system.

Usability evaluation of interfaces is necessary because it influences the users' experience of websites. If users struggle to locate the information they are looking for on a webpage in a timely fashion they simply abandon it for other available alternatives regardless of the quality of the website's content. Therefore the evaluation process should guarantee that the design of the user interface provides a friendly, natural, and clear environment for end users to interact with different elements of the website and complete tasks successfully.

Despite the importance of usability, there is surprisingly little published in the computer science education literature on the usability of educational software and websites. Instead, most of the research centres on evaluating and measuring the learning outcomes of end users after using these resources irrespective of how students or teachers interact and experience these learning tools (Levy and Ben-Ari 2007, Stern, Markham and Hanewald 2005). This problem arises from the lack of standardised design guidelines and usability testing for the development of educational software. This problem is significantly pertinent in tools with rich content such as interactive features and graphics. An early usability evaluation provides the developers with quick and valuable feedback during the design process of any problems that might occur with the interface and suggestions for solutions. This can be used in combination with other evaluation methods during the development process of these tools to ensure a seamless experience for the end user.

5.3 Usability Evaluation Methods

A number of well-established usability evaluation methods have been developed to identify usability problems and evaluate the user's interaction with the interface. Most of these methods are either classified as expert based, user based or automated software. The choice of a particular evaluation method depends on many factors, for example the interface development stage, type of users and level of involvement, expected kind of data and most importantly limitations such as time, cost and availability of eager testers (Aedo et al 1996) .

Expert based evaluation is a cheap and effective method that could be applied to a software during any stage of its development either as a prototype or end products. The objective is to present basic tasks as they are represented in the interface to a group of usability experts who would assume the role of representative end users and try and detect any issues in the interface design (Karoulis and Athanasis 2006). Whilst user based evaluation seems like an obvious alternative to recruiting experts, it comes with several drawbacks, for example, personal bias and subjectivity of testers, the users inability to articulate errors, missing them or providing useless information during the test through no fault of their own as they lack expert knowledge. Additionally, it is difficult to employ representative end users to evaluate interfaces under real life situations (Lewis and Rieman 1994). Moreover, a user based evaluation can only be applied at the end stage of a product. In contrast, expert based evaluations are inexpensive as it is easier to locate experts willing to participate in the evaluation process. They also provide valuable knowledge and offer different perspectives both in the field of HCI (human computer interaction) and other domains and as such can detect usability issues just looking at uncompleted projects.

The decision to apply a usability evaluation method over another is central to the success of a product. The choice is mostly influenced by the development stage the product is undergoing, time constraints, and the kind of data that needs to be collected for evaluation. Predicting user satisfaction with an end product is somewhat difficult by an expert as opposed to an end user, it is also difficult for an end user to reliably detect efficiency issues of an interface, therefore, it is recommended to combine different types of evaluation methods if possible (Dillon 2001). In light of the literature on usability evaluation, PILeT was evaluated by both expert users (Section 5.4.1) and end users (Section 5.5.1).

5.4 Heuristic Evaluation

A heuristic evolution is a usability inspection method that was developed by Nielsen and Molich (Nielsen and Molich 1990) for analysing and detecting any usability problems in the interface design. The methodology involves recruiting a small number of expert evaluators, usually five, to measure the user interface compliance with a set of usability principles or heuristics (Nielsen 1995). The heuristics are:

1. **Visibility of system status:** end users should constantly be updated and informed about operations occurring in the system (visible status) in a user friendly language within a reasonable timeframe.

2. **Match between the system and the real world:** the system should replicate the language and ideas that end users encounter in real world environment depending on their target users. This could be accomplished by presenting ideas in a logical manner which matches users' life experiences.
3. **User control and freedom:** give end users the option to backtrack steps in case of mistakes, this includes undo and redo commands.
4. **Consistency and standards:** designers of interfaces must guarantee that textual and graphical elements are consistent in meaning and function throughout the platform.
5. **Error prevention:** interface designers should aspire to keep the number of potential errors and mistakes to a minimum. This could be achieved by eliminating situation where errors are most likely to occur and offer the end user an option to confirm their selected action before proceeding.
6. **Recognition rather than recall:** end users' cognitive load should be kept to a minimum by making different elements and options more visible. The instructions to use the system and different relating dialogue should not be remembered by users, instead, different recognition techniques should be used by designers in the interface to navigate the system.
7. **Flexibility and efficiency of use:** for a quick and efficient navigation, designers should allow the use of shortcuts and customised commands in the interface for repeated actions of end users.
8. **Aesthetic and minimalistic design:** the interface should only contain the necessary elements for the completion of tasks. Therefore, supplementary information and dialogs should be kept to a minimum to avoid clutter.
9. **Help users recognise, diagnose and recover from errors:** error messages should be presented in a user friendly manner without technical jargon. The messages should be concise, describe the problem, and offer solutions in way that end users will be able to understand.
10. **Help and documentation:** although it is ideal to develop systems that can be used without the need of documentation for navigation, it may be necessary to include it at times specially for large systems with several functionalities. End users should be able to easily search and locate information related to the task they need assistance with.

Despite the popularity of Nielsen's heuristics, some researchers argue that they are too broad and ambiguous for the purpose of evaluating a website as these heuristics were developed before the popularity of the internet, and were initially intended to evaluate products with screens (Brinck et al 2002, Pearrow and Mark 2000). Therefore, researchers advise developing a new set of usability guidelines or design specific heuristics in line with Nielsen's for evaluating the usability of an interface (Rogers et al 2011).

A heuristic evaluation is performed by asking each inspector individually to examine the interface in order to collect impartial opinions. After the inspection is completed by everyone, the results are combined and analysed. The evaluation process is documented either by the use of recording devices that capture verbal comments of evaluators as they inspect the interface or by the use of written reports. The observer, who runs the evaluation session, can offer assistance to evaluators with any issues with the interface when necessary as opposed to user based evaluation where intervention is prohibited.

A usual session usually lasts two hours in which the evaluators explore the website more than once and examine several features and dialogues of the interface. These elements are then compared against the heuristics in addition to any usability guidelines that were developed for the website. At the end of the process, the evaluators give their suggestions and recommendations for the interface design.

There are several advantages to heuristic evaluation, they include:

- Being cost effective as they provide a reasonably quick and simple way to report feedback to interface designers during the design process. This is beneficial as it is easier to fix problems at an earlier stage rather than in a later phase.
- Each usability problem detected is examined against an established heuristic which make it easy to categorise and fix errors.
- A heuristic evaluation could be used to detect different error types in an interface (major and minor) and could be used in combination with other evaluation methods in order to refine any usability issues.
- A heuristic evaluation does not follow the same ethical consideration associated with recruited real life users.

5.4.1 Heuristic Evaluation of PILeT

For the evaluation, five experts used a set of ten comprehensive heuristics in order to detect any usability problems associated with PILeT's interface, and to identify any issues with the overall design of the website. The focus of the evaluation was on the main functionalities of the interface: registration and logging in, accessing chapters, using interactive features, and submissions of exercises and quizzes. The experts evaluated PILeT individually in separate sessions, and the recorded results were later aggregated along with comments and recommendations. As a result, the heuristic evaluation revealed some usability issues that were prioritised and addressed. A user based testing followed this evaluation to further investigate any problems with the usability of PILeT in addition to user satisfaction with the website.

5.4.1.1. Usability Heuristics

Since the inception of Nielsen's heuristics, several usability principles and guidelines emerged for the purpose of evaluating commercial websites. As an educational tool, PILeT has a slightly different interface similar to navigating an eBook but with interactive elements embedded in each page. For the purpose of this dissertation and for a successful evaluation process, a list of 10 heuristics was specifically developed for evaluating PILeT's interface. The list is in line with Jakob Nielsen's heuristics (Nielsen 1995) and Shneiderman's eight golden rules for interface design (Shneiderman 2004). The latter was specifically chosen as it has been used to evaluate educational websites aimed at novice learners. The list was reviewed and evaluated for suitability by two experts with experience in HCI and interface design. The heuristics are:

1. **Inform user of system status and offer feedback:** offer system feedback to users frequently, display the progress level of long tasks and provide notification messages when a process is completed.
2. **Speak in the user natural language:** use simple text to communicate with users, avoid technical jargon and use familiar terms.
3. **Allow easy navigation and reversal of actions:** make commands easily accessible, group related commands, and allow users to undo and redo actions.
4. **Strive for consistency across the system:** text format and graphics should be consistent within the system, this includes icons, texts labels and dialog boxes. Additionally, buttons must perform the same tasks throughout the system.
5. **Design dialogues to yield closure and allow exists:** actions should follow a sequence (beginning, middle end). Allow users to cancel or abort actions.

6. **Prevent errors and offer simple error handling:** offer users the option to confirm selection before proceeding, eliminate error prone situations and present errors in a simple language with suggested solutions.
7. **Reduce cognitive load:** employ recognition methods for system instructions, make menu options visible to users and use labels for buttons.
8. **Provide shortcuts for repeated actions:** reduce number of user interaction with interface by using shortcuts and macros for repetitive actions.
9. **Use minimalistic design:** keep information to minimum, use user friendly text for all user groups (e.g. users with dyslexia), employ colour and text size to highlight important ideas and use spacing between paragraphs to divide sections.
10. **Offer help at all times:** offer users assistance with all tasks when necessary and enable users to search for problem solutions.

5.4.1.2. Usability Problems Prioritisation

To categorise the usability issues discovered in the evaluation process, the problems were clustered in accordance with the defined heuristics above. In order to examine the effect of each detected issue, a ranking system was used in which each problem is rated based on it severity (Nielsen 1996) and ease of fixing the problem (Olson 2004). The severity rating was influenced by three factors:

- The frequency of the problem's occurrence: is it a rare or common problem?
- The impact of the occurred problem: is it easy or difficult to overcome the problem by users?
- The persistence of the usability problem: will the problem be solved once and for all by users or will it be repeatedly faced by them?

The following tables describe the severity rating for usability problems (Table 5.1) and ease of fixing rating (Table 5.2). Both tables were used to prioritise each usability problem (Table 5.4).

Rating	Severity Rating
0	I don't agree that this is a usability problem at all.
1	Cosmetic problem only: need not be fixed unless extra time is available on project.
2	Minor usability problem: fixing this should be given low priority.
3	Major usability problem: important to fix, so should be given high priority.
4	Usability catastrophe: imperative to fix this before product can be released.

Table 5.1 Severity Rating for Usability Problems

Rating	Ease of Fixing
0	Problem would be extremely easy to fix. Could be completed by one team member before next release.
1	Problem would be easy to fix. Involves specific interface elements and solution is clear.
2	Problem would require some effort to fix. Involves multiple aspects of the interface or would require team of developers to implement changes before next release or solution is not clear.
3	Usability problem would be difficult to fix. Requires concentrated development effort to finish before next release, involves multiple aspects of interface. Solution may not be immediately obvious or may be disputed.

Table 5.2 Ease of Fixing Problems

5.4.1.3. Recruitment

According to Nielsen, research shows that usability experts (people with double experience in both user interface design and usability issues) are better at detecting usability problems than other expert groups. However, these specialists are difficult to find and recruit. As a protective measure, Stone et al (Stone et al 2005) suggest hiring evaluators with different backgrounds such as domain expert, web designer or usability tester. As for determining the optimal number of evaluators, Nielsen (Nielsen 2000) discovered that the number relies on their work experience; if they are usability experts then hiring three to five evaluators will result in the discovery of 73% to 86% of usability issues whereas if they were specialists in more than one field, then hiring two to three experts will unveil between 80% to 91% of usability problems. If the evaluators are novices with no usability experience, then fourteen evaluators are at least needed to detect more than 70% of the problems.

For PILeT's heuristic evaluation, five experts who had personal connections with the author were invited to participate. All of the experts combined had different experiences, academically and commercially, in the fields of interface design, usability and online learning. Table 5.3 describes the profile of the evaluators.

Evaluator	Gender	Domain	Interface Design Experience	Usability Experience	Online Learning Experience
1	Female	Academia	Yes	Yes	Yes
2	Male	Academia	Yes	Yes	No
3	Female	Commercial	Yes	Yes	Yes
4	Male	Commercial	Yes	Yes	No
5	Male	Commercial	Yes	Yes	No

Table 5.3 Profile of Evaluators

5.4.1.4. Evaluation Process

At the beginning of the heuristic evaluation session, each expert was instructed to create a student profile and navigate PILeT freely for twenty minute in order to get a feel of the website and understand its objective. They were also encouraged to attempt the coding examples and exercises in the user manual. Next, the list of usability heuristics (Appendix 4) was given to them in order to compare the website interface with the usability principles whilst trying to complete different user tasks. For each detected usability problem, the expert had to measure it against a heuristic principle, rate the severity of the problem, ease of fixing it, and give any additional comments they wished to make. At the end of the evaluation, all of the results were combined and aggregated in one report.

5.4.1.5. Evaluation Results

Twelve usability problems were identified as a result of PILeT heuristic evaluation (Appendix 5). Some of these problems were detected more than once by different usability expert. Not all of the heuristics were violated (for example 2, 6, 8, and 10). Table 5.4 lists all of the problems ranked by the severity rating and ease of fixing. Most of the problems identified were minor or cosmetic usability issues which were easy to fix.

Number	Usability Problem	Usability Heuristic	Severity Rating	Ease of Fixing Rating
1	Once a profile is created, the user cannot access the profile page again to change credentials (for example password).	3. Allow easy navigation and reversal of actions.	4	3
2	The link to (Loops) chapter and its subsequent sections is broken	4. Strive for consistency across the system	4	2
3	There are dead and unnecessary buttons	4. Strive for consistency across the system	3	2
4	Instead of having the user manual as a separate section, make it compulsory and at the beginning as most users are new users.	5. Design dialogue to yield closure and allow exists 7. Reduce cognitive load	2	2
5	The confirm password field box is not immediately under the password field box.	4. Strive for consistency across the system	2	1
6	There is no clear indication that the user is logged in	1. Inform user of system status and offer feedback	2	1
7	The option to log in is visible to already logged in users	1. Inform user of system status and offer feedback	2	1
8	Users can see instructor's page option, it should be separate and at the beginning	3. Allow easy navigation and reversal of actions	2	1
9	Users are unaware of their progress level or whether a chapter is completed	1. Inform user of system status and offer feedback	1	1
10	Some tooltips have the same text as the button making them redundant	9. Use minimalistic design	1	0
11	Some concepts had a relatively larger number of material and exercises	9. Use minimalistic design:	0	0
12	The word "caution" is a bit alarming	5. Speak in the user natural language	0	0

Table 5.4 Usability Problems Ranked By Severity and Ease of Fixing

5.4.1.6. Heuristic Evaluation Summary

A detailed heuristic evaluation, based on a list of ten usability principles, was performed by five experts in order to detect any usability or design problems with the user interface. And whilst all of the evaluators agreed that PILeT's interface was user friendly and suitable for first time learners of programming, they detected a small number of problems that might interfere with the usability of PILeT. All of the issues and comments from the evaluators were grouped into twelve usability problems and ranked based on the severity rating and ease of fixing. All of the twelve problems were closely examined and fixed based on solutions and recommendations from the experts.

5.5 User Testing

User testing is another usability evaluation method that is conducted by users rather than experts. The objective is to measure the end users' satisfaction with the product, and how they interact with the interface under conditions similar to real life situations. In a typical test, a selection of representative users are hired to completed scripted tasks or scenarios successfully while the observer is watching and taking notes. At the end, users are asked to give a couple of statements about their satisfaction and experience with the product. In contrast to a heuristic evaluation, the test takes place at the end of the development stage of a product in order to detect any necessary changes to improve the performance of users (Usability.gov 2013).

5.5.1 User Testing of PILeT

For this task, a usability test plan was created to outline the steps and procedures to evaluate PILeT. The plan was executed after fixing the usability issues that were detected in the heuristic evaluation. The test plan included:

- Test artefacts.
- User profiles and number of users.
- Test environment and equipment.
- Usability metrics.
- Evaluation Method.
- Evaluation Results.

5.5.1.1. Test Artefacts

A couple of test artefacts were specifically developed for PILeT evaluation session. It consisted of:

- Moderator script from usability.gov (Appendix 6).
- Consent form (Appendix 7).
- Task scenarios (Appendix 8).
- Post-test questionnaire (Appendix 9)

Moderator Script

A moderator script from usability.gov was repurposed for this evaluation (Appendix 6). The script was used by the evaluator to welcome participants and thank them for their involvement. Additionally, the script provided information about the purpose of the evaluation and the process which will be recorded.

Consent Form

Participants were required to sign a consent form (Appendix 7) before participating in the evaluation. The form informed participants that their involvement was voluntary and as such they were free to withdraw from the evaluation at any time without giving a reason. It also stated that their data will be anonymously stored for later analysis.

Task Scenarios

A total of 14 tasks were developed for evaluation (Appendix 8). These tasks involved standard actions that a typical user of an educational programming website would perform. The tasks stemmed from a list of specific goals that PILeT aims to accomplish which include:

- Register with PILeT.
- Browse introductory Python concepts.
- Use different interactive features to learn.
- Submit exercises and quizzes.

In order for participants to perform the tasks, each task was assigned a scenario in accordance with Nielsen's recommendations for turning goals into task scenarios (Nielsen 2014). The tasks are:

1. Access PILeT by visiting www.obpilt.com
2. Create a user profile using the following credentials
    2.1. Username: evaluator1
    2.2. First name: Jane
    2.3. Last name: Doe
    2.4. Email: evaluator1@gmail.com
    2.5. Password: Abcd1234!
    2.6. Confirm password: Abcd1234!
    2.7. Course name: pilet
3. Skip the user manual.
4. Access (Variables and Expressions) chapter and browse the content.
5. Play the video titled (Input Statement).
6. Scroll down to (ActiveCode 1) then run the embedded code.
7. Change the code to print("Goodbye!"),save the code, delete it, load it, then run it.
8. Scroll down to (CodeLens1), step forward, back, first, last to trace code execution with Python Tutor.
9. Navigate to (Parson's Programming Puzzles 1), drag and drop the code blocks from the left column to the right column. Click (Check Me) and (Reset) buttons.
10. Navigate to the quiz section, try answering the first question by selecting an answer and clicking the (Check Me) button.
11. Move on to the next chapter (Selection Statements).
12. Edit the user profile you created by changing your first name to (Jack).
13. Navigate to (Functions) chapter.
14. Log off PILeT.

For each scenario, an estimated time and an optimal pathway to complete each task efficiently was detailed but hidden from participants.

Post-test Questionnaire

A post-test questionnaire (Appendix 9) was used to collect information regarding usability preferences and overall satisfaction with the interface design. The questionnaire was adopted from the World Wide Web Consortium (W3C 2003) as it is considered a validated and reliable instrument for usability testing. Respondents rate their agreement with 30 statements using a five point Likert scale ranging from (1. Strongly disagree - 1) to (5. Strongly agree).

5.5.1.2. User Profile and Number of Users

The primary target of the website is users with no coding experience interested in learning Python programming language. Based on this objective, a table of characteristics based on Rubin and Chisnell's work on identifying user profiles (Rubin and Chisnell 1994) was created.

Characteristic	Requirement
Age	Over 18
Education level	Undergraduate level
Experience using computers	1 + Years
Experience using the internet	1 + Years

Table 5.5 User Profile

As for the numbers of testers, a couple of usability experts recommend hiring between 8 and 10 participants (Brinck et al 2002, Rubin and Chisnell 1994) whilst Nielsen (Nielsen 2006) recommended 20 users in order to obtain statistically valid results as there are significant individual differences when tasks are performed by users.

For the recruitment process, an email was sent out to first year undergraduate student at Oxford Brookes University at the Department of Computing and Communication Technologies. The email contained a description of the study and eligibility requirements. However, only five respondents volunteered due to time restrictions and coursework load. After screening the participants, all of them were recruited.

(It is important to note that PILeT was evaluated for usability again later in the year during a study to measure the learning outcomes of PILeT using a SUS (System Usability Scale)) in Chapter 6.

5.5.1.3. Test Environment and Equipment

The evaluation took place in a computer lab with internet connection. The operating system of the computer used to access PILeT was Windows 7 with the latest version of Google Chrome web browser installed.

The observer recorded each participant's actions, reactions and opinions using a notepad and an audio recorder.

5.5.1.4. Usability Metrics

In order to accurately measure the user's usability of PILeT, four usability metrics were used based on Nielsen's recommendations (Nielsen 2001):

- **Success rate:** the percentage in which a task is completed by the user correctly.
- **Task time:** the time it requires to complete each task successfully in minutes.
- **Error rate:** the frequency the error occurs over a specific time.

- **Satisfaction:** a user's subjective satisfaction with the task. This metric is measured using SEQ (Single Ease Question); a seven point Likert Scale about the difficulty of the task (Sauro 2012).

5.5.1.5. Evaluation Method

The evaluation took place over 5 individual sessions. Each session lasted an hour in which every participant was briefed at the beginning about the objective of the study and test procedures (read from the moderator script by the observer). After signing the consent form, the participant was asked to complete a set of 14 tasked scenarios whilst concurrently thinking out loud to understand their thought process and the reasoning behind their actions (usability.gov 2014). After each scenario was completed, a question was asked about the participants' satisfaction with the task. At the end of the evaluation, a post-test questionnaire was given to each participant in order to get their overall impression of the website. During the evaluation, the observer recorded the usability metrics of each task (Appendix 10).

5.5.1.6. Evaluation Results

Most of the tasks listed in the scenarios (Appendix 7) were completed successfully without any problems. However there were some minor issues reported with tasks (6, 7, 8, and 11):

1. Visit PILeT website: *AVG satisfaction score: 7 / 7*
The task involved visiting the website for the first time. All participants managed to complete the task without any errors.

2. Create user profile: *AVG satisfaction score: 7 / 7*
Testers had to create a user profile with given credentials. No problems were reported with this task.

3. Skip user manual: *AVG satisfaction score: 7 / 7*
The task involved bypassing the user manual to access the content. The task was completed successfully.

4. Access (Variables and Expressions) chapter and browse content: *AVG satisfaction score: 7 / 7*
Participants were required to visit the requested chapter. No issues were reported.

5. Play video: *AVG satisfaction score: 7 / 7*
The task involved playing a video that explained the concept. All participants located the play button.

6. Run ActiveCode: *AVG satisfaction score: 6.6 / 7*
The task involved executing a piece of embedded code. New users of PILeT would not know what that means unless they read the user manual, which was skipped for the purpose of the evaluation. Since the participants completed the task successfully after explaining the term, it was concluded that the task itself was not problematic but rather the way it was approached.

7. Edit, save, load and run code: *AVG satisfaction score: 5.8 / 7*
This task, like number 6, involved a minimum understanding of how programming works. The first participants who failed to complete the task argued that while they understood what "save" and "load" referred to, they had difficulties applying it in a programming context. The second participant kept making syntactical errors until they eventually succeeded. Both of these mistakes could be attributed to understanding programming rather than usability issues with the task.

8. Use Codelens: *AVG satisfaction score: 6.8 / 7*
1 participant asked for the meaning of the terms “frames” and “objects”, a non-usability issue.

9. Use Parson’s Programming Puzzles: *AVG satisfaction score: 7 / 7*
The task involved dragging blocks of code from one side to the other in addition to resetting the order. All participants managed to complete the task successfully.

10. Answer quiz: *AVG satisfaction score: 7 / 7*
Participants were required to test out the quiz functions regardless of the correctness of the answer. The task was completed without any reported errors.

11. Move to next chapter: *AVG satisfaction score: 7 / 7*
1 participant suggested that the button should be bigger.

12. Edit user profile: *AVG satisfaction score: 7 / 7*
Participants were asked to change some information on their profile. No problems were reported.

13. Navigate to (Function) chapter: *AVG satisfaction score: 7 / 7*
The task involved navigating from the current page to another chapter. The task was completed successfully by all of them.

14. Log off PILeT: *AVG satisfaction score: 7 / 7*
Participants were asked to log off the website by locating the log off button. All participants completed the task.

Overall, these results suggest that participants only struggled with tasks involving programming rather than using the interactive feature itself that was being evaluated (ActiveCode). This issue could have been easily prevented by asking them to try out the feature in the user manual which describes its use in detail.

5.5.1.7 Post-test Questionnaire results

Appendix 11 show five respondents’ satisfaction with the usability and the interface design of PILeT. Any responses with rating from 1 to 3 were considered negative statements (usability problems) whereas responses with 4 or 5 rating were considered positive ones (non-usability problems).

Most of the answers ranged between (4. Agree) and (5. Strongly agree). Interestingly, questions 27 through 30 received a rating of 5 by all respondents.

27. The site has a clear purpose.
28. I always felt I knew what it was possible to do next.
29. It is clear how screen elements (e.g., pop-ups, scrolling lists, menu options, etc.) work.
30. My mistakes were easy to correct.

These questions are concerning the objective of the website, the sequence of actions to complete a task, usability of different elements, and error handling.

Combined together, the results from the user testing and the post-test questionnaire show that users had little to no problems browsing the website or using its interactive features. Additionally, users responded favourably towards the interface design of PILeT.

5.6 Summary

This chapter described the process of evaluating PILeT using two methods: heuristic evaluation (Section 5.4.1) and user testing (Section 5.5.1). The objective of this evaluation is to measure PILeT compliance with design guideline 1: usability and design guideline 2: discoverability.

The heuristic usability evaluation of was carried out by experts to detect any usability issues associated with the overall design of PILeT. For this evaluation, a set of 10 comprehensive heuristics was developed specifically for educational websites. As a result of this evaluation, a number of usability problems were identified and fixed based on experts recommendations.

A user based testing followed the heuristic evaluation in order to measure end users' satisfaction with the interface and discoverability. For this evaluation, a number of represented users were hired and asked to complete 14 tasked scenarios. The evaluator observed users' interaction with different interface elements and recorded 4 usability metrics for each task. The results of this evaluation showed that users have a positive impression of PILeT.

The next chapter presents the methodology of measuring the learning outcomes of PILeT in addition to usability guideline 4: customisation.

Chapter Six: Learning Outcomes and Customisation Evaluation of PILeT

6.1 Chapter Overview

This chapter describes the process of measuring the learning outcomes of PILeT by comparing the evaluation results against another education tool. In order to establish a fair comparison, both tools were evaluated for usability (Section 6.4) before measuring the learning outcomes (Section 6.5). (Section 6.6) measures design guideline 4: customisation. The chapter concludes with a summary of all evaluations (Section 6.7).

6.2 Learning Outcomes Measurement

A central choice in evaluating an educational tool is selecting the evaluation method and measuring the learning outcomes. Unfortunately, this process is complex and involves a wide range of variables (e.g. efficacy, mental models, motivation and learning environment) to name a few.

In most programming modules, a combination of programming exercises, quizzes and the final mark would be used to assess the students' performance in a module. Ideally in the same vein, developers of educational programming tools would like to use the same evaluation methods to measure students' performance. However, there are several ethical constraints with that method, most importantly, a group of students cannot be divided and offered different teaching methods within the same module as one group might be at a disadvantage which eventually affects their final mark. Another problem with learning outcomes evaluation is the willingness of participants themselves to fully commit to using the educational tool at specific times without the influence of other teaching methods for a long period. Whilst this might be achieved with paid participants keen on learning programming, it is very difficult to recruit them. As a result, many researchers settled for conducting an experiment to measure the results of one programming task (Redmiles 1993, Rowe and Thorburn 2000) or use pre and post-tests in a single experimental session to evaluate the tool (Milne and Rowe 2002b, Shah and Kumar 2002).

In order to get meaningful results, PILeT was evaluated by measuring the students' performance after learning two programming concepts using two different student groups at different times. Additionally, their performance was compared against users of another educational tool; CS Circles.

6.3 CS Circles

Computer Science Circles (Pritchard and Vasiga 2013) is a free interactive website designed to teach Python programming language to a novice audience with no previous programming experience. The website was developed by the Centre for Education in Mathematics and Computing at the University of Waterloo. CS Circles was chosen for comparison for two reasons:

- It teaches the same introductory concepts as PILeT (variables and expressions, selection statements, loops, functions and lists).
- It contains some similar interactive features embedded in the website. These features are:
    - A built in console for code execution, parallel to ActiveCode.
    - CodeLens for code visualisation; the same in PILeT.
    - Code scrambler: an exercise for arranging code blocks in the right order. Parallel to Parson Programming Puzzles.

Whilst CS Circles sounds like good contender to PILeT at first, it suffers from several drawbacks (number 1 and 3 were reported by the developers themselves):

1. **Interactive input and output is currently unavailable in the website:** all of the examples in CS Circles are fixed and not editable, as such, users can only run the code to see the result of the execution. If they wanted to experiment with the program they have to write it again in the console. PILeT's Activecode however allows users to execute worked examples, modify them, or experiment with their own code in the same window without switching to another console.
2. **The console displays the exact same error messages as other IDE's:** whilst this is not considered a significant issue as the error messages in Python are user friendly, they can be quite intimidating for first time learners. In ActiveCode, errors are described in a manner that matches the user's natural language, in terms of words and phrases, instead of system oriented expressions. It also offers suggestions for fixing the error.
3. **The code scrambler does not contain extra lines or space for the solution:** this means that users have to rearrange the code in the same space which is inconvenient. On the other hand, Parson Programming Puzzles contains two columns; one for the scrambled code on the left and the other is for the solution on the right. This makes it easy for users to drag the code blocks in the right order.
4. **CS Circles was not evaluated for usability:** whilst the developers have usage statistics, it does not give a clear picture of how users interact with the website or their impression of the design. For example, the only recorded data is: the number of submitted code and automated assessments solved regardless of the correctness of the solution for both. Another detected concern is that the developers use the time elapsed between user registration and exercise completion as the time it took for a user to complete an exercise which is unrealistic.
5. **The concept pages are not consistent:** in terms of content length, uniformity of interface design, and the number and type of interactive features and exercises.

Despite these drawbacks, CS Circles was considered a suitable tool for comparison purposes as it is bears the closest resemblance to PILeT in terms of usability and learning objectives. Furthermore, comparing the interface design and interactive elements of PILeT against CS Circles before measuring the learning outcomes of both will produce unbiased results about the usability of each website, and evidence on whether those features affect the learning process of users.

6.4 Usability Evaluation

Due to time constraints, it was unrealistic to conduct a usability evaluation for CS Circle the same way it was carried out for PILeT in (Chapter 5). For the purpose of ensuring a fair comparison, both websites were evaluated using a usability survey and an SUS (System Usability Scale) questionnaire (Brooke 1996).

The usability survey was distilled from a systematic review of studies on the requirements of e-book designs and interactive instructions (Chong et al 2009a, Ericson et al 2015, Fenwick et al 2013, Korhonen et al 2013, Pei Fen Chong et al 2008, Perrenet and Kaasenbrood 2006, Ruth Wilson et al 2009). The survey investigates the design preferences of both interfaces in terms of:

- **Navigation design:** the quality of seamlessly navigating between webpages to find specific information.
- **Page layout:** the arrangement of different website elements, in addition the amount of content.

- **Content design:** the approach in which the visual and textual elements are displayed on the interface.
- **Interactive features:** the usefulness of the built-in features.

The SUS questionnaire is a simple instrument that is used to measure the efficiency, effectiveness and satisfaction of websites. The questionnaire consists of 10 questions with 5 options for respondents ranging from (1. Strongly disagree) to (5. Strongly agree). The average SUS score is 68; a score above 68 is considered (above average) whereas a score below is considered (below average). The score is then normalised to produce percentile ranking and grades from A to F. For example, an SUS score of 64 converts to a percentile rank of 60% which is interpreted as grade C.

6.4.1 Participants and Method

Participants were recruited from the Department of Computing and Communication Technology at Oxford Brookes University. A call for participation was sent out to first year undergraduate students via email which includes a short description of study and eligibility requirements. 49 students volunteered to take part, all of which been screened and recruited for the study.

Prior to the study, a 10 minute demonstration of PILeT and CS Circles took place at computer lab to introduce the students to both websites, and allow them enough time for registration and signing the consent form (Appendix 12). Next, the students were instructed to experiment with the websites, and navigate between the different pages on their own for 30 minutes. After the time was over, the students were asked to complete the usability survey (Appendix 13) and SUS questionnaire (Appendix 14) for PILeT and CS Circles online using Google forms. To avoid any bias that might result from first impressions (using a website first before the other), 24 participants were randomly assigned to start with PILeT first and 25 to start with CS Circles first. The study lasted 60 minutes.

6.4.2 Results

For the usability survey, out of 49 participants, 46 responses were received (24 for CS Circles and 22 for PILeT). In terms of navigation design, Table 6.1 shows that the students preferred the navigation design of PILeT in terms of ease of navigation, page landing, and locating other pages within the website.

No	Statement	Statement	CS Circles	PILeT
1	I found it easy to navigate [website name here].	Strongly disagree	0	1
		Disagree	7	1
		Neither agree nor disagree	6	7
		Agree	9	7
		Strongly agree	2	6
2	It was easy to know which page I was on and what other pages I visited.	Strongly disagree	0	0
		Disagree	7	1
		Neither agree nor disagree	11	6
		Agree	3	7
		Strongly agree	3	8
3	It was easy to locate certain pages of [website name here]	Strongly disagree	0	1
		Disagree	9	0
		Neither agree nor disagree	8	10
		Agree	7	5
		Strongly agree	0	6

Table 6.1 Navigation Design Results

Table 6.2 shows positive responses towards the layout of PILeT in regards to the amount of information on the page, the arrangement of the interactive features, and satisfaction towards the table of content. On the other hand, CS Circles received lower scores.

For content design preferences, PILeT received higher rating across the board in terms of page length, page legibility, typography, scan reading and page colour as seen in Table 6.3.

Finally, Table 6.4 shows that a great number of students were able to determine the purpose and usefulness of Active Code, Codelens and Parson Programming Puzzles in PILeT. In contrast, they struggled to identify the purpose of the console, Codelens and Code scrambler in CS Circles. Therefore it is no surprise that the student did not find them useful.

The results from the usability survey indicate that PILeT's interface design make it easier for students to navigate the website, read the learning material, and use the different interactive features of the website, which makes it a favourable choice amongst them.

No	Statement	Statement	CS Circles	PILeT
1	I am happy with the amount of information on the page.	Strongly disagree	4	0
		Disagree	5	3
		Neither agree nor disagree	6	4
		Agree	9	8
		Strongly agree	0	7
2	I am happy with the arrangement of the interactive features.	Strongly disagree	3	1
		Disagree	10	1
		Neither agree nor disagree	4	5
		Agree	7	7
		Strongly agree	0	8
3	I am happy with the table of content.	Strongly disagree	4	0
		Disagree	7	3
		Neither agree nor disagree	5	5
		Agree	8	6
		Strongly agree	0	8

Table 6.2 Page Layout Results

No	Statement	Statement	CS Circles	PILeT
1	I found the font easy to read.	Strongly disagree	0	0
		Disagree	3	3
		Neither agree nor disagree	4	4
		Agree	12	5
		Strongly agree	5	10
2	I found the page of acceptable length.	Strongly disagree	0	0
		Disagree	3	3
		Neither agree nor disagree	9	7
		Agree	12	5
		Strongly agree	0	7
3	I found the page legible.	Strongly disagree	1	1
		Disagree	4	1
		Neither agree nor disagree	7	6
		Agree	10	5
		Strongly agree	2	9
4	I liked the typographical aspects of the page.	Strongly disagree	0	0
		Disagree	9	3
		Neither agree nor disagree	8	6
		Agree	6	6
		Strongly agree	1	7
5	I found it easy to scan through the page.	Strongly disagree	2	1
		Disagree	10	4
		Neither agree nor disagree	5	5
		Agree	6	3
		Strongly agree	1	9
6	I liked the colours of the page.	Strongly disagree	4	3
		Disagree	7	1
		Neither agree nor disagree	9	8
		Agree	3	5
		Strongly agree	1	5

Table 6.3 Content Design Results

Number	Statement	CS Circles		PILeT	
1	I knew the purpose of	Console	Yes No 8 16	ActiveCode	Yes No 22 0
2	I found the feature useful	Console	Yes No 14 10	ActiveCode	Yes No 20 2
3	I knew the purpose of	Codelens	Yes No 7 17	CodeLens	Yes No 22 0
4	I found the feature useful	Codelens	Yes No 15 19	CodeLens	Yes No 18 4
5	I knew the purpose of	Code scrambler	Yes No 1 23	Parson's Programming Puzzles	Yes No 22 0
6	I found the feature useful	Code scrambler	Yes No 11 13	Parson's Programming Puzzles	Yes No 19 3

Table 6.4 Interactive Features Results

As for the SUS questionnaire, out of 49 participants, only 24 responses were returned for CS Circles (Appendix 15) and 25 for PILeT (Appendix 16). The average usability score of 24 CS Circles questionnaires was 46.7, which is below the SUS average (68). After normalising the score, it produced a percentile ranking of grade F. For PILeT, the average usability score of 25 questionnaires was 72.4, which is above the SUS average. After normalising the score, it produced a percentile ranking of grade B, a significantly better ranking than CS Circles. This result means that CS Circles has failed the usability test whilst PILeT is considered usable.

In summary, the results of the usability study suggest that users are satisfied with the interface design and usability of PILeT. Based on those findings, it is possible to assume that the learning outcomes of PILeT would be greater than CS Circles because the content design and interactive features of the website do not obstruct the learning process of users interested in programming. This assumption will be tested in the next section.

6.5 Learning Outcomes Evaluation

The research hypothesis states that "combining multiple teaching methods to accommodate different learners' preferences will significantly improve performance in programming". In order to test the hypothesis, PILeT and CS Circles were used on two separate occasions to teach two different Python concepts (selection statements and loops). Their performance was compared against students who learned those concepts using the apprenticeship model (Alshaigy et al 2015).

In the apprenticeship model, students are first exposed to a Python concept using PowerPoint slides. Next, they spend a short period, usually 15 minutes, practising these concepts by solving small related programming exercises under the supervision of the lecturer. This process is repeated until the end of the lecture.

Although this method of teaching shows promising results, it is very expensive to administer and manage by the department. Since PILeT is suggested as a suitable alternative for teaching programming, as it caters to several learners' preferences as a lecturer would, the apprenticeship model students were used as the control group and PILeT and CS Circles students as the experimental group.

6.5.1 Participants and Method

For this study, first year undergraduate students, who were already enrolled in a Python module at the university, volunteered to take part.

6.5.1.1. Selection statements

In order to evaluate the learning outcome of each teaching method, 57 students were randomly and equally grouped as follow:

- Apprenticeship model group: 19 participants. (9 students withdrew halfway through the study).
- CS Circles group: 19 participants.
- PILeT group: 19 participants.

After learning the concept, the students were tested with a quiz consisting of 3 programming questions (Appendix 17). For each question they were awarded:

- 0 marks for no submissions, incomplete or major mistakes.
- 1 for minor mistakes.
- 2 for correct answer.

The quiz was marked out of 6. The study lasted 75 minutes.

6.5.1.2. Loops

As for evaluating the learning outcome of loops, 35 students were randomly grouped as follow:

- Apprenticeship model group: 10 participants.
- CS Circles group: 11 participants.
- PILeT group: 14 participants.

After learning the concept, the students were tested with a quiz consisting of 4 short programming questions (Appendix 18) with the same marking scheme as selection statements. The quiz was marked out of 8. The study lasted 75 minutes as well.

All of the questions included in both quizzes were similar to the examples and exercises they learn from in the lecture or both websites.

6.5.2 Results

The results of each participant on the quiz are in Appendix 19. The descriptive statistics associated with students' performance on the selection statements quiz is summarised in Table 6.1.

For the results, a statistical analysis was conducted using a one way ANOVA technique in order to find out whether or not a difference exists somewhere between the three different groups. The single factor in this analysis is the teaching method. Therefore:

Null hypothesis (H_0): combining multiple teaching methods to accommodate different learners' preferences has no effect on the students' performance in programming selection statements.

Alternative hypothesis (H_A): combining multiple teaching methods to accommodate different learners' preferences has a positive effect on the students' performance in programming selection statements.

Between the three teaching methods, the CS Circles group was associated with a numerically smallest mean level for students' mark (M = 3.42), and the apprenticeship model group, the control group, was associated with the numerically highest mean level for students' marks (M = 5.9).

A one way ANOVA between subjects was applied to compare the effect of the teaching method on the quiz marks of three different students groups (Table 6.6). The results present statistically significant evidence that there is a difference in the mean of the groups that use different teaching methods (F(2, 45) = 9.92, p = <0.001). Therefore based on the analysis, the null hypothesis is rejected and the data supports the alternative hypothesis.

Groups	Count	Sum	Average	Variance
Control Group	10	59	5.9	0.1
CS Circles	19	65	3.421	2.035
PILeT	19	89	4.684	3.228

Table 6.5 Summary of Students' Results (Selection Statements)

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	42.175	2	21.087	9.922	0.00026	3.204
Within Groups	95.636	45	2.125			
Total	137.812	47				

Table 6.6 One Way ANOVA (Selection Statements)

In order to rule out the chance of committing a Type 1 error (incorrect rejection of a true null hypothesis; a false positive), a post hoc t-test called the Bonferroni correction was conducted between the three groups. Another benefit of the test is to explore where the differences between the three group means are found.

The new critical p value is:

P value (α) / no comparison
0.05 / 3 = 0.0167

As seen in Table 6.7, the value of P(T<=t) two-tail for the control group is (0.0000). Since the result of the comparison between (0.0000 < 0.0167) is True, it can be concluded that there is a 95% chance that the control group is different than the CS Circles group.

Moving on to Table 6.8, the value of the P(T<=t) for the control group is (0.0447). Since the result of the comparison between (0.0447 < 0.0167) is False, it can be concluded that there are no significant differences between the Control group and PILeT.

	Control Group	**CS Circles**
Mean	5.9	3.421
Variance	0.1	2.035
Observations	10	19
Pooled Variance	1.390	
Hypothesized Mean Difference	0	
Df	27	
t Stat	5.381	
P(T<=t) one-tail	5.468	
t Critical one-tail	1.7032	
P(T<=t) two-tail	0.00001	
t Critical two-tail	2.0518	

Table 6.7 t-test Between Control Group and CS Circles Group (Selection Statements)

	Control Group	PILeT
Mean	5.9	4.684
Variance	0.1	3.228
Observations	10	19
Pooled Variance	2.185	
Hypothesized Mean Difference	0	
Df	27	
t Stat	2.105	
P(T<=t) one-tail	0.022	
t Critical one-tail	1.703	
P(T<=t) two-tail	0.044	
t Critical two-tail	2.051	

Table 6.8 t-test between Control Group and PILeT Group (Selection Statements)

For the loops concept, a list of the students' results is in Appendix 15. Table 6.9 describes a summary the students' performance in the loops quiz.

For the results, a statistical analysis was conducted using a one way ANOVA technique in order to find out whether or not a difference exists somewhere between the three different groups. The single factor in this analysis is the teaching method. Therefore:

Null hypothesis (H_0): combining multiple teaching methods to accommodate different learners' preferences has no effect on the students' performance in programming loops.

Alternative hypothesis (H_A): combining multiple teaching methods to accommodate different learners' preferences has a positive effect on the students' performance in programming loops.

Between the three teaching methods, it can be seen that the CS Circles group was associated with a numerically smallest mean level for students' mark (M = 1.27) and the PILeT group was associated with the numerically highest mean level for students' marks (M = 5.71).

Again, a one way ANOVA between subjects was applied to compare the effect of the teaching method on the quiz marks (Table 6.10). The results present statistically significant evidence that there is a difference in the mean of the groups that use different teaching methods (F(2, 32) = 19.24, p = <0.001). Therefore based on the analysis, the null hypothesis is rejected and the data supports the alternative hypothesis.

Groups	Count	Sum	Average	Variance
Control Group	10	53	5.3	3.3444
CS Circles	11	14	1.272	5.818
PILeT	14	80	5.714	2.065

Table 6.9 Summary of Students' Results (Loops)

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	138.461	2	69.230	19.240	0.0000	3.294
Within Groups	115.138	32	3.598			
Total	253.6	34				

Table 6.10 One Way ANOVA (Loops)

As seen in Table 6.11, the value of P(T<=t) two-tail for the control group is (0.0004). Since the result of the comparison between (0.0004 < 0.0167) is True, it can be concluded that there is a 95% chance that the control group is different than the CS Circles group.

As for Table 6.12, the value of of the P(T<=t) for the control group is (0.05404). Since the result of the comparison between (0.0540 < 0.0167) is False, it can be concluded that there are no significant differences between the Control group and PILeT.

	Control Group	CS Circles
Mean	5.3	1.2727
Variance	3.34444	5.8181
Observations	10	11
Pooled Variance	4.64641	
Hypothesized Mean Difference	0	
Df	19	
t Stat	4.2760	
P(T<=t) one-tail	0.00020	
t Critical one-tail	1.7291	
P(T<=t) two-tail	0.0004	
t Critical two-tail	2.0930	

Table 6.11 t-test Between Control Group and CS Circles Group (Loops)

	Control Group	PILeT
Mean	5.3	5.71428
Variance	3.34444	2.0659
Observations	10	14
Pooled Variance	2.5889	
Hypothesized Mean Difference	0	
Df	22	
t Stat	-0.6218	
P(T<=t) one-tail	0.2702	
t Critical one-tail	1.7171	
P(T<=t) two-tail	0.5404	
t Critical two-tail	2.0738	

Table 6.12 t-test Between Control Group and PILeT Group (Loops)

In summary, for both concepts (selection statements and loops), the ANOVA test was significant, and a Bonferroni corrected post-test (t-test) indicate the control group is significantly higher than CS Circles whereas no significant differences were found between the control group and PILeT. This means that the results of the learning outcome evaluation show that teaching programming using CS Circles has negative consequences on students' learning whereas using PILeT for teaching has the same effect as the apprenticeship model. This result confirms the assumption made in usability study (Section 6.4.2) that the content design and interactive features of a website could either contribute or distract the students from the learning process.

6.6 PILeT 1.1

In the initial learning outcomes evaluation study (Section 6.5), Google Analytics was implemented to track the students' use of different elements of the website (e.g. textual, visual), and match it against their learning styles preferences which the students' identified at the beginning of the study using the ILS (Index of Learning Style) questionnaire (Felder and Soloman 1997) (Appendix 20). However, after looking at the analytics report, it was discovered that all of the students clicked on every website element regardless of their learning preference. This could be attributed to many reasons: fear of missing out on important information, not getting used to the website, or even because the student did actually need the supplementary material in a different medium to understand the concept. Therefore, it was decided that in order to test the "accommodate different learners' preferences" part of the hypothesis correctly in addition to guideline 4: customisation, PILeT needs to be customised for each users' needs and evaluated again.

PILeT 1.1 was released after 2 weeks. The difference between the old version and the new one is that instead of presenting the user with all of the teaching content in all of the different formats available, users get customised pages based on their learning style choice. So for example, if a user identifies as:

Active learners - ~~Reflective learners~~
Visual learners - ~~Verbal learners~~
~~Sensing learners~~ - **Intuitive learners**

Then the user will be presented with:

- Videos to explain the concept
- Codelens for code visualisation.
- Supplementary images.
- Interactive examples and exercise (ActiveCode, automated assessments).

If another user identifies as:

~~Active learners~~ - **Reflective learners**
~~Visual learners~~ - **Verbal learners**
Sensing learners - ~~Intuitive learners~~

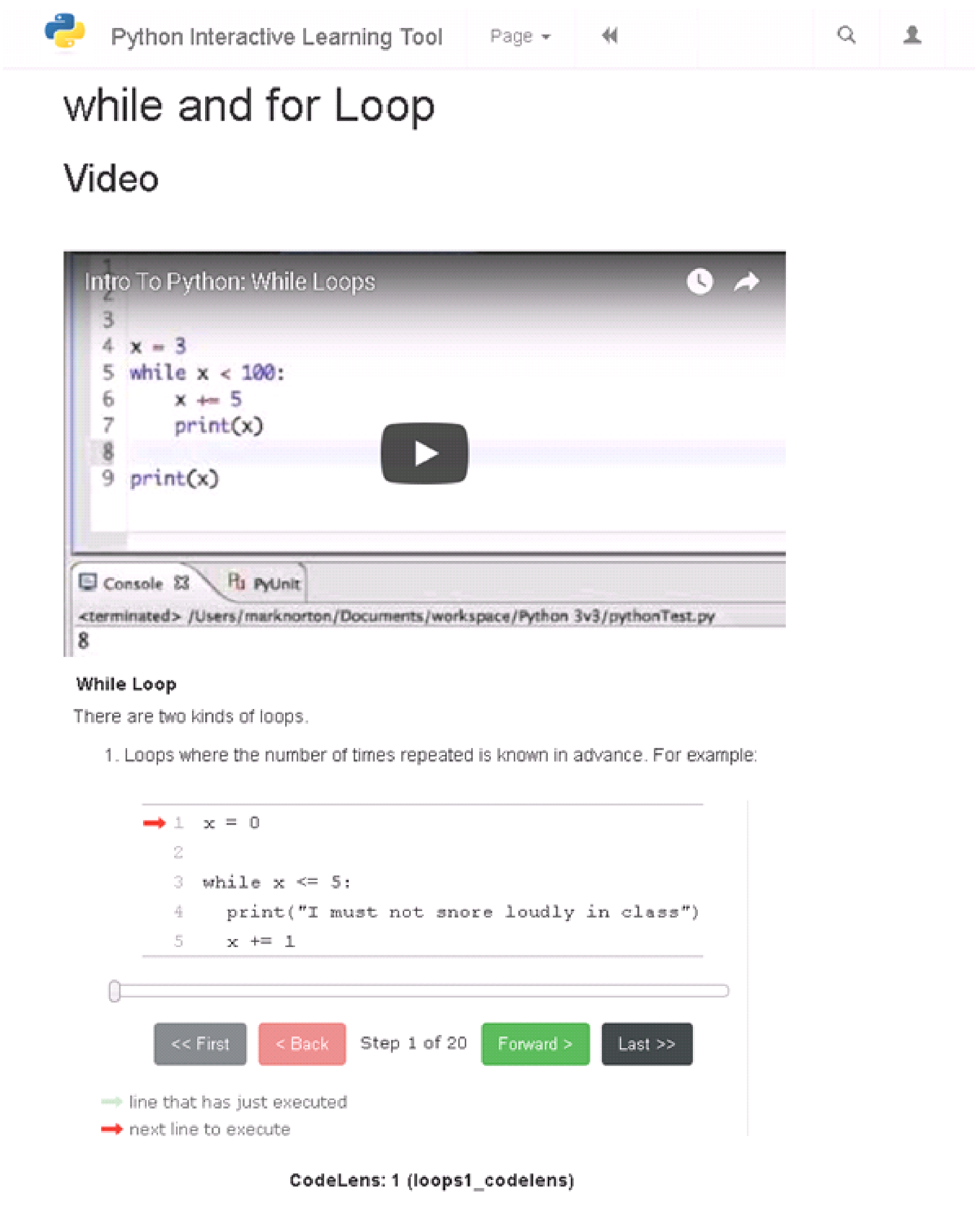
Then the user will be presented with:

- Text and slides to explain the concept.
- Hyperlinks that directs the user to additional reading material.
- Parson's Programming Puzzles.
- Programming questions.

Due to time restrictions, the new version of PILeT was developed to teach only two concepts (selection statements, loops) and as such, the (sequential, global learners) classification was not considered.

6.6.1 Demo Session

New users to the website register in the same way they did in the previous version by submitting their details. Next, users fill in a form consisting of 44 multiple choice questions to determine their learning style based on the four dimensions identified in the Felder – Silverman model. After completing the index of learning styles questionnaire, users are directed to a customised selection statements page with content based on their preference. The forward button at the top of the page takes the users to a similarly customised loops page. Figure 6.1 shows a screenshot of a customised loops page for a visual user.


Python Interactive Learning Tool
Page
Intro To Python: While Loops
Console
PyUnit
<< First
< Back
Step 1 of 20
Forward >
Last >>
line that has just executed
next line to execute

while and for Loop

Video

```
x = 3
while x < 100:
    x += 5
    print(x)

print(x)
```

```
<terminated> /Users/marknorton/Documents/workspace/Python 3v3/pythonTest.py
8
```

While Loop

There are two kinds of loops.

1. Loops where the number of times repeated is known in advance. For example:

```
x = 0

while x <= 5:
  print("I must not snore loudly in class")
  x += 1
```

CodeLens: 1 (loops1_codelens)

Figure 6.1 Screenshot of Loops Page (Visual User)

The next section describes the learning outcomes evaluation of PILeT 2.0. The study follows the same structure and format as the previous one (Section 6.5), however CS Circles was not included in the study as the results about its usability were already conclusive.

6.6.2 Participants and Method

For the evaluation, two user groups volunteered to take part:

- First year undergraduate students already enrolled in a Python module at Oxford Brookes University.
- Undergraduate students from Oxford University enrolled in a CodeFirst: Girls Python course.

6.6.2.1. Selection statements

In order to evaluate the learning outcome of each teaching method, 24 Oxford Brookes University students were randomly and equally grouped as follow:

- Apprenticeship model group: 12 participants
- PILeT group: 12 participants.

After learning the concept, the students were tested with a quiz consisting of 3 programming questions (Appendix 17). For each question they were awarded:

- 0 marks for no submissions, incomplete or major mistakes.
- 1 for minor mistakes.
- 2 for correct answer.

The quiz was marked out of 6. The evaluation lasted 75 minutes.

6.6.2.2. Loops

As for evaluating the learning outcome of loops, 13 Oxford University students were used as the experimental group, and 12 Oxford Brookes University students were used as the control group.

- Apprenticeship model group: 12 participants
- PILeT group: 13 participants.

After learning the concept, the students were tested with a quiz consisting of 4 short programming questions (Appendix 18) with the same marking scheme as selection statements. The quiz was marked out of 8. The study lasted 75 minutes as well.

6.6.3 Results

The results of each participant on the quiz are in Appendix 21. The descriptive statistics associated with students' performance on the selection statements quiz is summarised in Table 6.13.

For the results, a statistical analysis was conducting using a two independent sample t-test in order to find out whether there is a statistically significant difference between the means of the two groups. Therefore:

Null hypothesis (H_0): for selection statements, there is no difference between the mean marks of the apprenticeship model group (control group) and PILeT group.

Alternative hypothesis (H_A): for selection statements, there is a difference between the mean marks of the apprenticeship model group (control group) and PILeT group.

Between the two teaching methods, the PILeT group was associated with a numerically highest mean level for students' mark (M = 5.58), and the apprenticeship model group, the control group, was associated with the numerically smallest mean level for students' marks (M = 4.92).

An independent-samples t-test was applied to compare the effect of the teaching method on the quiz marks of two different student groups (Table 6.14). The results present no statistically significant evidence that there is a difference in the mean of the PILeT group (M = 5.58, SD = 0.793) and control group (M = 5.58, SD = 1.240) conditions; t(22) = -1.568, p = 0.131. Additionally, the value of (T<=t) two-tail for the control group is (0.131) is not less than 0.005. Therefore based on the analysis, we fail to reject the null hypothesis.

Groups	Count	Sum	Average	Variance	SD
Control Group	12	59	4.9166	1.5378	1.240
PILeT	12	67	5.58333	0.6287	0.793

Table 6.13 Summary of Students' Results (Selection Statements)

	Control Group	PILeT
Mean	4.9166	5.5833
Variance	1.5378	0.6287
Observations	12	12
Pooled Variance	1.0833	
Hypothesized Mean Difference	0	
Df	22	
t Stat	-1.5689	
P(T<=t) one-tail	0.0654	
t Critical one-tail	1.7171	
P(T<=t) two-tail	0.1309	
t Critical two-tail	2.0738	

Table 6.14 t-test Between Control Group and PILeT Group (Selection Statements)

For the loops concept, a list of the students' results is in Appendix 17. Table 6.15 describes a summary the students' performance in the loops quiz.

For the results, a statistical analysis was conducting using a two independent sample t-test in order to find out whether there is a statistically significant difference between the means of the two groups. Therefore:

Null hypothesis (H_0): for loops, there is no difference between the mean marks of the apprenticeship model group (control group) and PILeT group.

Alternative hypothesis (H_A): for loops, there is a difference between the mean marks of the apprenticeship model group (control group) and PILeT group.

Between the two teaching methods, the PILeT group was associated with a numerically highest mean level for students' mark (M = 7.17), and the apprenticeship model group, the control group, was associated with the numerically smallest mean level for students' marks (M = 5.25).

Again, an independent-samples t-test was applied to compare the effect of the teaching method on the quiz marks of two different student groups (Table 6.16). The results present statistically significant evidence that there is a difference in the mean of the PILeT group (M = 7.17, SD = 0.832) and control group (M = 5.25, SD = 1.712) conditions; t(22) = -3.485, p = 0.002. Additionally, the value of (T<=t) two-tail for the control group is (0.002) is less than 0.005. Therefore based on the analysis, the null hypothesis is rejected and the data supports the alternative hypothesis.

Groups	Count	Sum	Average	Variance	SD
Control Group	12	63	5.25	2.9318	1.712
PILeT	13	94	7.2307	0.6923	0.832

Table 6.15 Summary of Students' Results (Loops)

	Control Group	PILeT
Mean	5.25	7.1666
Variance	2.9318	0.6969
Observations	12	12
Pooled Variance	1.8143	
Hypothesized Mean Difference	0	
Df	22	
t Stat	-3.4854	
P(T<=t) one-tail	0.0010	
t Critical one-tail	1.7171	
P(T<=t) two-tail	0.0020	
t Critical two-tail	2.0738	

Table 6.16 t-test Between Control Group and PILeT Group (Loops)

In summary, for selection statements, the independent-samples t-test results show that there are no statistical differences between the means of the two groups, whereas for loops, the results show that the mean of the PILeT group is statistically higher than the mean of the control group. This means that the results of the learning outcomes evaluation show that whilst teaching programming in PILeT yields higher results for both concepts, it was proven that the difference were not significant in one instance (selection statement) and significant in the other (loops). Combing these results with the findings from (Section 6.5.2), indicates that teaching programming using PILeT has the same or greater affect than teaching it using the apprenticeship model. This means that in cases where the apprenticeship model is not administered, PILeT would be considered a suitable alternative.

6.7 Summary

This chapter described the process of evaluating PILeT in terms of customisation and learning outcomes against another educational website; CS Circles.

For the evaluation, a usability survey in addition to an SUS questionnaire were used for the process. The evaluation results show that students prefer the interface design and the usability of PILeT over CS Circles for learning programming.

As for the learning outcomes evaluation, a quiz was used to measure the students' performance in two programming concept (selection statements and loops). Their performance was compared against students who learned Python using the apprenticeship model. The evaluation results show that teaching programming using PILeT has the same or greater effect than teaching it in the apprenticeship model, while it has negative effects teaching it using CS Circles.

In the next chapter, a reflection on the research statement and objectives is included along with threats to validity and future work.

Chapter Seven: Conclusion and Future Work

7.1 Chapter Overview

The final chapter in this dissertation reflects on the research statement, objectives and contributions. It also presents some threats that might have influenced the results of the studies. The chapter concludes with recommendations for future work.

7.1 Reflection on the Research Statement and Objectives

The motivation behind this research was to improve the teaching of introductory programming courses at university by using a pedagogical tool to support students' learning. Whilst several educational software and online systems are already available for that purpose, several learners struggle to use them due to major design issues. In addition, many of these environments fail to recognise the learning styles exhibited by a diverse cohort of student. Therefore, the research stated the hypothesis that combining multiple teaching methods to accommodate different learners' preferences will significantly improve performance in programming.

Consequently, the research's' main contribution is devising a set of 11 essential design guidelines for the development of educational programming environments. These guidelines were derived by combining the results of an extensive literature review on educational software with established design guidelines for websites. In order to evaluate them, an interactive learning tool, PILeT, was developed which conformed to the guidelines. PILeT offers a combination of pedagogical methods to support students' learning style. As such, each programming concept is explained using videos, reading material, examples, and exercises adequately on their own, or by mixing these approaches together. Additionally, automated assessment methods and code submissions are available at the end of each lesson to assess the students' comprehension and performance on each taught concept. Finally, PILeT was tested by combining usability tests with measurements of conceptual understanding of programming knowledge in students.

The development of the tool was accomplished by meeting five core objectives as outlined in the beginning of the research in Chapter 1. Below is a summarised description of how they were achieved.

Objective one: identify essential design guidelines for the development of pedagogical tools

For this objective, an extensive literature review was conducted to support the hypothesis in Chapter 2 and to identify the guidelines for the development stage in Chapter 4. The areas covered:

1. Predominant programming problems exhibited by first time learners.
2. Teaching methods in introductory courses.
3. Learning styles and preferences in students.
4. Evidence of existing relationships between learning styles and teaching methods.
5. The implications of the choice of the first programming language taught to students.
6. Examples of popular programming environments for novices.
7. Students' and teachers' use of pedagogical tools.

Following an analysis of the literature review and synthesis of the findings, a set of eleven design principles were developed for PILeT. They are:

1. Ease of use.
2. Discoverability.
3. Interactive functionalities.
4. Customisation.
5. Error handling.
6. Automated assessment and feedback.
7. Visualisation.
8. Improving problem solving techniques.
9. Minimise cognitive overload.
10. Cover core programming concepts taught in introductory programming course.
11. Dependency.

Objective two: develop the interactive tool (PILeT) based on those guidelines

Chapter 4 illustrated the process of developing PILeT including a layout of the system's architecture, a list of the interactive features, in addition to a description of a demo session. The chapter also explained how PILeT complied with the design guidelines and the Felder-Silverman learning style model.

Objective three: perform a heuristic evaluation of PILeT

For this evaluation, a set of 10 comprehensive heuristics was developed specifically for educational websites based on work by prominent usability consultants Jakob Nielsen (Nielsen 1995) and Ben Shneiderman (Shneiderman 2004) (Chapter 5). The heuristics are:

1. Inform user of system status and offer feedback.
2. Speak in the user natural language.
3. Allow easy navigation and reversal of actions.
4. Strive for consistency across the system.
5. Design dialogue to yield closure and allow exists.
6. Prevent errors and offer simple error handling.
7. Reduce cognitive load.
8. Provide shortcuts for repeated actions.
9. Use minimalistic design.
10. Offer help at all times.

Following this step, a heuristic evaluation of PILeT was carried out to detect any usability problems associated with the interface design by five usability experts. The process resulted in identifying 12 usability problems which were addressed and fixed based on experts' recommendations.

Objective four: perform user based evaluation of PILeT

A user based testing followed the heuristic evaluation of PILeT in order to measure end users' satisfaction with the interface and improve their experiences with the tool. For this objective, five represented users were hired and asked to complete 14 tasked scenarios for testing the website (Chapter 5). The results of this evaluation showed that users responded favourably towards the design and usability of PILeT.

Objective five: measure learning outcomes of PILeT

In order to get meaningful results for the learning outcomes, PILeT was evaluated by measuring the students' performance after learning two programming concepts (selection statements and loops) and comparing their results against users of another education tool called CS Circles (Chapter 6).The evaluation results showed that teaching programming using PILeT significantly improved performance in programming which confirms the hypothesis.

7.2 Threats to Validity

During the course of this research, every measure was taken to ensure the accuracy and reliability of the design and methods. In few cases, a number of issues that might influence the validity of results were encountered. Some were addressed at the time, for example how to evaluate support tool in Chapter 6, the rest are acknowledged and discussed below.

7.2.1 Internal Validity

Broadly speaking, experimental research design is always under the threat of violating a couple of rules that might affect its validity mainly because either the results are influenced by other factors that were not considered, or the interpretation of results is flawed (Reis and Judd 2000). Some of the factors that affected this research were:

Research participants:

Whilst the recruitment process aimed to be as diverse as possible, all of the research participants who volunteered for the studies are either personal contacts or current students of the researcher. This is not surprising as the experiment design required lengthy and mentally taxing activities that might not interest people who are not keen on learning programming. Even with the current computing students at the university, it was difficult to get them engaged in the study without offering monetary incentives. Therefore, it was difficult to get the target sample size for the learning outcomes evaluation study. To address this shortfall, this research used statistical methods that deal with small sample sizes (one way ANOVA, two independent sample t-test which compares the means of the groups (Sauro 2013) in addition to repeating the experiment to ensure the reliability of the results.

Another main concern with participants was controlling the factors that might influence their performance during the study such as demotivation (Robert Fitzgerald and Fuller 1982), programming anxiety (Connolly et al 2009) and subjects bias (Furnham 1986). To minimize their effects, volunteered were constantly reminded about the objective of the research and the anonymity of their responses.

Study Design:

For the learning outcomes study, the experiment setup does not reflect a real life situation. For instance, the time it requires a student to learn a concept varies from one person to another. In addition, quiz scores exclusively are not indicative of comprehension of programming concepts or ability. In order to reduce its impact, careful consideration went into deciding the concepts that were being tested to ensure that they are just challenging enough to reflect participants' understanding. In addition, there was not a time limit for participants to complete the study although they might have felt pressured to finish when other participants left early.

7.2.2 Validity of Learning Styles

There is a lot of controversy and negative connotations associated with the term “learning styles”. Many argue that the theories are either myths or pseudoscience that is not supported by validated scientific evidence (Association for Psychological Science 2016). Therefore, it is redundant to customise teaching instructions to accommodate learners if there is no added learning value. However, there is evidence to support the idea that “people do learn differently” (Willingham 2016) but there is a difference between style and ability.

Irrespective of the learning style model, if a student is offered several options and resources for learning, they have the freedom to choose the method or medium they are comfortable with which results in an increase in interest and retention. And whilst some of the learning styles models are questionable, or based on self-report questionnaires, or even designed in such a way to support their theory, does that deny the existence of learning styles? Suppose a student expressed difficulties learning programming from a textbook, and they identify themselves as "visual learner", regardless of whether their diagnosis is correct, should the instructor dismiss their preference and insist on proceeding with the textbook rather than offer alternatives such as code visualisation? Moreover, some researchers assume that "categorising individuals can lead to the assumption of fixed or rigid learning style, which impair motivation to apply oneself or adapt " (The Guardian 2017), a fact which is disputed by proponents of learning styles as they recognise that students’ preferences are fluid and develop overtime due to the teaching environment or other aspects. As a matter of fact, the Felder-Silverman model specifically does not pigeonhole learners into categories, instead it postulates that students have a tendency towards a preference more than the other depending on the concept they are learning or even the subject. Furthermore, even though the process of customising the teaching material based on the learning style is costly, it is a one-off cost. That is not to say that instructors should spend their time classifying each learning style in existence and producing appropriate material accordingly, but instead they should offer a number of alternative options beneficial to many students.

7.3 Recommendations for Future Work

For the future development of PILeT, a number of recommendations are suggested to address any limitations or threats that were encountered during the research.

For an accurate measurement of the learning outcomes of PILeT, the tool should be integrated in the lectures during the whole academic semester. Not only will it guarantee a larger sample size, but testing the tool with a greater number of concepts at the end would give a better reflection of students’ abilities in programming. For evaluation purposes, qualitative methods should be used in combination with quantitative techniques in order to get a rich and comprehensive understanding of how students’ really feel about the tool as a teaching instrument. This process should be replicated multiple times with different participants and concepts to ensure the validity and reliability of results.

The design of PILeT allows it to be repurposed for teaching other Computer Science topics or expand to different fields with minimal configurations. Replacing traditional teaching with interactive features engages learners, increases participation, and improves students’ understanding in the long run.

Another area of consideration with huge potential is harnessing the power of artificial intelligence for developing customised content based on learners’ preference and progress, and providing meaningful feedback. The sheer computational power of AI has the ability to track students over extended periods and capture nuances that would go undetected by the instructor and thus catering to learners' cognitive skills and psychological needs. The flexibility and convenience of AI will allow students to manage and control their own learning even outside the lecture.

7.4 Summary

Learning programming is a notoriously difficult. Computer Science educators are constantly researching solutions to elevate this difficulty. Given the recent advancement in educational technologies, many are inclined to adopt these resources in their teaching. However, these initiatives tend to impede students' progress due to several reasons mainly because these tools were designed for specific user groups. This research detailed the process of developing and testing an educational programming tool for a diverse audience taking into account their individual preferences and needs. One of the main contributions is devising a set of standardised design guidelines for the development of educational tools. The findings of this research show promising results that could be expanded to other fields.

References

Aedo I, Catenazzi N and Díaz P (1996) *The evaluation of a hypermedia learning environment: the CESAR experience*. *Journal of Educational Multimedia and Hypermedia*. Association for the Advancement of Computing in Education. Available at: http://dl.acm.org/citation.cfm?id=228071 (accessed 23/08/17).

Agarwal KK and Agarwal A (2006) Simply Python for CS0. *J. Comput. Sci. Coll.* 21(4): 162–170.

Alaoutinen S and Smolander K (2010) Are Computer Science Students Different Learners? .

Allert J (2004) Learning style and factors contributing to success in an introductory computer science course. *IEEE International Conference on Advanced Learning Technologies, 2004. Proceedings.* IEEE, 385–389. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1357442 (accessed 07/06/16).

Allert J (2004) Learning Style and Factors Contributing to Success in an Introductory Computer Science Course. *Proceedings of the IEEE International Conference on Advanced Learning Technologies*. IEEE Computer Society.

Alshaigy B, Kamal S, Mitchell F, Martin C and Aldea A (2015) PILeT: An Interactive Learning Tool To Teach Python. *Proceedings of the Workshop in Primary and Secondary Computing Education* (Figure 1): 76–79. Available at: http://doi.acm.org/10.1145/2818314.2818319.

Ardito C, De Marsico M, Lanzilotti R, Levialdi S, Roselli T, Rossano V and Tersigni M (2004) Usability of E-learning tools. *Proceedings of the working conference on Advanced visual interfaces - AVI '04*. New York, New York, USA: ACM Press, 80. Available at: http://portal.acm.org/citation.cfm?doid=989863.989873 (accessed 01/04/18).

Association for Psychological Science (2016) *Learning Styles Debunked: There is No Evidence Supporting Auditory and Visual Learning, Psychologists Say – Association for Psychological Science*. . Available at: http://www.psychologicalscience.org/news/releases/learning-styles-debunked-there-is-no-evidence-supporting-auditory-and-visual-learning-psychologists-say.html#.WUqE3OvyuHu (accessed 21/06/17).

Astrachan O, Bruce K, Koffman E, Kölling M and Reges S (2005) Resolved. *ACM SIGCSE Bulletin* 37(1): 451–452. Available at: http://portal.acm.org/citation.cfm?doid=1047124.1047359 (accessed 26/10/16).

Bayliss JD (2009) Using games in introductory courses. *Proceedings of the 40th ACM technical symposium on Computer science education - SIGCSE '09*. New York, New York, USA: ACM Press, 337. Available at: http://portal.acm.org/citation.cfm?doid=1508865.1508989 (accessed 07/06/16).

Beaubouef T, Lucas R and Howatt J (2001) The *UNLOCK* system. *ACM SIGCSE Bulletin*. ACM 33(2): 43. Available at: http://portal.acm.org/citation.cfm?doid=571922.571953 (accessed 07/06/16).

Beaubouef T and Mason J (2005) Why the high attrition rate for computer science students. *ACM SIGCSE Bulletin*. ACM 37(2): 103. Available at: http://portal.acm.org/citation.cfm?doid=1083431.1083474 (accessed 07/06/16).

Bishop-Clark C, Courte J, Evans D and Howard E V. (2007) A Quantitative and Qualitative Investigation of Using Alice Programming to Improve Confidence, Enjoyment and Achievement among Non-Majors. *Journal of Educational Computing Research*. SAGE PublicationsSage CA: Los Angeles, CA 37(2): 193–207. Available at: http://journals.sagepub.com/doi/10.2190/J8W3-74U6-Q064-12J5 (accessed 13/03/17).

Bjork RA, Dunlosky J and Kornell N (2013) Self-Regulated Learning: Beliefs, Techniques, and

Illusions. *Annu. Rev. Psychol* 64: 417–44. Available at: http://cognitrn.psych.indiana.edu/rgoldsto/courses/cogscilearning/bjorkdunlosky.pdf (accessed 05/06/17).

Blikstein P, Worsley M, Piech C, Sahami M, Cooper S and Koller D (2014) Programming Pluralism: Using Learning Analytics to Detect Patterns in the Learning of Computer Programming. *Journal of the Learning Sciences* 23(4): 561–599. Available at: http://www.tandfonline.com/doi/abs/10.1080/10508406.2014.954750 (accessed 27/01/17).

Bornat R and Dehnadi S (2008) Mental models, Consistency and Programming Aptitude. .

Borstler J, Hall MS, Nordstr M, Paterson JH, Sanders K, Schulte C and Thomas L (2010) An evaluation of object oriented example programs in introductory programming textbooks. *SIGCSE Bull.* 41(4): 126–143.

Bosse Y and Gerosa MA (2017) Why is programming so difficult to learn? *ACM SIGSOFT Software Engineering Notes*. ACM 41(6): 1–6. Available at: http://dl.acm.org/citation.cfm?doid=3011286.3011301 (accessed 29/03/18).

Bouvier D, Zarb M, Lovellette E, Matta J, Alshaigy B, Becker BA, Craig M, Jackova J, McCartney R and Sanders K (2016) Novice Programmers and the Problem Description Effect. *Proceedings of the 2016 ITiCSE Working Group Reports on - ITiCSE '16*. New York, New York, USA: ACM Press, 103–118. Available at: http://dl.acm.org/citation.cfm?doid=3024906.3024912 (accessed 31/03/18).

Brandl G (2008) *Overview — Sphinx 1.6.3+ documentation*. . Available at: http://www.sphinx-doc.org/en/stable/ (accessed 31/05/17).

Braun V and Clarke V (2006) Using thematic analysis in psychology. *Qualitative Research in Psychology* 3(2): 77–101. Available at: http://www.tandfonline.com/doi/abs/10.1191/1478088706qp063oa (accessed 01/04/18).

Briggs Myers I and Myers PB (1980) Gifts differing: Understanding personality type. *Palo Alto: CPP Books*.

Brinck T, Gergle D and Wood SD (2002) *Designing Web sites that work : usability for the Web*. Morgan Kaufmann Publishers.

Brooke J (1996) SUS-A quick and dirty usability scale. *Usability evaluation in industry* 189(194): 4–7.

Bruce KB (2004) Controversy on how to teach CS 1. *Working group reports from ITiCSE on Innovation and technology in computer science education - ITiCSE-WGR '04*. New York, New York, USA: ACM Press, 29. Available at: http://portal.acm.org/citation.cfm?doid=1044550.1041652 (accessed 25/10/16).

Bruce KB and Bruce KB (2004) Controversy on how to teach CS 1. *Working group reports from ITiCSE on Innovation and technology in computer science education - ITiCSE-WGR '04*. New York, New York, USA: ACM Press, 29. Available at: http://portal.acm.org/citation.cfm?doid=1044550.1041652 (accessed 07/06/16).

Brusilovsky P (2001) *WebEx: Learning from Examples in a Programming Course*. . Available at: http://www.pitt.edu/~peterb/papers/WebNet01.html (accessed 04/06/17).

Brusilovsky P, Grady J, Spring M and Lee C-H (2006) What should be visualized?: faculty perception of priority topics for program visualization. *SIGCSE Bull.* 38(2): 44–48.

Carbone A, Hurst J, Mitchell I and Gunstone D (2000) Principles for designing programming exercises to minimise poor learning behaviours in students. *Proceedings of the Australasian* *conference on Computing education*. Melbourne, Australia: ACM.

Carbone A, Hurst J, Mitchell I and Gunstone D (2001) Characteristics of programming exercises that lead to poor learning tendencies: Part II. *Proceedings of the 6th annual conference on Innovation and technology in computer science education*. Canterbury, United Kingdom: ACM.

Carbone A, Hurst J, Mitchell I and Gunstone D (2009) *An exploration of internal factors influencing student learning of programming*. *Proceedings of the Eleventh Australasian Conference on Computing Education - Volume 95*. Australian Computer Society, Inc.

Carlisle MC, Wilson TA, Humphries JW, Hadfield SM, Carlisle M, Humphries J, Hadfield S and af mil U (2005) RAPTOR : A Visual Programming Environment for Teaching Algorithmic Problem Solving. . Available at: https://www.researchgate.net/profile/Martin_Carlisle/publication/221537443_RAPTOR_A_visual_programming_environment_for_teaching_algorithmic_problem_solving/links/0912f5102d415b0198000000.pdf (accessed 13/03/17).

Carmo L, Marcelino M and Mendes A (2007) The Impact of Learning Styles in Introductory Programming Learning. *In International Conference on Engineering Education-ICEE*.

Carter J and Jenkins T (2010) The problems of Teaching Programming: Do They Change with Time? *11 th Annual Conference of the Subject Centre*. Available at: http://www.academia.edu/download/30848015/download.pdf#page=9 (accessed 25/10/16).

Carter P and Paul (2009) An experiment with online instruction and active learning in an introductory computing course for engineers. *Proceedings of the 14th Western Canadian Conference on Computing Education - WCCCE '09*. New York, New York, USA: ACM Press, 103. Available at: http://portal.acm.org/citation.cfm?doid=1536274.1536305 (accessed 02/06/17).

Carver CA, Howard RA and Lane WD (1999) Enhancing student learning through hypermedia courseware and incorporation of student learning styles. *IEEE Transactions on Education* 42(1): 33–38. Available at: http://ieeexplore.ieee.org/document/746332/ (accessed 31/01/17).

Cha S, Kwon D and Lee W (2007) Using puzzles: problem-solving and abstraction. *Proceedings of the 8th ACM SIGITE conference on Information technology education*. Destin, Florida, USA: ACM.

Chin C and Brown DE (2000) Learning in Science: A Comparison of Deep and Surface Approaches. *Journal of Research in Science Teaching*. Wiley-Blackwell 37(2): 109–138. Available at: http://doi.wiley.com/10.1002/%28SICI%291098-2736%28200002%2937%3A2%3C109%3A%3AAID-TEA3%3E3.0.CO%3B2-7 (accessed 31/03/18).

Chong P, Lim Y and Ling S (2009a) On the Design Preferences for Ebooks. *IETE Technical Review* 26(3): 213. Available at: http://tr.ietejournals.org/text.asp?2009/26/3/213/50706 (accessed 02/09/17).

Chong P, Lim Y and Ling S (2009b) On the Design Preferences for Ebooks. *IETE Technical Review* 26(3): 213. Available at: http://tr.ietejournals.org/text.asp?2009/26/3/213/50706 (accessed 01/03/17).

Chudowsky N, Glaser R and Pellegrino JW (2001) Knowing what students know: The science and design of educational assessment. *National Academy Press*.

Clark RC and Mayer RE (2016) *E-Learning and the Science of Instruction: Proven Guidelines for Consumers ... - Ruth C. Clark, Richard E. Mayer - Google Books*. John Wiley & Sons. Available at: https://books.google.co.uk/books?hl=en&lr=&id=v1uzCgAAQBAJ&oi=fnd&pg=PR17&dq=e-Learning+and+the+science+of+instruction:+proven+guidelines+for+consumers+and+designers+of+multimedia+learning&ots=TLBLkHaQ9m&sig=DajHmw8clMc44bfFZBZuBg-0jzU#v=onepage&q (accessed 01/03/17).

Cliburn DC and Miller S (2008) Games, stories, or something more traditional: the types of assignments college students prefer. *SIGCSE Bull.* 40(1): 138–142.

Coffield F, Moseley D, Hall E and Ecclestone K (2004) *Learning styles and pedagogy in post 16 learning: a systematic and critical review*. London: The Learning and Skills Research Centre.

Connolly C, Murphy E and Moore S (2009) Programming Anxiety amongst Computing Students – A Key in the Retention Debate ? . Available at: https://www.researchgate.net/profile/Cornelia_Connolly/publication/220675700_Programming_Anxiety_Amongst_Computing_Students_-_A_Key_in_the_Retention_Debate/links/57d81e8808ae0c0081edf189.pdf (accessed 12/09/17).

Crump BJ (2004) New Arrival Students: Mitigating Factors on the Culture of the Computing Learning Environment. 30.

Curtis SA (2005a) Word puzzles in Haskell. *Proceedings of the 2005 workshop on Functional and declaritive programming in education - FDPE '05*. New York, New York, USA: ACM Press, 15. Available at: http://portal.acm.org/citation.cfm?doid=1085114.1085119 (accessed 07/06/16).

Curtis SA (2005b) Word puzzles in Haskell: interactive games for functional programming exercises. *Proceedings of the 2005 workshop on Functional and declarative programming in education*. Tallinn, Estonia: ACM.

Cutts Q, Haden P, Sutton K, Box I, Hamer J, Lister R, Tolhurst D, Fincher S, Anthony Robins kentacuk, Baker B, de Raadt M, Hamilton M, Petre M and Tutty J (2006) The Ability to Articulate Strategy as a Predictor of Programming Skill. 52.

Dale NB (2006) Most difficult topics in CS1: results of an online survey of educators. *SIGCSE Bull.* 38(2): 49–53.

Dillon A (2001) The Evaluation of software usability Item type Book Chapter The evaluation of software usability. . Available at: http://hdl.handle.net/10150/105344 (accessed 24/08/17).

Dingle A, Zander C, Dingle A, Zander C, Dingle A and Zander C (2000) Assessing the ripple effect of CS1 language choice. *Journal of Computing Sciences in Colleges*. Consortium for Computing Sciences in Colleges 16(2): 85–93.

Eagle M, Barnes T, Eagle M and Barnes T (2008) Wu's castle. *Proceedings of the 13th annual conference on Innovation and technology in computer science education - ITiCSE '08*. New York, New York, USA: ACM Press, 245. Available at: http://portal.acm.org/citation.cfm?doid=1384271.1384337 (accessed 07/06/16).

Eckerdal A, McCartney R, Mostr JE, Ratcliffe M, Sanders K and Zander C (2006) Putting threshold concepts into context in computer science education. *SIGCSE Bull.* 38(3): 103–107.

Edmondson C (2009) Proglets for first-year programming in Java. *SIGCSE Bull.* 41(2): 108–112.

EDU-SIG (2016) *EDU-SIG: Python in Education | Python.org*. . Available at: https://www.python.org/community/sigs/current/edu-sig/.

Elza D, Goodger D and Wiemann L (2012) *Creating reStructuredText Directives*. . Available at: http://docutils.sourceforge.net/docs/howto/rst-directives.html (accessed 31/05/17).

Ericson B, Moore S, Morrison B and Guzdial M (2015) Usability and Usage of Interactive Features in an Online Ebook for CS Teachers. *Proceedings of the Workshop in Primary and Secondary Computing Education on ZZZ - WiPSCE '15*. New York, New York, USA: ACM Press, 111–120. Available at: http://dl.acm.org/citation.cfm?doid=2818314.2818335 (accessed 02/09/17).

Ernie Giangrande J (2007) CS1 programming language options. *J. Comput. Small Coll.* 22(3): 153–

160.

Fagin BS (2002) Quantitative Analysis of the Effects of Robots on Introductory Computer Science Education. *ACM Journal of Educational Resources in Computing* 2(4): 1–18. Available at: http://xphileprof.com/professional/publications and presentations/2002_12 jeric.pdf (accessed 13/03/17).

Felder RM (1996) Matters of Style. *ASEE Prism* 6(4): 18–23.

Felder RM and Brent R (2005) Understanding Student Differences. *Journal of Engineering Education* 94(1): 57–72.

Felder RM and Silverman LK (1988) Learning and teaching styles in engineering education. *Engineering education* 78(7): 674–681.

Felder RM and Soloman B (1997) Index of Learning Style Questionnaire. . Available at: http://www.engr.ncsu.edu/learningstyles/ilsweb.html.

Felder RM and Spurlin J (2005) Applications, reliability and validity of the index of learning styles. *International Journal of Engineering Education* 21(1): 103–112.

Fenwick JB, Kurtz BL, Meznar P, Phillips R and Weidner A (2013) Developing a highly interactive ebook for CS instruction. *Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13*. New York, New York, USA: ACM Press, 135. Available at: http://dl.acm.org/citation.cfm?doid=2445196.2445241 (accessed 01/03/17).

Fidge C and Teague D (2009) Losing their marbles: syntax-free programming for assessing problem-solving skills. *Proceedings of the Eleventh Australasian Conference on Computing Education - Volume 95*. Wellington, New Zealand: Australian Computer Society, Inc.

Fitzgerald R and Fuller L (1982) I Hear You Knocking But You Can't Come In. *Sociological Methods & Research*. SAGE PUBLICATIONS 11(1): 3–32. Available at: http://journals.sagepub.com/doi/10.1177/004912418201100101 (accessed 12/09/17).

Fitzgerald S, Lewandowski G, Mccauley R, Murphy L, Simon B, Thomas L and Zander C (2008) Debugging : finding , fixing and flailing , a multi - institutional study of novice debuggers. .

Franzoni AL and Assar S (2009) Student Learning Styles Adaptation Method Based on Teaching Strategies and Electronic Media. *Educational Technology & Society* 12(4): 15–29.

Furnham A (1986) Response bias, social desirability and dissimulation. *Personality and Individual Differences* 7(3): 385–400. Available at: http://linkinghub.elsevier.com/retrieve/pii/0191886986900140 (accessed 12/09/17).

Goldman K, Gross P, Heeren C, Herman G, Kaczmarczyk L, Loui MC and Zilles C (2008) Identifying important and difficult concepts in introductory computing courses using a delphi process. *Proceedings of the 39th SIGCSE technical symposium on Computer science education - SIGCSE '08*. New York, New York, USA: ACM Press, 256. Available at: http://portal.acm.org/citation.cfm?doid=1352135.1352226 (accessed 25/10/16).

Gomes AJ, Santos AN and Mendes AJ (2012) A study on students' behaviours and attitudes towards learning to program. *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education - ITiCSE '12*. New York, New York, USA: ACM Press, 132. Available at: http://dl.acm.org/citation.cfm?doid=2325296.2325331 (accessed 29/03/18).

Gomez-Albarran M (2005) The Teaching and Learning of Programming: A Survey of Supporting Software Tools. *The Computer Journal* 48(2): 130–144. Available at: http://comjnl.oupjournals.org/cgi/doi/10.1093/comjnl/bxh080 (accessed 07/06/16).

Goodger D (2002) *reStructuredText*. . Available at: http://docutils.sourceforge.net/rst.html (accessed 31/05/17).

Graf S, Rita S and Leo T (2007) In-Depth Analysis of the Felder-Silverman Learning Style Dimensions. *Journal of Research on Technology in Education*.

Graham S (2010) *Skulpt*. . Available at: http://www.skulpt.org/ (accessed 01/06/17).

Guo P (2013) *Visualize Python, Java, JavaScript, TypeScript, and Ruby code execution*. . Available at: http://www.pythontutor.com/visualize.html#mode=edit (accessed 01/06/17).

Guzdial M (2003) Programming Environments for Novices. .

Guzidial M (2016) *CS Principles: Big Ideas in Programming — Runestone Interactive Overview*. . Available at: http://interactivepython.org/runestone/static/StudentCSP/index.html (accessed 31/05/17).

Haden P, Fincher S and Petre M (2004) Kent Academic Repository Versions of research Citation for published version. . Available at: http://kar.kent.ac.uk/14131/ (accessed 30/01/17).

Hamilton S, Hamilton M and Simon (2008) *Mental models, consistency and programming aptitude. Proceedings of the tenth conference on Australasian computing education - Volume 78*. Australian Computer Society, Inc. Available at: https://dl.acm.org/citation.cfm?id=1379253 (accessed 31/03/18).

Hansen S and Eddy E (2007) Engagement and frustration in programming projects. *Proceedings of the 38th SIGCSE technical symposium on Computer science education*. Covington, Kentucky, USA: ACM.

Haverbeke M (2011) *CodeMirror*. . Available at: http://codemirror.net/ (accessed 01/06/17).

Heliotis J and Zanibbi R (2011) Moving away from programming and towards computer science in the CS first year. *J. Comput. Small Coll.* 26(3): 115–125.

Hermann N (1982) *Hermann Brain Dominance Instrument*. Applied Services.

Higher Education Student Data (2017) *What do HE students study? | HESA*. . Available at: https://www.hesa.ac.uk/data-and-analysis/students/what-study (accessed 29/03/18).

Holland S, Griffiths R and Woodman M (1997) Avoiding object misconceptions. *SIGCSE Bull.* 29(1): 131–134.

Hristova M, Misra A, Rutter M and Mercuri R (2003) Identifying and Correcting Java Programming Errors for Introductory Computer Science Students. . Available at: http://notablesoftware.com/Papers/SIGCSEfin162.pdf (accessed 03/06/17).

Kaczmarczyk LC, Petrick ER, East JP and Herman GL (2010a) Identifying student misconceptions of programming. *Proceedings of the 41st ACM technical symposium on Computer science education - SIGCSE '10*. New York, New York, USA: ACM Press, 107. Available at: http://portal.acm.org/citation.cfm?doid=1734263.1734299 (accessed 25/10/16).

Kaczmarczyk LC, Petrick ER, East JP and Herman GL (2010b) Identifying student misconceptions of programming. *Proceedings of the 41st ACM technical symposium on Computer science education*. Milwaukee, Wisconsin, USA: ACM.

Karat C-M, Campbell R and Fiegel T (1992) Comparison of empirical testing and walkthrough methods in user interface evaluation. *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '92*. New York, New York, USA: ACM Press, 397–404. Available at: http://portal.acm.org/citation.cfm?doid=142750.142873 (accessed 23/08/17).

Karoulis A and Athanasis (2006) Evaluating the LEGO--RoboLab interface with experts. *Computers in Entertainment*. ACM 4(2): 6. Available at: http://portal.acm.org/citation.cfm?doid=1129006.1129017 (accessed 13/02/17).

Kawash J (2012) Engaging students by intertwining puzzle-based and problem-based learning. *Proceedings of the 13th annual conference on Information technology education*. Calgary, Alberta, Canada: ACM.

Keefe JW (1979) *Learning style: An overview." Student learning styles: Diagnosing and prescribing programs*. .

Khalife JT (2006) Threshold for the introduction of programming: Providing learners with a simple computer model. *Information Technology Interfaces, 2006. 28th International Conference on*. IEEE, 71–76.

Kolb DA (1984) *Experiential learning: Experience as the source of learning and development*. Prentice-Hall Englewood Cliffs, NJ.

Kölling M and Michael (2010) The Greenfoot Programming Environment. *ACM Transactions on Computing Education*. ACM 10(4): 1–21. Available at: http://portal.acm.org/citation.cfm?doid=1868358.1868361 (accessed 13/02/17).

Koohang A and Harman K (2007) *Learning objects and instructional design*. Informing Science Press. Available at: https://books.google.co.uk/books?hl=en&lr=&id=DOZFrbLt1CUC&oi=fnd&pg=PR9&dq=learning+objects+and+instructional+design&ots=QogskK2az4&sig=yn4UGHJs2M9wZNWLCBGwir9XpQo#v=onepage&q=learning objects and instructional design&f=false (accessed 02/06/17).

Korhonen A, Ross R, Shaffer CA, Naps T, Boisvert C, Crescenzi P, Karavirta V, Mannila L, Miller B, Morrison B and Rodger SH (2013) Requirements and design strategies for open source interactive computer science eBooks. *Proceedings of the ITiCSE working group reports conference on Innovation and technology in computer science education-working group reports - ITiCSE -WGR '13*. New York, New York, USA: ACM Press, 53–72. Available at: http://dl.acm.org/citation.cfm?doid=2543882.2543886 (accessed 01/03/17).

Lahtinen E, Ala-Mutka K, Järvinen H-M, Lahtinen E, Ala-Mutka K and Järvinen H-M (2005) A study of the difficulties of novice programmers. *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education - ITiCSE '05*. New York, New York, USA: ACM Press, 14. Available at: http://portal.acm.org/citation.cfm?doid=1067445.1067453 (accessed 07/06/16).

Layman L, Williams L and Slaten K (2007) Note to self: make assignments meaningful. *Proceedings of the 38th SIGCSE technical symposium on Computer science education*. Covington, Kentucky, USA: ACM.

Leping V, Lepp M, Niitsoo M, T E, nisson, Vene V and Villems A (2009) Python prevails. *Proceedings of the International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing*. Ruse, Bulgaria: ACM.

Levy RB-B and Ben-Ari M (2007) We work so hard and they don't use it: acceptance of software tools by teachers. *Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*. Dundee, Scotland: ACM.

Levy RB-B, Ben-Ari M, Levy RB-B and Ben-Ari M (2007) We work so hard and they don't use it. *Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education - ITiCSE '07*. New York, New York, USA: ACM Press, 246. Available at: http://portal.acm.org/citation.cfm?doid=1268784.1268856 (accessed 07/06/16).

Lewis C and Rieman J (1994) Task-Centered User Interface Design A Practical Introduction. . Available at: https://web.cs.dal.ca/~jamie/TCUID/covers-tcuid.pdf (accessed 23/08/17).

Lister R, Seppälä O, Simon B, Thomas L, Adams ES, Fitzgerald S, Fone W, Hamer J, Lindholm M, McCartney R, Moström JE, Sanders K, Lister R, Adams ES, Fitzgerald S, Fone W, Hamer J, Lindholm M, McCartney R, Moström JE, Sanders K, Seppälä O, Simon B and Thomas L (2004) A multi-national study of reading and tracing skills in novice programmers. *Working group reports from ITiCSE on Innovation and technology in computer science education - ITiCSE-WGR '04*. New York, New York, USA: ACM Press, 119. Available at: http://portal.acm.org/citation.cfm?doid=1044550.1041673 (accessed 07/06/16).

Malan DJ (2007) Podcasting Computer Science E-1. Association for Computing Machinery 389–393. Available at: https://dash.harvard.edu/bitstream/handle/1/2829931/1239630301-fp150-malan.pdf?sequence=2 (accessed 02/06/17).

Malan K and Halland K (2004) Examples that can do harm in learning programming. *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*. Vancouver, BC, CANADA: ACM.

Mannila L, Peltomäki M and Salakoski T (2006) What about a simple language? Analyzing the difficulties in learning to program. *Computer Science Education*. Routledge 16(3): 211–227. Available at: http://www.tandfonline.com/doi/abs/10.1080/08993400600912384 (accessed 17/11/16).

Mannila L and de Raadt M (2006) An objective comparison of languages for teaching introductory programming. *Proceedings of the 6th Baltic Sea conference on Computing education research Koli Calling 2006 - Baltic Sea '06*. New York, New York, USA: ACM Press, 32. Available at: http://portal.acm.org/citation.cfm?doid=1315803.1315811 (accessed 25/10/16).

Mannila L and Raadt M de (2006) An objective comparison of languages for teaching introductory programming. *Proceedings of the 6th Baltic Sea conference on Computing education research: Koli Calling 2006*. Uppsala, Sweden: ACM.

Maravić Čisar S, Pinter R and Radosav D (2011) Effectiveness of Program Visualization in Learning Java: a Case Study with Jeliot 3. *International Journal of Computers Communications & Control* 6(4): 668. Available at: http://univagora.ro/jour/index.php/ijccc/article/view/2094 (accessed 13/03/17).

McCracken M, Wilusz T, Almstrum V, Diaz D, Guzdial M, Hagan D, Kolikant YB-D, Laxer C, Thomas L and Utting I (2001) A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin*. ACM 33(4): 125. Available at: http://portal.acm.org/citation.cfm?doid=572139.572181 (accessed 07/06/16).

McGettrick A, Boyle R, Ibbett R, Lloyd J, Lovegrove G and Mander K (2005a) Grand Challenges in Computing: Education--A Summary. *The Computer Journal* 48(1): 42–48. Available at: http://comjnl.oupjournals.org/cgi/doi/10.1093/comjnl/bxh064 (accessed 07/06/16).

McGettrick A, Boyle R, Ibbett R, Lloyd J, Lovegrove G and Mander K (2005) Grand Challenges in Computing: Education—A Summary. *The Computer Journal* 48(1): 42–48. Available at: http://comjnl.oxfordjournals.org/content/48/1/42.abstract.

McGettrick A, Boyle R, Ibbett R, Lloyd J, Lovegrove G and Mander K (2005b) Grand Challenges in Computing: Education--A Summary. *The Computer Journal*. Oxford University Press 48(1): 42–48. Available at: http://comjnl.oupjournals.org/cgi/doi/10.1093/comjnl/bxh064 (accessed 25/10/16).

Merrick KE (2010) An empirical evaluation of puzzle-based learning as an interest approach for teaching introductory computer science. *Education, IEEE Transactions on* 53(4): 677–680.

Milbrandt G (1995) Using Problem Solving to Teach a Programming Language. *Learning & Leading with Technology* 23(2): 27–31.

Miller B (2015) *Fundamentals of Web Programming — Fundamentals of Web Programming*. . Available at: http://interactivepython.org/runestone/static/webfundamentals/index.html (accessed 31/05/17).

Milne I and Rowe G (2002a) Difficulties in Learning and Teaching Programming\&mdash;Views of Students and Tutors. *Education and Information Technologies* 7(1): 55–66.

Milne I and Rowe G (2002b) OGRE-3D Program Visualization for C++. *In Proceedings of the 3rd Annual LTSN-ICS Conference*.

Moons J and De Backer C (2013) The design and pilot evaluation of an interactive learning environment for introductory programming influenced by cognitive load theory and constructivism. *Computers & Education* 60(1): 368–384.

Murphy C, Kim E, Kaiser G, Cannon A, Murphy C, Kim E, Kaiser G and Cannon A (2008) Backstop. *ACM SIGCSE Bulletin*. ACM 40(1): 173. Available at: http://portal.acm.org/citation.cfm?doid=1352322.1352193 (accessed 07/06/16).

Naps TL, R G, ling, Almstrum V, Dann W, Fleischer R, Hundhausen C, Korhonen A, Malmi L, McNally M, Rodger S, J, Vel ngel and zquez I (2002) Exploring the role of visualization and engagement in computer science education. *SIGCSE Bull.* 35(2): 131–152.

Nielsen J (1995) How to Conduct a Heuristic Evaluation. Nielsen Norman Group. Available at: http://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/.

Nielsen J (1996) *Severity Ratings for Usability Problems: Article by Jakob Nielsen*. . Available at: https://www.nngroup.com/articles/how-to-rate-the-severity-of-usability-problems/ (accessed 25/08/17).

Nielsen J (2000) Why You Only Need to Test with 5 Users. Nielsen Norman Group. Available at: http://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/.

Nielsen J (2001) *Usability Metrics*. . Available at: https://www.nngroup.com/articles/usability-metrics/ (accessed 31/08/17).

Nielsen J (2006) *How Many Test Users in a Usability Study?* . Available at: https://www.nngroup.com/articles/how-many-test-users/ (accessed 30/08/17).

Nielsen J (2012) Usability 101: Introduction to Usability. *Jakob Nielsen's Alertbox*. Available at: http://www.nngroup.com/articles/usability-101-introduction-to-usability/.

Nielsen J (2014) *Task Scenarios for Usability Testing*. . Available at: https://www.nngroup.com/articles/task-scenarios-usability-testing/ (accessed 30/08/17).

Nielsen J and Molich R (1990) Heuristic evaluation of user interfaces. *Proceedings of the SIGCHI conference on Human factors in computing systems Empowering people - CHI '90*. New York, New York, USA: ACM Press, 249–256. Available at: http://portal.acm.org/citation.cfm?doid=97243.97281 (accessed 24/08/17).

Nikula U, Gotel O and Kasurinen J (2011) A Motivation Guided Holistic Rehabilitation of the First Programming Course. *ACM Trans . Comput . Educ . Article* 11(24).

Olson J (2004) Quick Methods: Checklists, Heuristic Evaluation, Cognitive Walkthrough. Evaluation of Systems and Services. Power Point Presentation.

Özmen B and Altun A (2014) Undergraduate Students' Experiences in Programming: Difficulties and Obstacles. *Turkish Online Journal of Qualitative Inquiry* 5(3): 1–27. Available at: http://dergipark.gov.tr/doi/10.17569/tojqi.20328 (accessed 29/03/18).

Palumbo DB (1990) Programming Language/Problem-Solving Research: A Review of Relevant Issues. *Review of Educational Research*. SAGE Publications 60(1): 65–89. Available at: http://rer.sagepub.com/cgi/doi/10.3102/00346543060001065 (accessed 17/11/16).

Paredes P and Rodriguez P (2004) A MIXED APPROACH TO MODELLING LEARNING STYLES IN ADAPTIVE EDUCATIONAL HYPERMEDIA. .

Parsons D and Haden P (2006) Parson's programming puzzles: a fun and effective learning tool for first programming courses. *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52*. Hobart, Australia: Australian Computer Society, Inc.

Pashler H, McDaniel M, Rohrer D and Bjork R (2008) Learning Styles: Concepts and Evidence. *Psychological Science in the Public Interest*. Sage Publications, Inc.Association for Psychological Science, 105–119. Available at: http://www.jstor.org/stable/20697325 (accessed 31/03/18).

Pearrow M and Mark (2000) *Web site usability handbook*. Charles River Media, Inc. Available at: http://dl.acm.org/citation.cfm?id=518262 (accessed 24/08/17).

Pears A, Seidman S, Malmi L, Mannila L, Adams E, Bennedsen J, Devlin M, Paterson J, Pears A, Seidman S, Malmi L, Mannila L, Adams E, Bennedsen J, Devlin M and Paterson J (2007) A survey of literature on the teaching of introductory programming. *ACM SIGCSE Bulletin*. ACM 39(4): 204. Available at: http://portal.acm.org/citation.cfm?doid=1345375.1345441 (accessed 07/06/16).

Pei Fen Chong, Yan-Peng Lim and Siew Woei Ling (2008) E-book design preferences: A case study. *2008 International Symposium on Information Technology*. IEEE, 1–8. Available at: http://ieeexplore.ieee.org/document/4631538/ (accessed 02/09/17).

Peña C-I, Marzo J-L and De La Rosa J-L (2002) Intelligent Agents in a Teaching and Learning Environment on the Web. .

Perrenet J and Kaasenbrood E (2006) Levels of abstraction in students' understanding of the concept of algorithm: the qualitative perspective. *Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*. Bologna, Italy: ACM.

Pollock L and Harvey T (2011a) Combining multiple pedagogies to boost learning and enthusiasm. *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education - ITiCSE '11*. New York, New York, USA: ACM Press, 258. Available at: http://portal.acm.org/citation.cfm?doid=1999747.1999820 (accessed 07/06/16).

Pollock L and Harvey T (2011b) Combining multiple pedagogies to boost learning and enthusiasm. *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*. Darmstadt, Germany: ACM.

Potatoes H (2001) *Hot Potatoes Home Page*. . Available at: https://hotpot.uvic.ca/ (accessed 01/06/17).

Preece J, Rogers Y, Sharp H, Benyon D, Holland S and Carey T (1994) *Human-computer interaction*. Addison-Wesley Pub. Co. Available at: http://dl.acm.org/citation.cfm?id=561701 (accessed 23/08/17).

Pritchard D and Vasiga T (2013) CS circles. *Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13*. New York, New York, USA: ACM Press, 591. Available at: http://dl.acm.org/citation.cfm?doid=2445196.2445370 (accessed 02/09/17).

de Raadt M, Watson R and Toleman M (2002) Language trends in introductory programming courses. Informing Science Institute.

de Raadt M, Watson R and Toleman M (2004) Introductory programming: what's happening today and will there be any students to teach tomorrow? *Proceedings of the Sixth Australasian Conference on Computing Education - Volume 30*. Australian Computer Society, Inc., 277–282.

Radenski A (2006) 'Python first': a lab-based digital introduction to computer science. *SIGCSE Bull.* 38(3): 197–201.

Ramalingam V, Labelle D and Wiedenbeck S (2004) Self-Efficacy and Mental Models in Learning to Program. *ITiCSE '04 Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, 171–175.

Randolph and J. J (2007) Computer science education research at the crossroads: a methodological review of computer science education research, 2000--2005. Utah State University.

Redmiles DF (1993) Reducing the Variability of Programmers' Performance Through Explained Examples Reducing the Variability of Programmers' Performance Through Explained Examples. . Available at: https://www.researchgate.net/profile/David_Redmiles/publication/2607698_Reducing_the_Variability_of_Programmers'_Performance_Through_Explained_Examples/links/55eb7fba08ae65b6389dec1e.pdf (accessed 06/09/17).

Reis HT and Judd CM (2000) *Handbook of research methods in social and personality psychology*. . Available at: https://books.google.co.uk/books?hl=en&lr=&id=NRWTAgAAQBAJ&oi=fnd&pg=PA11&dq=Research+Design+and+Issues+of+Validity&ots=3Hh8wtTqnK&sig=2NWrIe7D98bHSBZ1FNKu8ODkgJA#v=onepage&q=Research Design and Issues of Validity&f=false (accessed 12/09/17).

Robins A (2010) Learning edge momentum: A new account of outcomes in CS1. *Computer Science Education* 20(1). Available at: http://www.informaworld.com/smpp/ (accessed 30/01/17).

Robins A, Rountree J and Rountree N (2003) Learning and Teaching Programming: A Review and Discussion. *Computer Science Education* 13(2): 137–172. Available at: http://www.tandfonline.com/doi/abs/10.1076/csed.13.2.137.14200 (accessed 07/06/16).

Rogers Y, Sharp H and Preece J (2011) *Interaction design : beyond human-computer interaction*. Wiley. Available at: https://books.google.co.uk/books?hl=en&lr=&id=b-v_6BeCwwQC&oi=fnd&pg=PR11&dq=Interaction+Design:+beyond+human&ots=QJjK48FKza&sig=_ws63lhiR9ELtLLVXlly-v1Mo-w#v=onepage&q=Interaction Design%3A beyond human&f=false (accessed 24/08/17).

Ross JM (2002) Guiding students through programming puzzles: value and examples of Java game assignments. *SIGCSE Bull.* 34(4): 94–98.

Rowe G and Thorburn G (2000) VINCE-an on-line tutorial tool for teaching introductory programming. *British Journal of Educational Technology*. Blackwell Publishers Ltd 31(4): 359–369. Available at: http://doi.wiley.com/10.1111/1467-8535.00168 (accessed 06/09/17).

Rubin J and Chisnell D (1994) Handbook of Usability Testing How to Plan, Design, and Conduct Effective Tests Handbook of Usability Testing, Second Edition: How to Plan, Design, and Conduct Effective Tests. . Available at: http://ccftp.scu.edu.cn:8090/Download/efa2417b-08ba-438a-b814-92db3dde0eb6.pdf (accessed 30/08/17).

Runestone (2012) *Runestone Interactive Documentation — Runestone Interactive 1.0 documentation*. . Available at: http://runestoneinteractive.org/build/html/index.html (accessed 31/05/17).

Sanders ID and Langford S (2008) Students’ perceptions of python as a first programming language at wits. *SIGCSE Bull.* 40(3): 365.

Sauro J (2012) *MeasuringU: 10 Things To Know About The Single Ease Question (SEQ)*. . Available at: https://measuringu.com/seq10/ (accessed 31/08/17).

Sauro J (2013) *MeasuringU: Best Practices for Using Statistics on Small Sample Sizes*. . Available at: https://measuringu.com/small-n/ (accessed 12/09/17).

Shah H and Kumar AN (2002) A tutoring system for parameter passing in programming languages. *Proceedings of the 7th annual conference on Innovation and technology in computer science education - ITiCSE’02*. New York, New York, USA: ACM Press, 170. Available at: http://portal.acm.org/citation.cfm?doid=544414.544464 (accessed 06/09/17).

Shang Y, Shi H and Chen S-S (2001) An Intelligent Distributed Environment for Active Learning. .

Shneiderman B (2004) Designing for fun. *interactions*. ACM 11(5): 48. Available at: http://portal.acm.org/citation.cfm?doid=1015530.1015552 (accessed 25/08/17).

Shuhidan SM, Hamilton M and D’Souza D (2011) Understanding novice programmer difficulties via guided learning. *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*. Darmstadt, Germany: ACM.

Shute VJ (1991) Who is Likely to Acquire Programming Skills? *Journal of Educational Computing Research*. SAGE Publications 7(1): 1–24. Available at: http://jec.sagepub.com/lookup/doi/10.2190/VQJD-T1YD-5WVB-RYPJ (accessed 25/10/16).

Sorva J (2010) Reflections on threshold concepts in computer programming and beyond. *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*. Koli, Finland: ACM.

Sorva J, Karavirta V and Malmi L (2013) A Review of Generic Program Visualization Systems for Introductory Programming Education. *ACM Transactions on Computing Education*. ACM 13(4): 1–64. Available at: http://dl.acm.org/citation.cfm?doid=2543488.2490822 (accessed 07/06/16).

Stash N, Cristea A and De Bra P (2006) Adaptation to Learning Styles in E-Learning: Approach Evaluation. .

Stephenson B (2009) Using python and QuickDraw to foster student engagement in CS1. *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*. Orlando, Florida, USA: ACM.

Stern L, Markham S and Hanewald R (2005) You can lead a horse to water: how students really use pedagogical software. *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*. Caparica, Portugal: ACM.

Stern L, Markham S, Hanewald R, Stern L, Markham S and Hanewald R (2005) You can lead a horse to water. *ACM SIGCSE Bulletin*. ACM 37(3): 246. Available at: http://portal.acm.org/citation.cfm?doid=1151954.1067513 (accessed 07/06/16).

Stone D, Jarrett C, Woodroffe M and Minocha S (2005) *User interface design and evaluation*. Elsevier.

Sung K, Hillyard C, Angotti RL, Panitz MW, Goldstein DS and Nordlinger J (2011) Game-Themed Programming Assignment Modules: A Pathway for Gradual Integration of Gaming Context Into Existing Introductory Programming Courses. *IEEE Transactions on Education* 54(3): 416–427. Available at: http://ieeexplore.ieee.org/document/5559399/ (accessed 26/10/16).

Sweller J (1988) Cognitive Load During Problem Solving: Effects on Learning. *Cognitive Science*. Lawrence Erlbaum Associates, Inc. 12(2): 257–285. Available at: http://doi.wiley.com/10.1207/s15516709cog1202_4 (accessed 05/06/17).

Sykes ER (2007) Determining the effectiveness of the 3D Alice programming environment at the computer science I level. *Journal of Educational Computing Research* 36.2: 223–244. Available at: https://www.researchgate.net/profile/Edward_Sykes2/publication/240793117_Determining_the_Effectiveness_of_the_3D_Alice_Programming_Environment_at_the_Computer_Science_I_Level/links/0deec53b8b7011c7c9000000.pdf (accessed 13/03/17).

Tan P-H, Ting C-Y and Ling S-W (2009) Learning Difficulties in Programming Courses: Undergraduates' Perspective and Perception. *Computer Technology and Development, 2009. ICCTD'09. International Conference on*. IEEE, 42–46.

Tasneem S (2012) Critical thinking in an introductory programming course. *J. Comput. Sci. Coll.* 27(6): 81–83.

The Guardian (2017) *No evidence to back idea of learning styles | Letter | Education | The Guardian*. . Available at: https://www.theguardian.com/education/2017/mar/12/no-evidence-to-back-idea-of-learning-styles (accessed 12/09/17).

The Joint Task Force on Computing Curricula (2013) *Computer Science Curricula 2013. Association for Computing Machinery (ACM) IEEE Computer Society*. Available at: http://www.acm.org/education/CS2013-final-report.pdf (accessed 02/03/17).

Thomas L, Ratcliffe M, Woodbury J and Jarman E (2002a) Learning styles and performance in the introductory programming sequence. *Proceedings of the 33rd SIGCSE technical symposium on Computer science education - SIGCSE '02*. New York, New York, USA: ACM Press, 33. Available at: http://portal.acm.org/citation.cfm?doid=563340.563352 (accessed 25/10/16).

Thomas L, Ratcliffe M, Woodbury J and Jarman E (2002b) Learning styles and performance in the introductory programming sequence. *SIGCSE Bull.* 34(1): 33–37.

Tie H-H and Umar IN (2010) The Impact of Learning Styles and Instructional Methods on Students' Recall and Retention in Programming Education. .

TIOBE (2016) *TIOBE Index | TIOBE - The Software Quality Company*. . Available at: http://www.tiobe.com/tiobe-index/.

Universities UK (2015) *Patterns and trends in UK higher education – 2015*. . Available at: http://www.universitiesuk.ac.uk/facts-and-stats/data-and-analysis/Pages/patterns-and-trends-uk-higher-education-2015.aspx (accessed 07/06/16).

usability.gov (2013) *Usability Evaluation Basics*. Department of Health and Human Services. Available at: https://www.usability.gov/what-and-why/usability-evaluation.html (accessed 23/08/17).

usability.gov (2014) Running a Usability Test. Department of Health and Human Services. Available at: https://www.usability.gov/how-to-and-tools/methods/running-usability-tests.html (accessed 31/08/17).

Usability.gov (2013) Usability Testing. Department of Health and Human Services. Available at: https://www.usability.gov/how-to-and-tools/methods/usability-testing.html (accessed 30/08/17).

W3C (2003) *WAI Site Usability Testing Questions*. . Available at: https://www.w3.org/WAI/EO/Drafts/UCD/questions.html (accessed 30/08/17).

Willingham D (2016) *Daniel Willingham's Learning Styles FAQ - Daniel Willingham--Science &amp; Education*. . Available at: http://www.danielwillingham.com/learning-styles-faq.html (accessed 21/06/17).

Wilson BC and Shrock S (2001) Contributing to success in an introductory computer science course. *Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education - SIGCSE '01*. New York, New York, USA: ACM Press, 184–188. Available at: http://portal.acm.org/citation.cfm?doid=364447.364581 (accessed 25/10/16).

Wilson R, Landoni M and Gibb F (2009) Guidelines for Designing Electronic Books. *IETE Technical Review* 26(23): 213–222.

Winslow LE and E. L (1996) Programming pedagogy---a psychological overview. *ACM SIGCSE Bulletin*. ACM 28(3): 17–22. Available at: http://portal.acm.org/citation.cfm?doid=234867.234872 (accessed 25/10/16).

Zander C, Thomas L, Simon B, Murphy L, McCauley R, Hanks B, Fitzgerald S, Zander C, Thomas L, Simon B, Murphy L, McCauley R, Hanks B and Fitzgerald S (2009) Learning styles. *Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education - ITiCSE '09*. New York, New York, USA: ACM Press, 223. Available at: http://portal.acm.org/citation.cfm?doid=1562877.1562948 (accessed 25/10/16).

Zander C, Thomas L, Simon B, Murphy L, Ren, McCauley e, Hanks B and Fitzgerald S (2009) Learning styles: novices decide. *Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education*. Paris, France: ACM.

Zingaro D, Petersen A and Craig M (2012) Stepping up to integrative questions on CS1 exams. *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. Raleigh, North Carolina, USA: ACM.

Zualkernan IA, Allert J and Qadah GZ (2006) Learning styles of computer programming students: a Middle Eastern and American comparison. *Education, IEEE Transactions on* 49(4): 443–450.

Appendices

Appendix 1: Literature on Learning Environments

	Ease of use	Discoverability	Interactive functionalities	Customisation	Error handling	Automated assessment and feedback	Visualisation	Improving problem solving techniques	Minimise cognitive overload	Cover core programming concepts	Dependency
(Carbone et al 2000)								x	x		
(Franzoni and Assar 2009)			x	x			x		x		
(Kölling and Michael 2010)	x	x	x				x		x		
(Moons and De Backer 2013)			x				x		x	x	
(Clark and Mayer 2016)	x	x	x	x			x		x		
(Chong et al 2009b)		x									
(Fenwick et al 2013)		x	x		x	x	x				x

	Ease of use	**Discoverability**	**Interactive functionalities**	**Customisation**	**Error handling**	**Automated assessment and feedback**	**Visualisation**	**Improving problem solving techniques**	**Minimise cognitive overload**	**Cover core programming concepts**	**Dependency**
(Korhonen et al 2013)		x	x		x	x	x	x		x	x
(Ruth Wilson et al 2009)	x	x	x	x							x
(Guzdial 2003)	x		x		x	x	x	x			x

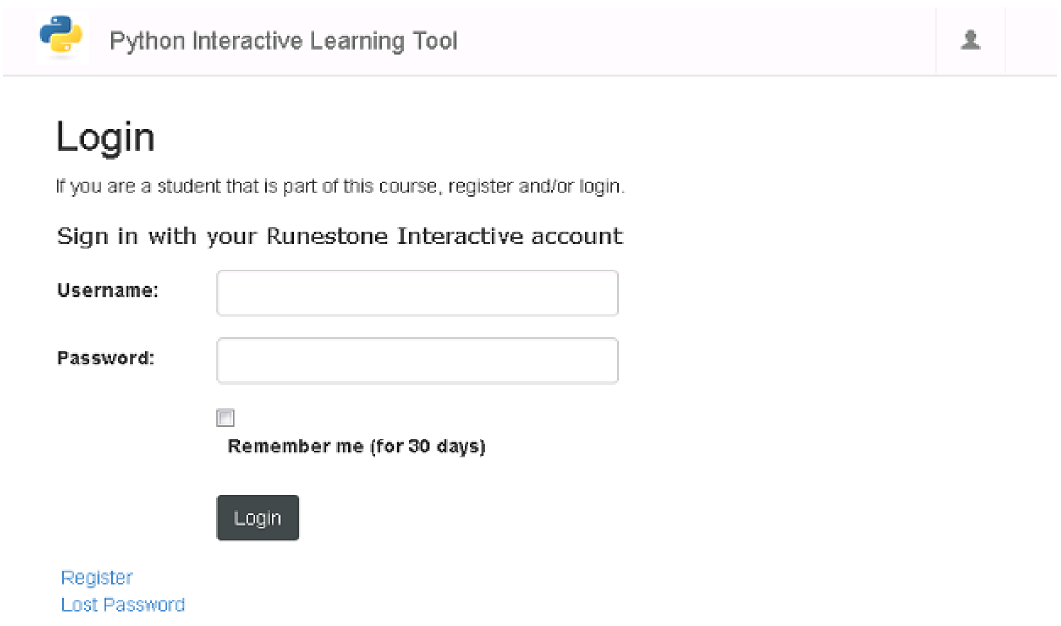
Appendix 2: Comparison of Novice Environments (Functional Features)

Functional Features		Alice	BlueJ	Jeliot3	RoboLab	RAPTOR
Programming style	functional					
	procedural			x	x	x
	object oriented		x			
	event based	x				
Programming concepts	variables	x	x	x	x	x
	conditional statements	x	x	x	x	x
	for loops	x	x	x	x	x
	while loops	x	x	x	x	x
	methods	x	x	x	x	x
	pre conditions and post conditions		x			x
Code representation	text	x	x			
	pictures		x	x		
	flowchart					x
	animation	x		x	x	
Code construction	typing in code		x	x		
	assembly and manipulation of graphic objects	x	x	x	x	x
Understanding programming support	backstories	x				
	debugging		x		x	
	examples					
	exercises					
Syntax error handling	direct syntax editing		x			
	user friendly error messages			x	x	
	limited selection of options					x

Appendix 3: PILeT Demo Session

Student interface

In order to use PILeT for the first time, students are required to create an account by registering with the website by clicking on the **register** link (Figure A.1). The registration page asks the student to submit a username, first name, last name, email, password, and course name. The course name is a unique field to prevent other users who are not affiliated with the course from accessing it and is provided by the instructor.



Python Interactive Learning Tool

Login

If you are a student that is part of this course, register and/or login.

Sign in with your Runestone Interactive account

Username:

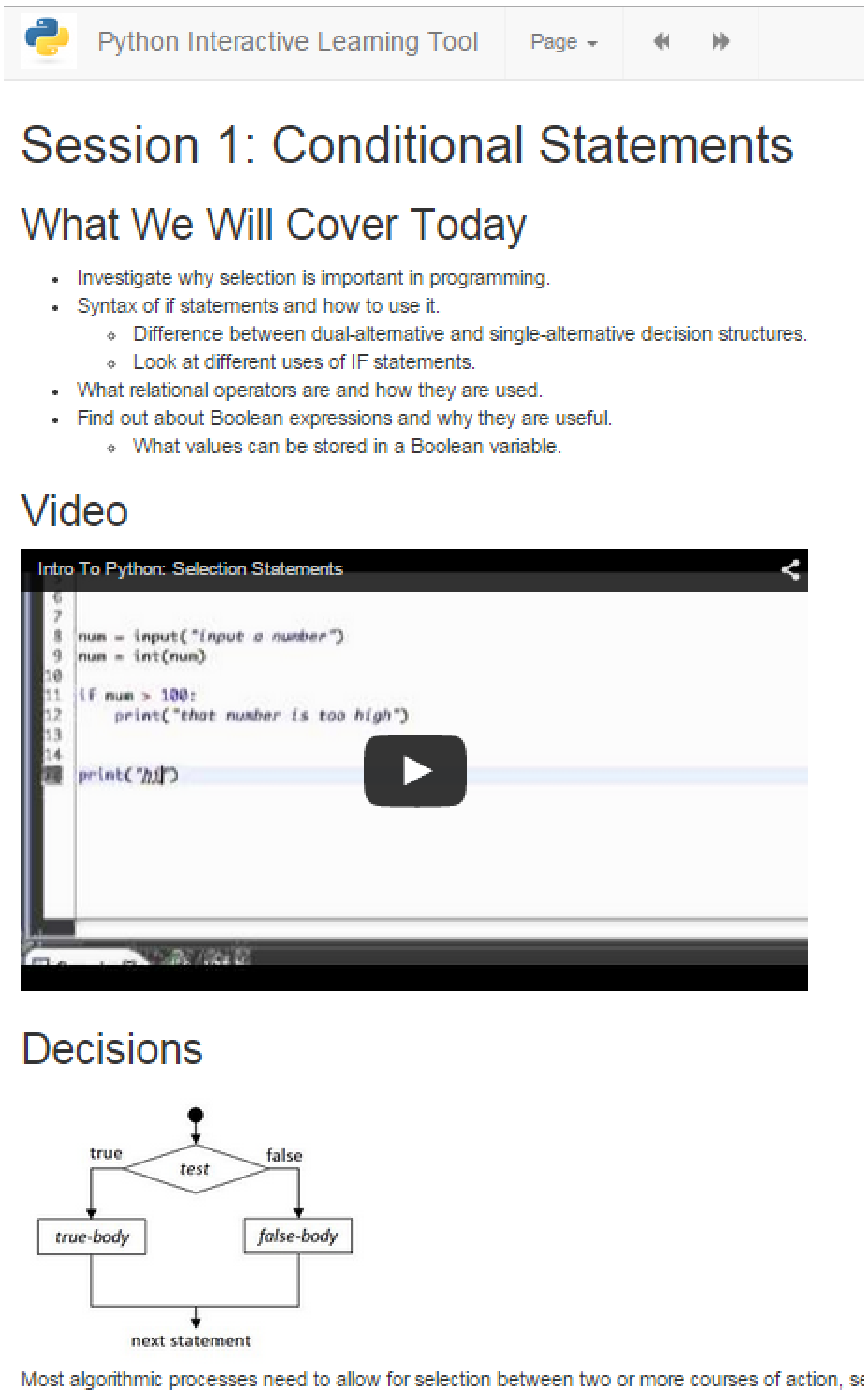
Password:

Remember me (for 30 days)

Figure A.1 PILeT Homepage

After logging in, a student land on the welcome page. This page compromises of two sections: a user manual at the top followed by a table of content. The user manual contains an introduction to PILeT along with information on how to use the different interactive features for new learners. The table of content lists five chapters each covering a Python concept (variables and expressions, selection statements, loops, functions and lists). Students have the freedom to jump between different chapters in a non-chronological order, however they are advised to complete each chapter consecutively in the user manual.

Each chapter starts with a small introduction of the lessons' objectives followed by embedded videos to explain the topic and provide examples (Figure A.2). Textual paragraphs are located below the videos together with images and figures as an alternative to the visual content in addition to source code visualisation. Next to follow, and throughout the chapter, are small executable examples, exercises, Parson's Programming Puzzles and multiple choice questions. These interactive features could be used on their own by students to learn programming in Python or to test their understanding of concepts. Multiple choice questions and Parson's Programming Puzzles are automatically assessed whilst coding exercises are not. Once students save their submissions by clicking on the **save** button, a copy of the source code is submitted to the instructor for marking and feedback. Finally built in PowerPoint slides are embedded at the end of the chapter for revision purposes along with a multiple choice question quiz.



Python Interactive Learning Tool   Page ▾

Session 1: Conditional Statements

What We Will Cover Today

- Investigate why selection is important in programming.
- Syntax of if statements and how to use it.
  - Difference between dual-alternative and single-alternative decision structures.
  - Look at different uses of IF statements.
- What relational operators are and how they are used.
- Find out about Boolean expressions and why they are useful.
  - What values can be stored in a Boolean variable.

Video

Intro To Python: Selection Statements

```
num = input("input a number")
num = int(num)

if num > 100:
    print("that number is too high")

print("hi")
```

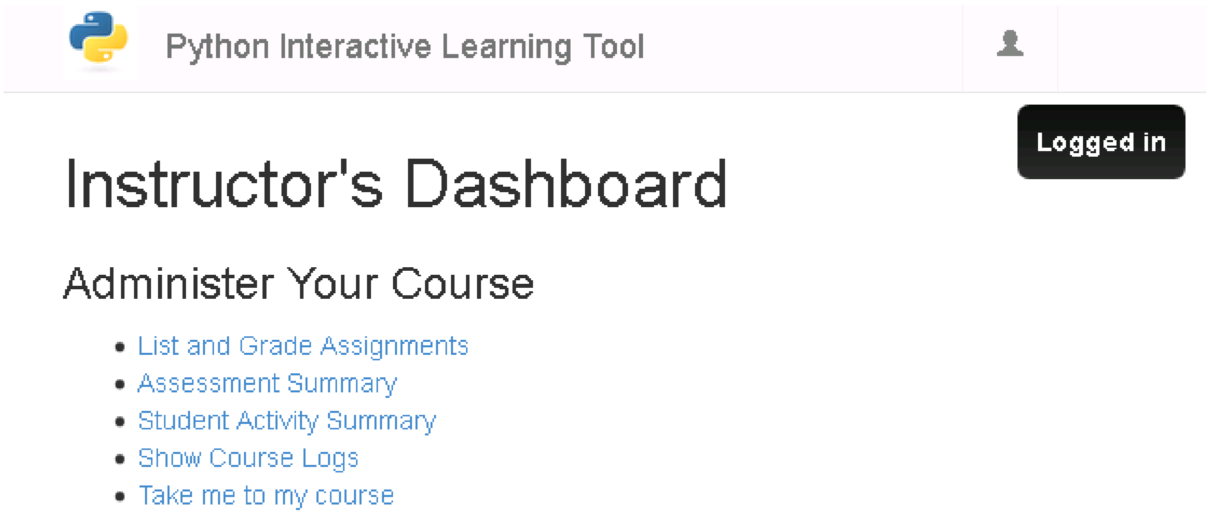
Decisions

Most algorithmic processes need to allow for selection between two or more courses of action, se

Figure A.2 Conditional Statements in PILeT

Instructor interface

PILeT contains a separate interface that can be accessed through the homepage. Once the instructor logs in with their credentials they land on the instructor's dashboard page (Figure A.3). The page consists of five links for the instructor to administer the Python course.


Python Interactive Learning Tool

Logged in

Instructor's Dashboard

Administer Your Course

- List and Grade Assignments
- Assessment Summary
- Student Activity Summary
- Show Course Logs
- Take me to my course

Figure A.3 Instructor's Dashboard in PILeT

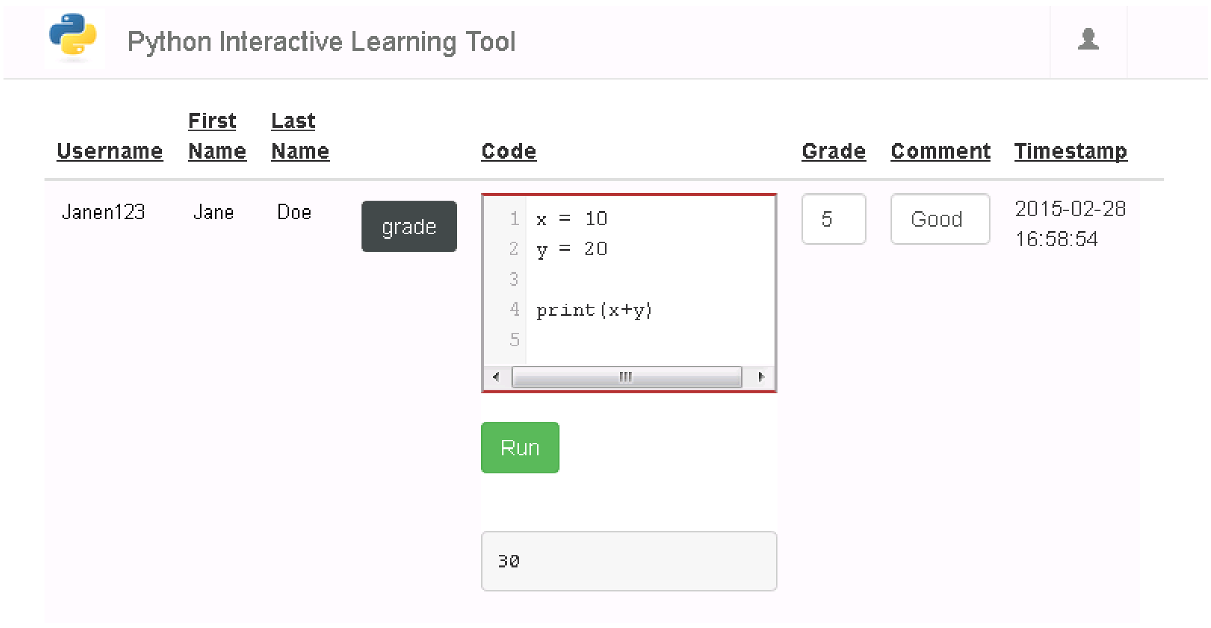
1. List and grade assignment:
This page contains a record of all of the assignments and exercises that need to be graded in PILeT (Figure A.4). Once an assignment is selected, the instructor is presented with a dashboard that lists all of the students who submitted the assignment including the username, first name, last name, timestamp (with date and time of submission), and source code. The instructor then grades the assignment and provides feedback.

2. Assessment summary:
This page shows a summary of all of the assessments submitted by the student ordered by last name including the grade.

3. Student activity summary:
Presents a page with a list of all the students registered with PILeT and the number of hours spent using the website.

4. Show course logs:
The page keeps of a record of all of the students' activities on PILeT. Every time a student clicks on an interactive feature, the action is logged in this page along with a timestamp.

5. Take me to my course:
Once clicked, the instructor lands on the welcome page with the table of content.



Username	First Name	Last Name		Code	Grade	Comment	Timestamp
Janen123	Jane	Doe	grade	x = 10 y = 20 print(x+y)	5	Good	2015-02-28 16:58:54

```
30
```

Figure A.4 Grading Assignments in PILeT

Appendix 4: Usability Heuristics

1	**Inform user of system status and offer feedback**	
Explanation: offer system feedback to users frequently, display the progress level of long tasks and provide notification messages when a process is completed		
Ease of fixing		**Evidence and comment:**
	0. Very easy	
	1. Easy	
	2. Medium	
	3. Hard	
Severity rating:		
	0. No problem	
	1. Cosmetic problem only	
	2. Minor usability problem	
	3. Major usability problem	
	4. Usability catastrophe	

2	**Speak in the user natural language**	
Explanation: use simple text to communicate with users, avoid technical jargon and use familiar terms		
Ease of fixing		**Evidence and comment:**
	0. Very easy	
	1. Easy	
	2. Medium	
	3. Hard	
Severity rating:		
	0. No problem	
	1. Cosmetic problem only	
	2. Minor usability problem	
	3. Major usability problem	
	4. Usability catastrophe	

3	Allow easy navigation and reversal of actions	
Explanation: make commands easily accessible, group related commands, and allow users to undo and redo actions		
Ease of fixing		**Evidence and comment:**
	0. Very easy	
	1. Easy	
	2. Medium	
	3. Hard	
Severity rating:		
	0. No problem	
	1. Cosmetic problem only	
	2. Minor usability problem	
	3. Major usability problem	
	4. Usability catastrophe	

4	Strive for consistency across the system	
Explanation: text format and graphics should be consistent within the system, this includes icons, texts labels and dialog boxes. Additionally, buttons must perform the same tasks throughout the system		
Ease of fixing		**Evidence and comment:**
	0. Very easy	
	1. Easy	
	2. Medium	
	3. Hard	
Severity rating:		
	0. No problem	
	1. Cosmetic problem only	
	2. Minor usability problem	
	3. Major usability problem	
	4. Usability catastrophe	

5	Design dialogue to yield closure and allow exists	
Explanation: actions should follow a sequence (beginning, middle end). Allow users to cancel or abort actions		
Ease of fixing		**Evidence and comment:**
	0. Very easy	
	1. Easy	
	2. Medium	
	3. Hard	
Severity rating:		
	0. No problem	
	1. Cosmetic problem only	
	2. Minor usability problem	
	3. Major usability problem	
	4. Usability catastrophe	

6	Prevent errors and offer simple error handling	
Explanation: offer users the option to confirm selection before proceeding, eliminate error prone situations and present errors in a simple language with suggested solutions		
Ease of fixing		**Evidence and comment:**
	0. Very easy	
	1. Easy	
	2. Medium	
	3. Hard	
Severity rating:		
	0. No problem	
	1. Cosmetic problem only	
	2. Minor usability problem	
	3. Major usability problem	
	4. Usability catastrophe	

7	Reduce cognitive load	
Explanation: employ recognition methods for system instructions, make menu options visible to users and use labels for buttons		
Ease of fixing		**Evidence and comment:**
	0. Very easy	
	1. Easy	
	2. Medium	
	3. Hard	
Severity rating:		
	0. No problem	
	1. Cosmetic problem only	
	2. Minor usability problem	
	3. Major usability problem	
	4. Usability catastrophe	

8	Provide shortcuts for repeated actions	
Explanation: reduce number of user interaction with interface by using shortcuts and macros for repetitive actions		
Ease of fixing		**Evidence and comment:**
	0. Very easy	
	1. Easy	
	2. Medium	
	3. Hard	
Severity rating:		
	0. No problem	
	1. Cosmetic problem only	
	2. Minor usability problem	
	3. Major usability problem	
	4. Usability catastrophe	

9	Use minimalistic design	
Explanation: keep information to minimum, use user friendly text for all user groups (e.g. users with dyslexia), employ colour and text size to highlight important ideas and use spacing between paragraphs to divide sections		
Ease of fixing		**Evidence and comment:**
	0. Very easy	
	1. Easy	
	2. Medium	
	3. Hard	
Severity rating:		
	0. No problem	
	1. Cosmetic problem only	
	2. Minor usability problem	
	3. Major usability problem	
	4. Usability catastrophe	

10	Offer help at all times	
Explanation: offer users assistance with all tasks when necessary and enable users to search for problem solutions		
Ease of fixing		**Evidence and comment:**
	0. Very easy	
	1. Easy	
	2. Medium	
	3. Hard	
Severity rating:		
	0. No problem	
	1. Cosmetic problem only	
	2. Minor usability problem	
	3. Major usability problem	
	4. Usability catastrophe	

Appendix 5: Usability Problems

List of problems

1. Once a profile is created, the user cannot access the profile page again to change credentials (for example password).

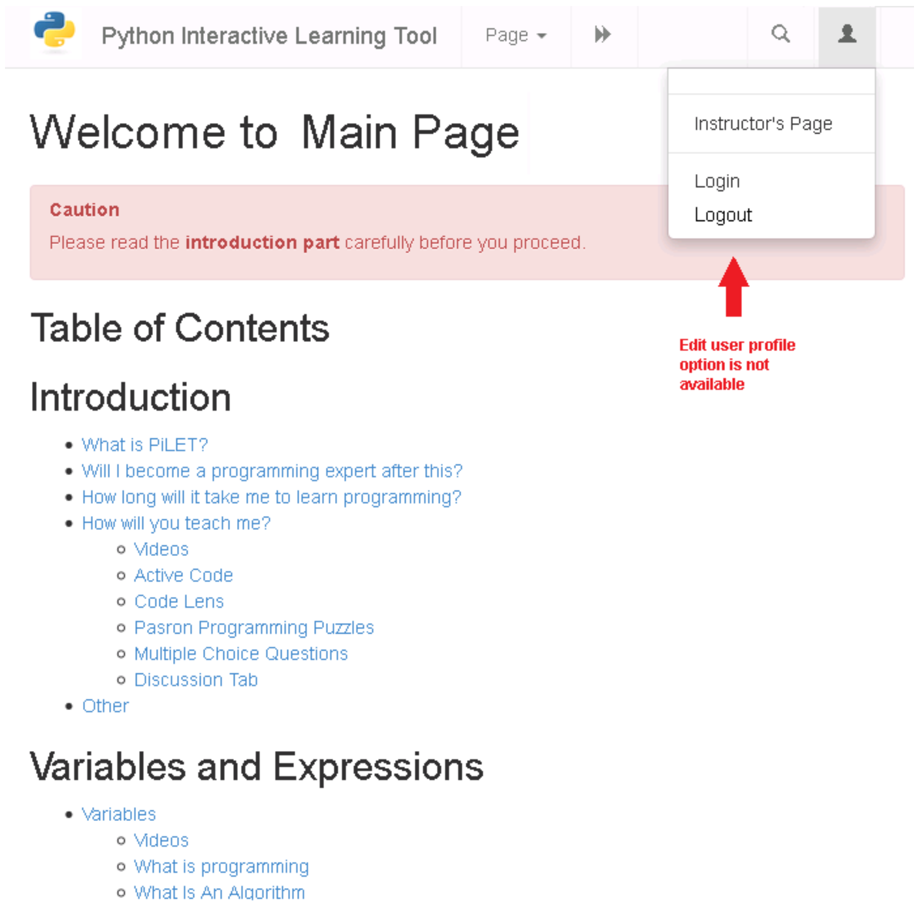
Number	Usability Problem	Usability Heuristic	Severity Rating	Ease of Fixing Rating
1	Once a profile is created, the user cannot access the profile page again to change credentials (for example password).	3. Allow easy navigation and reversal of actions.	4	3

Description:

In order to use PILeT for the first time, users are required to create an account by registering with the website. The registration page asks users to submit a username, first name, last name, email, password, and course name. Once the registration process is completed, users cannot edit or change any of the submitted information as that option does not exist. This problem violates usability heuristic 3 which states that users should easily be allowed to reverse their action.

Evidence:

Figure 5.1 illustrates PILeT main page. The page contains the table of contents which lists the programming concepts, the forward button that navigates the user automatically to the first chapter, the search button that locate a specific concept, and the user profile button at the top right corner. Once the button is clicked it does contain (edit user profile) as an option.


Python Interactive Learning Tool

Welcome to Main Page

Caution
Please read the **introduction part** carefully before you proceed.

Table of Contents

Introduction

- What is PiLET?
- Will I become a programming expert after this?
- How long will it take me to learn programming?
- How will you teach me?
  - Videos
  - Active Code
  - Code Lens
  - Pasron Programming Puzzles
  - Multiple Choice Questions
  - Discussion Tab
- Other

Variables and Expressions

- Variables
  - Videos
  - What is programming
  - What Is An Algorithm

Figure A.5 PILeT Main Page with Missing Option

Solution:

Whilst it was challenging the fix this usability issue as it requires data retrieval and manipulation, the problem was resolved by adding (edit user profile) option in the drop down menu and creating a (user profile) page. And while the page allows the user to change their first name, last name, email and password, it does not allow changing the username or the course name as this information is associated with any submissions made by the user (for example coding exercises and quizzes).

2. The link to (Loops) chapter is broken

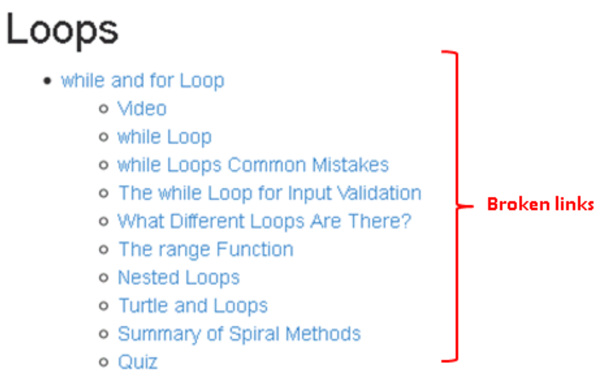
Number	Usability Problem	Usability Heuristic	Severity Rating	Ease of Fixing Rating
2	The link to (Loops) chapter and its subsequent sections is broken	4. Strive for consistency across the system	4	2

Description:

Whilst trying to access different Python concepts, all of the evaluators realised that the link that navigates the user to (Loops) chapter and its subsequent sections is broken (resulting in 404 Not Found Error in the browser). This error violates usability heuristic 4 which states that buttons and links should be consistent within the system, and must perform the same tasks throughout it. The severity rating of this problem is rated 4 (usability catastrophe) because the task cannot be performed by the user and as such the concept is inaccessible.

Evidence:

This problem happens once the user tries to click on the link.



Loops

- while and for Loop
  - Video
  - while Loop
  - while Loops Common Mistakes
  - The while Loop for Input Validation
  - What Different Loops Are There?
  - The range Function
  - Nested Loops
  - Turtle and Loops
  - Summary of Spiral Methods
  - Quiz

Figure A.6 Broken Links

Solution:

After investigating the problem for a while, it was realised that during the development stage of PILeT, there were two versions of the same concept, the old one named (Loop) and the other (Loops). The former was used as a mock page which was deleted later, however, after completion of the webpage (Loops), the old URL was not directing to the new page (Loops) but (Loop) instead. The error was fixed by changing the pathway to the page.

3. There are dead and unnecessary buttons

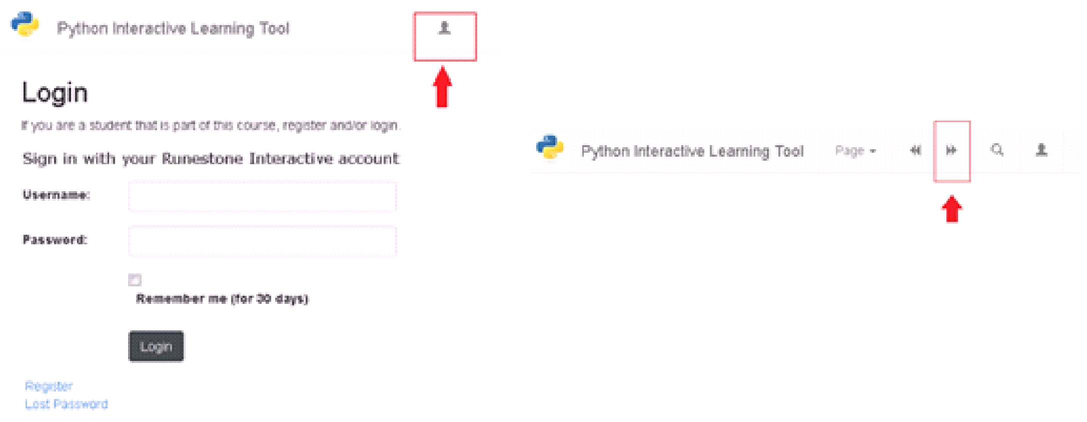
Number	Usability Problem	Usability Heuristic	Severity Rating	Ease of Fixing Rating
3	There are dead and unnecessary buttons	4. Strive for consistency across the system	3	1

Description

Four of the five evaluators (1, 2, 3, and 5) noticed that the user icon button in the homepage does not perform any actions whilst on other pages the user can click on it to log out. Additionally, there is an unnecessary forward button on (Lists) page which should be removed because the concept is the last chapter in the interactive tool. This problem also violates usability heuristic 4 and is rated as (usability catastrophe).

Evidence:

Below are two instances where the problem occurs.


Python Interactive Learning Tool
Login
Username:
Password:
Remember me (for 30 days)
Login
Register
Lost Password
Python Interactive Learning Tool
Page

If you are a student that is part of this course, register and/or login.

Sign in with your Runestone Interactive account

Figure A.7 Dead Buttons in Homepage and Final Page

Solution:

The problem was easily fixed easily by deleting the button on both pages.

4. Instead of having the user manual as a separate section, make it compulsory and at the beginning as most users are new users.

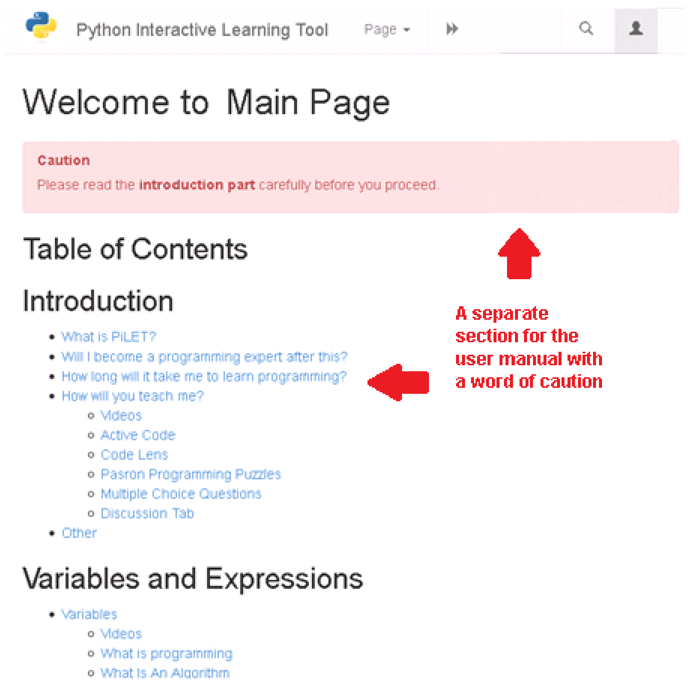
Number	**Usability Problem**	**Usability Heuristic**	**Severity Rating**	**Ease of Fixing Rating**
4	Instead of having the user manual as a separate section, make it compulsory and at the beginning as most users are new users.	5. Design dialogue to yield closure and allow exists 7. Reduce cognitive load	2	2

Description:

Evaluators 1 and 3, who have experience in online learning, felt that it was necessary for first time visitors of PILeT, who are likely to be without any previous programming experience, to read the user manual and try out the interactive features before learning about the concepts. This is especially true for using the embedded code compiler which requires users to type, run, save and load the code in addition to Codelens which visualises code execution.

Evidence:

As seen in Figure A.8, currently the user manual is a separate section that is not compulsory for user to read. Instead, a caution message was used to highlight its importance.


Python Interactive Learning Tool

Welcome to Main Page

Caution

Please read the **introduction part** carefully before you proceed.

Table of Contents

Introduction

- What is PiLET?
- Will I become a programming expert after this?
- How long will it take me to learn programming?
- How will you teach me?
  - Videos
  - Active Code
  - Code Lens
  - Pasron Programming Puzzles
  - Multiple Choice Questions
  - Discussion Tab
- Other

Variables and Expressions

- Variables
  - Videos
  - What is programming
  - What Is An Algorithm

Page
A separate section for the user manual with a word of caution

Figure A.8 User Manual in Separate Section

Solution:

After the registration step, all users land on the (user manual) page instead of the (main page). However, if the user has previous programming experience and is familiar with these features, an option to skip the manual is included.

5. The confirm password field box is not immediately under the password field box.

Number	Usability Problem	Usability Heuristic	Severity Rating	Ease of Fixing Rating
5	The confirm password field box is not immediately under the password field box.	4. Strive for consistency across the system	2	1

Description:

In the registration page, users are required to type in a password and confirm their selection by writing it again in the text field. However, both fields are separated by the (course name) text field.

Evidence:

Figure 5.5 shows the unnatural order of the text fields.

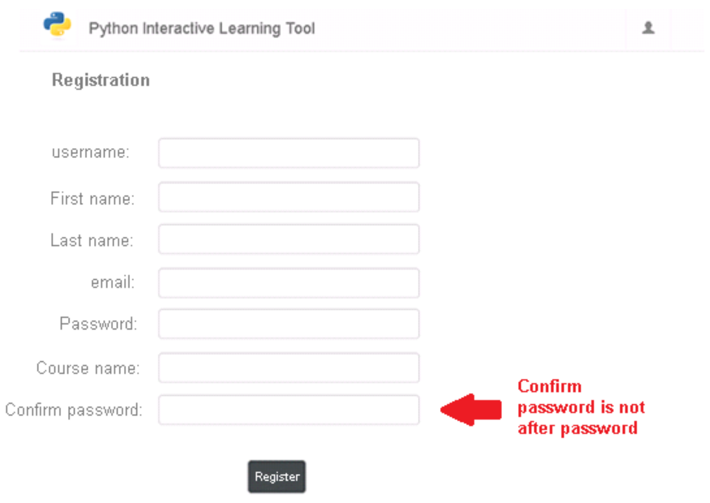


Figure A.9 Confirm Password Text Field Is Not After Password

Solution:

The (course name) text field was moved below (Confirm password).

6. There is no clear indication that the user is logged in

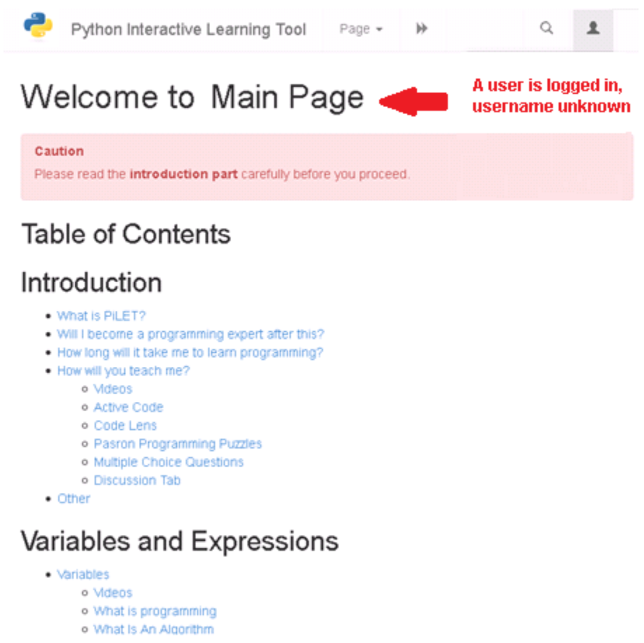
Number	Usability Problem	Usability Heuristic	Severity Rating	Ease of Fixing Rating
6	There is no clear indication that the user is logged in	1. Inform user of system status and offer feedback	2	1

Description:

Whilst it was not diagnosed as a major usability problem, the evaluators felt it was necessary to inform users of their status once they are logged in or registered as it is a common practice in UI (user interface) design that creates familiarity and friendliness with the website.

Evidence:

As Figure A.10 shows, a user is logged in to PILeT, however the user's identity is unknown.


Python Interactive Learning Tool
Page
A user is logged in, username unknown

Welcome to Main Page

Caution

Please read the **introduction part** carefully before you proceed.

Table of Contents

Introduction

- What is PiLET?
- Will I become a programming expert after this?
- How long will it take me to learn programming?
- How will you teach me?
  - Videos
  - Active Code
  - Code Lens
  - Pasron Programming Puzzles
  - Multiple Choice Questions
  - Discussion Tab
- Other

Variables and Expressions

- Variables
  - Videos
  - What is programming
  - What Is An Algorithm

Figure A.10 Unknown User Logged In

Solution:

The solution involved a simple interface fix; once the user successfully logs in, the username is displayed at the top of the page with a small welcome message as follows (Welcome *username*!).

7. The option to log in is visible to already logged in users

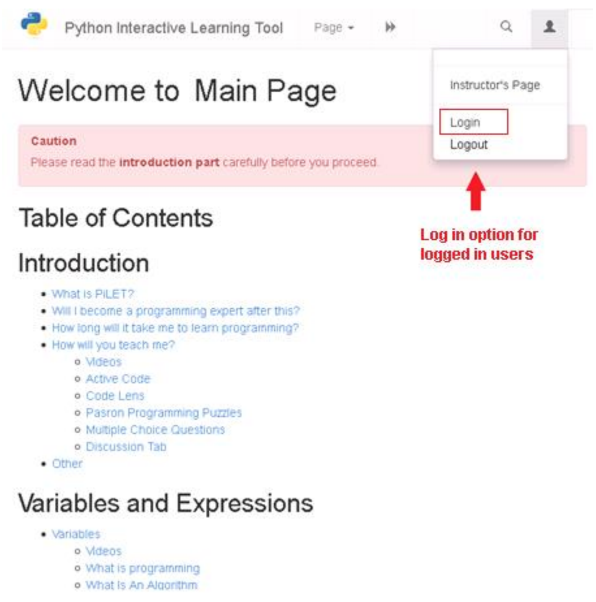
Number	Usability Problem	Usability Heuristic	Severity Rating	Ease of Fixing Rating
7	The option to log in is visible to already logged in users	1. Inform user of system status and offer feedback	2	1

Description:

This problem is in clear violation of usability heuristic 1: inform users of system status, as this option should not be provided to already logged in users.

Evidence:

A logged in user can select the log in option.


Python Interactive Learning Tool
Page
Instructor's Page
Login
Logout
Log in option for logged in users

Welcome to Main Page

Caution

Please read the **introduction part** carefully before you proceed.

Table of Contents

Introduction

- What is PiLET?
- Will I become a programming expert after this?
- How long will it take me to learn programming?
- How will you teach me?
  - Videos
  - Active Code
  - Code Lens
  - Pasron Programming Puzzles
  - Multiple Choice Questions
  - Discussion Tab
- Other

Variables and Expressions

- Variables
  - Videos
  - What is programming
  - What Is An Algorithm

Figure A.11 Log In Option For Logged In Users

Solution:

The option was removed from the drop down menu.

8. Users can see instructor's page option, it should be separate and at the beginning

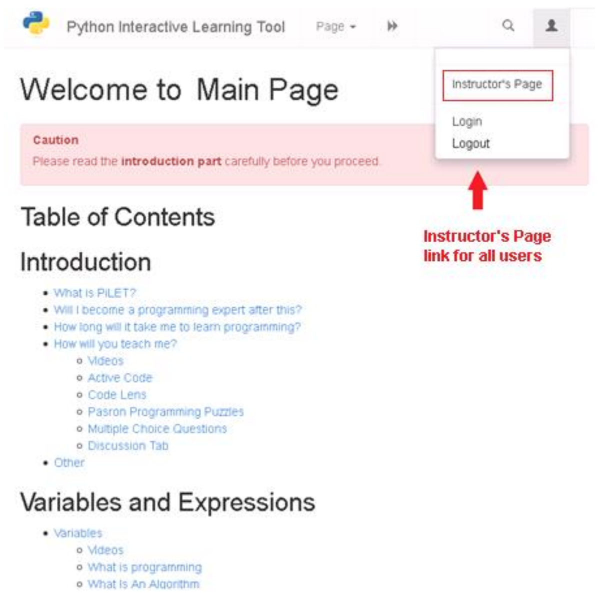
Number	Usability Problem	Usability Heuristic	Severity Rating	Ease of Fixing Rating
8	Users can see instructor's page option, it should be separate and at the beginning	3. Allow easy navigation and reversal of actions	2	1

Description:

Evaluators 3, 4 and 5 suggested that instead of the instructor logging in twice (once through the homepage and again using the link at PILeT main page), the process should be reduced to one step at the homepage, and remove that option from the students' interface.

Evidence:

A drop down menu is shown with instructor's page link in a student interface.


Python Interactive Learning Tool Page

Instructor's Page

Login

Logout

Welcome to Main Page

Caution

Please read the **introduction part** carefully before you proceed.

Table of Contents

Introduction

- What is PiLET?
- Will I become a programming expert after this?
- How long will it take me to learn programming?
- How will you teach me?
    - Videos
    - Active Code
    - Code Lens
    - Pasron Programming Puzzles
    - Multiple Choice Questions
    - Discussion Tab
- Other

Variables and Expressions

- Variables
    - Videos
    - What is programming
    - What Is An Algorithm

Figure A.12 Instructor's Page Link Is Shown To All Users

Solution:

This option is now only available in the instructor's interface and subsequently has been removed from the students' page.

9. There is no indication whether a user completed a chapter or progress level

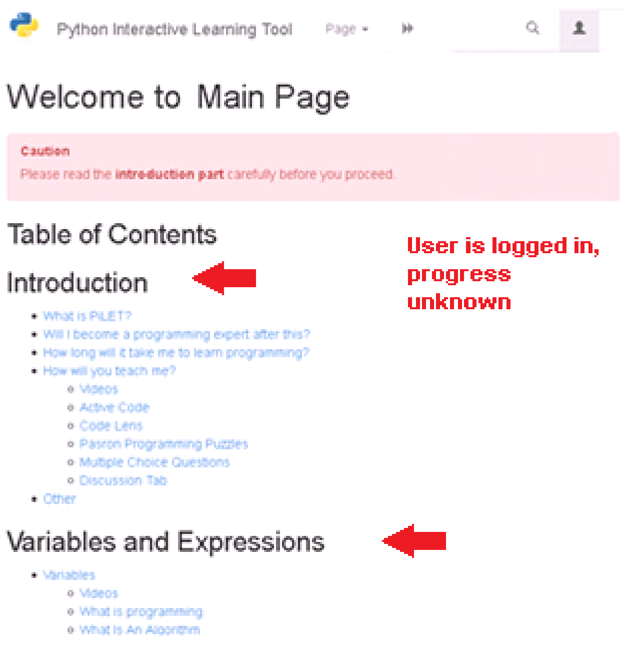
Number	Usability Problem	Usability Heuristic	Severity Rating	Ease of Fixing Rating
9	Users are unaware of their progress level or whether a chapter is completed	1. Inform user of system status and offer feedback	1	1

Description:

In most educational websites, users are informed of their progress level with the material by displaying a breakdown of their marks, the number of exercises and quizzes solved, or completion percentage out of a 100%. Currently, unless the user checks each chapter independently for the attempted exercises, only the instructor knows which exercises were completed. This problem is in breach of heuristic evaluation 1.

Evidence:

Figure A.13 displays PILeT main page. The progress level is nowhere to be seen.


Python Interactive Learning Tool

Page

Welcome to Main Page

Caution

Please read the **introduction part** carefully before you proceed.

Table of Contents

Introduction

- What is PILET?
- Will I become a programming expert after this?
- How long will it take me to learn programming?
- How will you teach me?
  - Videos
  - Active Code
  - Code Lens
  - Parson Programming Puzzles
  - Multiple Choice Questions
  - Discussion Tab
- Other

Variables and Expressions

- Variables
  - Videos
  - What is programming
  - What Is An Algorithm

Figure A.13 Progress Level Unknown For Logged In Users

Solution:

A shaded check mark has been added next to each chapter heading. Once the chapter is completed by the user (completion is this case meaning all of the exercises have been completed regardless of the correctness of the answer), the check mark turns green.

10. Some tooltips have the same text as the button making them redundant

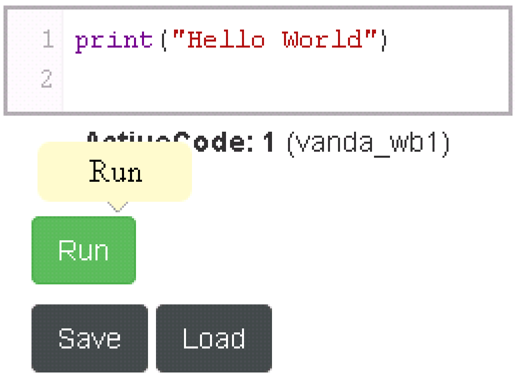
Number	Usability Problem	Usability Heuristic	Severity Rating	Ease of Fixing Rating
10	Some tooltips have the same text as the button making them redundant	9. Use minimalistic design	1	0

Description:

The evaluators noticed that a couple of the buttons contain unnecessary tooltips as the tip is the exact same text as the buttons. The buttons are: run, save, load, check me and reset.

Evidence:

Figure A.14 shows an example of the usability issue described; the text in the run button is exactly the same as the tooltip.



```
print("Hello World")
```

Figure A.14 Button Text Same As Tooltip

Solution:

All of buttons in PILeT were inspected again and as a result, the unnecessary tooltips were removed.

11. Some concepts had a relatively larger number of material and exercises.

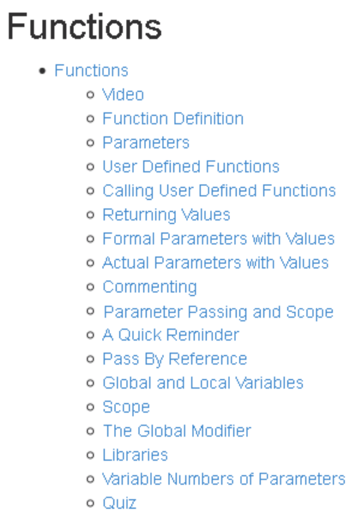
Number	Usability Problem	Usability Heuristic	Severity Rating	Ease of Fixing Rating
11	Some concepts had a relatively larger number of material and exercises	9. Use minimalistic design:	0	0

Description:

Both concepts (variables and expressions) and (functions) contained a large number of content for the user to go through which might be challenging for novice learners.

Evidence:

The concept function contains 18 subsections.


Functions

- Functions
  - Video
  - Function Definition
  - Parameters
  - User Defined Functions
  - Calling User Defined Functions
  - Returning Values
  - Formal Parameters with Values
  - Actual Parameters with Values
  - Commenting
  - Parameter Passing and Scope
  - A Quick Reminder
  - Pass By Reference
  - Global and Local Variables
  - Scope
  - The Global Modifier
  - Libraries
  - Variable Numbers of Parameters
  - Quiz

Figure A.15 The Function Concept in PILeT

Solution:

In order to keep the information to a minimum, without compromising the quality of the content, the concepts have been divided into a manageable number of sections and subsections. Adequate spacing between paragraphs was used to divide sections, in addition to using colours to highlight important ideas.

12. The word "caution" is a bit alarming

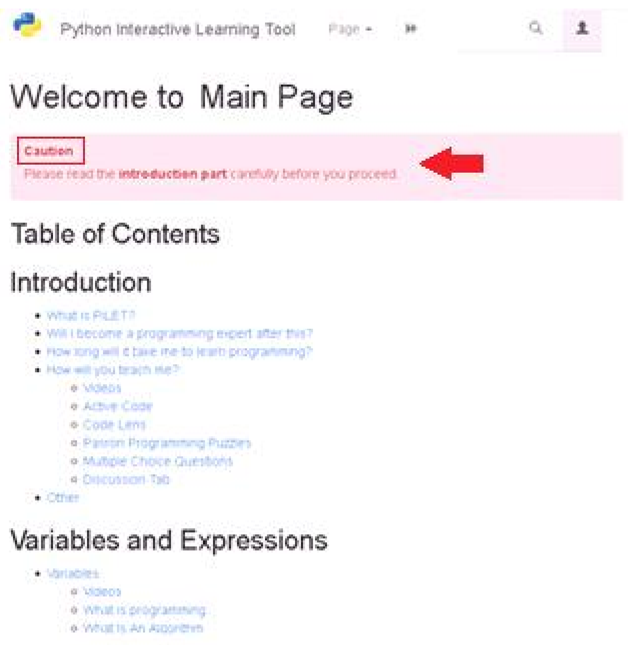
Number	Usability Problem	Usability Heuristic	Severity Rating	Ease of Fixing Rating
12	The word "caution" is a bit alarming	2. Speak in the user natural language	0	0

Description:

Evaluator 3 commented that the word "caution" is a bit alarming. Suggest replacing it with "attention"

Evidence:

Figure A.16 shows the "caution" on PILeT main page.


Python Interactive Learning Tool

Page

Welcome to Main Page

Caution

Please read the **introduction part** carefully before you proceed.

Table of Contents

Introduction

- What is PILET?
- Will I become a programming expert after this?
- How long will it take me to learn programming?
- How will you teach me?
    - Videos
    - Active Code
    - Code Lens
    - Parson Programming Puzzles
    - Multiple Choice Questions
    - Discussion Tab
- Other

Variables and Expressions

- Variables
    - Videos
    - What is programming
    - What Is An Algorithm

Figure A.16 Caution Message In PILeT Main Page

Solution:

As a result of usability problem 4, the "caution message" was removed.

Appendix 6: Moderator Script

Welcome and Purpose

Thank you for agreeing to take part in PILeT evaluation. Today I am asking you to serve as an evaluator of this website and to complete a set of scenarios. Our goal is to see how easy or difficult you find the site to use and your overall impression of the interface. I will audio record your reactions and opinions; so, I may ask you to clarify statements that you make from time to time. I will take notes and observe your interaction with the site. If you agree to take part, please sign the consent form and answer the questionnaire at the end of the evaluation.

During this session you will perform tasked scenarios whilst thinking out loud to understand your thought process, I will not be able to offer any suggestions or hints. There may be times, however, when I will ask you to explain why you said or did something. After you complete a scenario, please notify me so I can ask you a question about the task. You also will be asked a series of questions about your experience at the end of this session.

Things to Keep in Mind

- What we are doing today is all about how easy we have made it for people to use the site.
- There is no right or wrong answer. If you have any questions, comments or areas of confusion while you are working, please let me know.
- If you ever feel that you are lost or cannot complete a scenario with the information that you have been given, please let me know. I'll ask you what you might do in a real-world setting and then either put you on the right track or move you on to the next scenario.
- We will be recording this session for reference if needed. Your name will not be associated or reported with data or findings from this evaluation. Please fill the consent form.
- Finally, as you use the site, please do so as you would at home or your office. I do ask that when looking for information, you do so as quickly and as accurately as you can.

Do you have any questions before we begin?

Appendix 7: Consent Form

Study Title:

PILeT: Usability Study

Researcher Details:
Bedour Alshaigy, Research student (PhD)
12012361@brookes.ac.uk
Department of Computing and Communication Technologies, Turing Bldg.
Oxford Brookes University, Wheatley Campus, Wheatley, Oxford OX33 1HX
United Kingdom

	Please initial box
I confirm that I have read and understand the information sheet for the above study and have had the opportunity to ask questions.	☐
I understand that my participation is voluntary and that I am free to withdraw at any time, without giving reason.	☐
I agree to take part in the above study.	☐
I agree that my data gathered in this study may be stored (after it has been anonymised) in a specialist data centre and may be used for future research.	☐

Please tick box

	Yes	**No**
I agree to the use of anonymised quotes in publications	☐	☐

______________________	______________	______________________
Name of Participant	**Date**	**Signature**
______________________	______________	______________________
Name of Researcher	**Date**	**Signature**

Appendix 8: Task Scenarios

<u>Scenario 1:</u>

You want to learn how to program in Python and you heard about PILeT, the link to the website was given to you, how do you visit the website?

Task	Pathway	Estimated Time (min)
1	Go to www.obpilt.com	> 1

<u>Scenario 2:</u>

You are a completely new user to the website and you would like to use it, how would you register?

Task	Pathway	Estimated Time (min)
2	Click Register > fill form using credential > click Register	3

<u>Scenario 3:</u>

Imagine you are already familiar with the website and you don't need to read the user manual, what would you do?

Task	Pathway	Estimated Time (min)
3	Click Skip	> 1

<u>Scenario 4:</u>

The website teaches several programming concepts. You want to learn about (Variables and Expressions), what would you do?

Task	Pathway	Estimated Time (min)
4	Navigate to Variables and Expression in the main page > click link	> 1

<u>Scenario 5:</u>

You successfully managed to land on (Variables and Expressions) page and you noticed a couple of embedded video, the videos explain the how to declare and use variables in Python. One of the videos explains the use of (input statement), how would you watch it?

Task	Pathway	Estimated Time (min)
5	Press play	> 1

Scenario 6:

You scrolled down the page to (ActiveCode 1) section and noticed a piece of embedded code in a box (don't worry about the syntax), you wanted to try it out to see the result of the execution, what would you do?

Task	Pathway	Estimated Time (min)
6	Click Run button	> 1

Scenario 7:

You want to change (edit) the code in the box to print("Goodbye!") and save the code, what would you do? Delete the code then try loading and executing it, what would you do?

Task	Pathway	Estimated Time (min)
7	Type in code in ActiveCode box > click Save button > delete code > click Load button > click on Run	5

Scenario 8:

You further scrolled down the page to (CodeLens1) and you noticed a piece of embedded code in it. Codelens is a tool that visually executes the code step by step. You want to trace the execution of the code. How would you do it?

Task	Pathway	Estimated Time (min)
8	Click Forward, back, first and last buttons	2

Scenario 9:

On the same page you navigated to (Parson's Programming Puzzles 1), you notice an exercise with code blocks. The blocks are arranged in the wrong order. How would you rearrange the blocks to resemble working code (don't worry about the correction of the solution)?

Task	Pathway	Estimated Time (min)
9	Drag code block from the left column to the right column > click Check Me and Reset buttons	2

Scenario 10:

You finally reach the quiz section of (Variables and Expressions) page, you want to answer the first multiple choice question and want to check your answer. How would you do it?

Task	**Pathway**	**Estimated Time (min)**
10	Select a radio button option > click Check Me button	> 1

Scenario 11:

Now you want to learn about (Selection Statements) in Python, it happens to be the next chapter, how would you do it?

Task	**Pathway**	**Estimated Time (min)**
11	Click Forward symbol button at the top of the page Or: Click on PILeT logo > navigate to Selection Statements in the main page > click link	2

Scenario 12:

Imagine you want to edit the user profile you created by changing the first name to Jack. How would you do it?

Task	**Pathway**	**Estimated Time (min)**
12	Click user button at top right corner > change first name > click Save changes button	3

Scenario 13:

Now you want to learn about (Functions) in Python. How would you do it?

Task	**Pathway**	**Estimated Time (min)**
13	Click on PILeT logo > navigate to Selection Statements in the main page > click link	2

Scenario 14:

Finally you want to sign out of PILeT. How would you do it?

Task	**Pathway**	**Estimated Time (min)**
14	Click user button at top right corner > click logout	> 1

Appendix 9: Post-test Questionnaire

On a scale from 1 to 5, where

1. Strongly disagree
2. Disagree
3. Neither agree nor disagree
4. Agree
5. Strongly agree

Please rate your agreement with each statement.

Number	Statement	Rating
1	The homepage is attractive.	
2	The overall site is attractive.	
3	The site's graphics are pleasing.	
4	The site has a good balance of graphics versus text.	
5	The colours used throughout the site are attractive.	
6	The typography (lettering, headings, titles) is attractive.	
7	The homepage's content makes me want to explore the site further.	
8	It is easy to find my way around the site.	
9	I can get to information quickly.	
10	It is fun to explore the site.	
11	It is easy to remember where to find things.	
12	Information is layered effectively on different screens.	
13	The homepage is attention-getting.	
14	Information is easy to read.	
15	Information is written in a style that suits me.	
16	Screens have the right amount of information.	
17	The site effectively communicates the company's identity.	
18	The information is relevant to my professional needs.	
19	The site is designed with me in mind.	
20	The site's content interests me.	
21	The site's content would keep me coming back.	
22	The site has characteristics that make it especially appealing.	
23	The site reflects progressive, leading edge design.	
24	The site is exciting.	
25	The site is well-suited to first-time visitors.	
26	The site is well-suited to repeat visitors.	
27	The site has a clear purpose.	
28	I always felt I knew what it was possible to do next.	
29	It is clear how screen elements (e.g., pop-ups, scrolling lists, menu options, etc.) work.	
30	My mistakes were easy to correct.	

Appendix 10: Usability Metrics

Evaluator No:

Task No.	Success Rate	Task Time	Error Rate	Satisfaction	Comments
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					

Appendix 11: Post-test Questionnaire Results

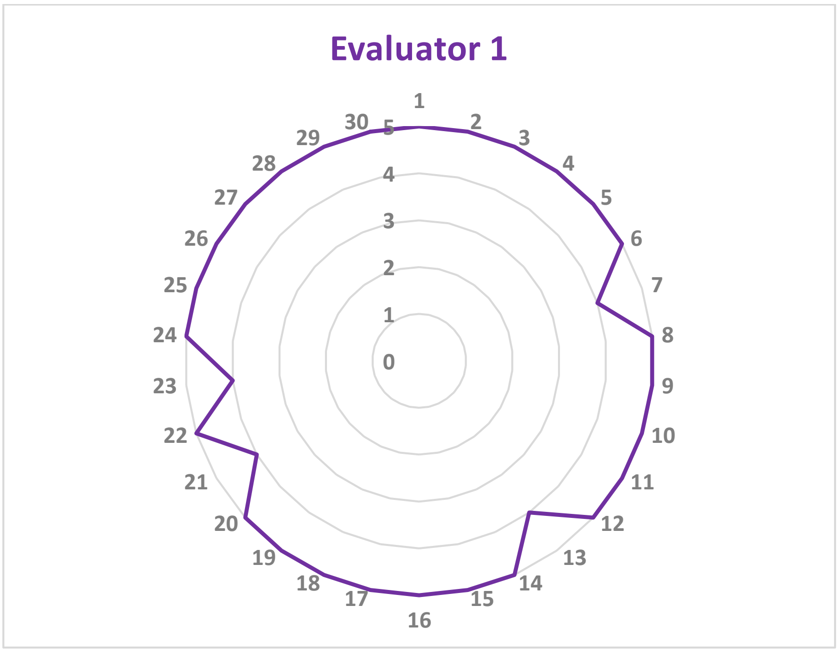


Figure A.17 Evaluator 1 Responses

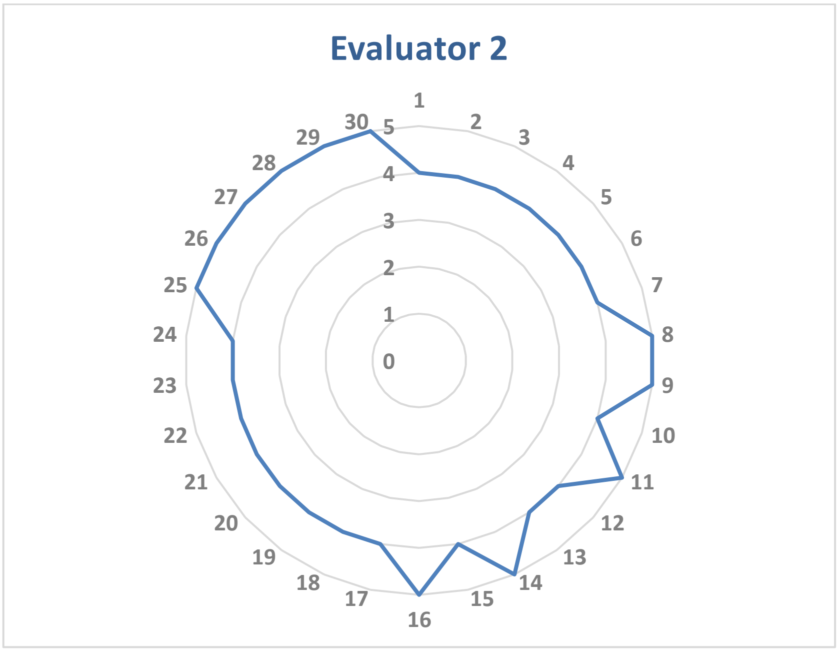


Figure A.18 Evaluator 2 Responses

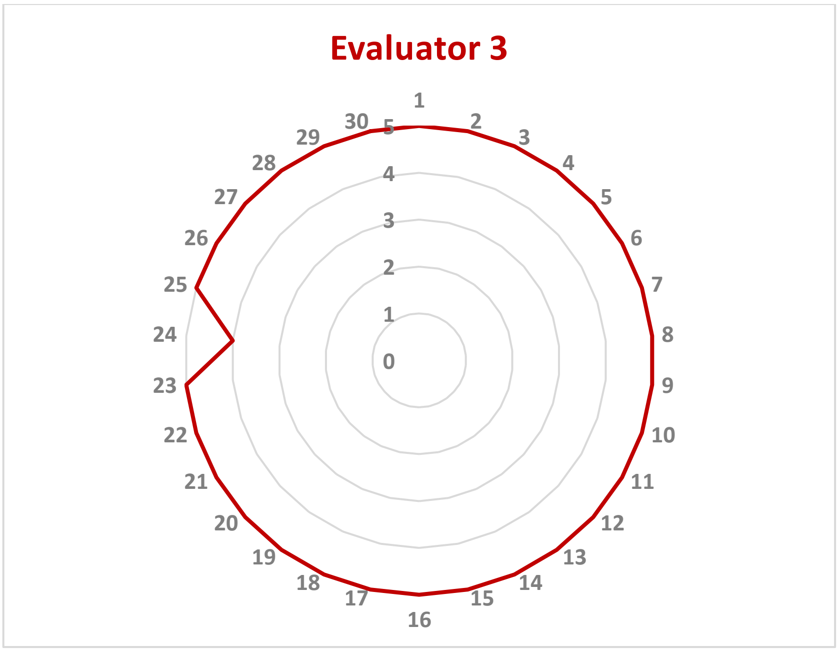


Figure A.19 Evaluator 3 Responses

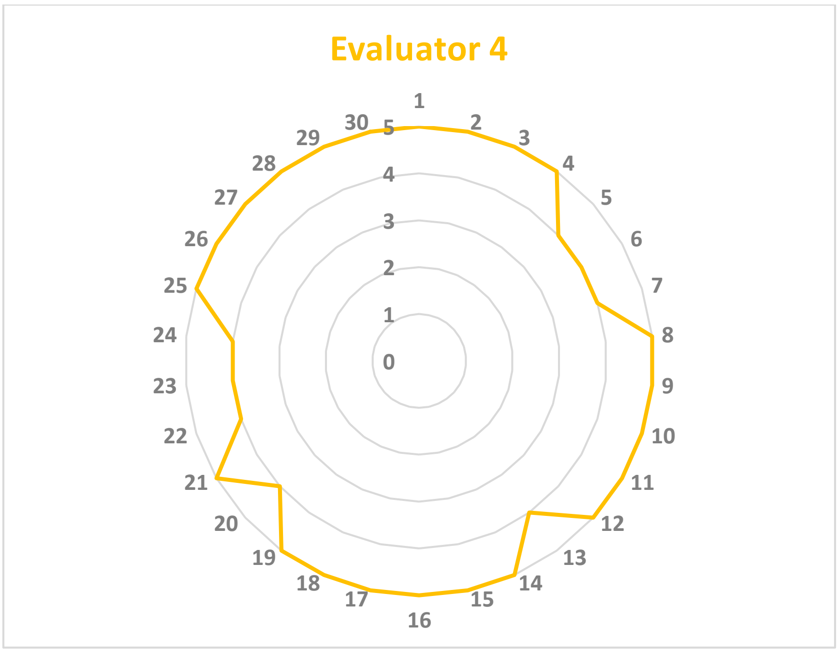


Figure A.20 Evaluator 4 Responses

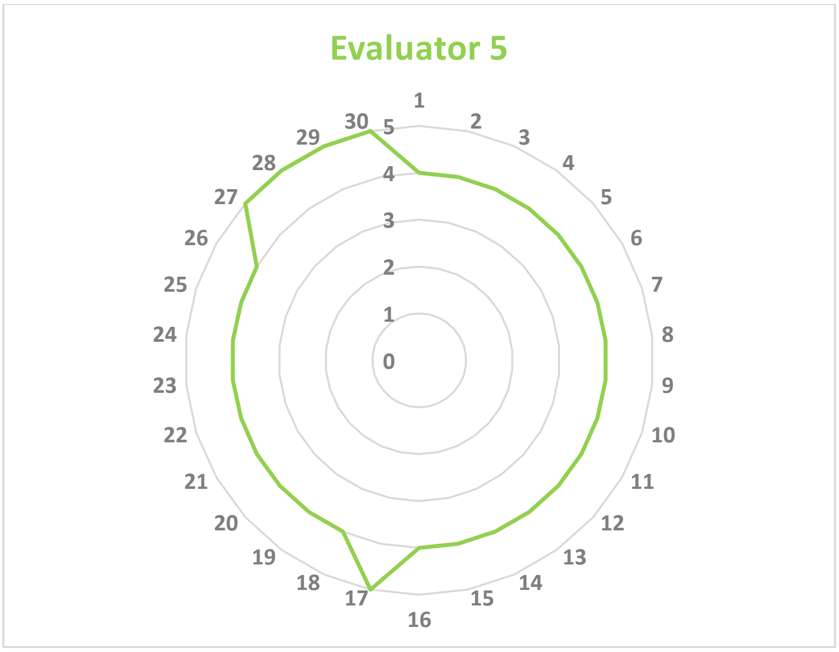


Figure A.21 Evaluator 5 Responses

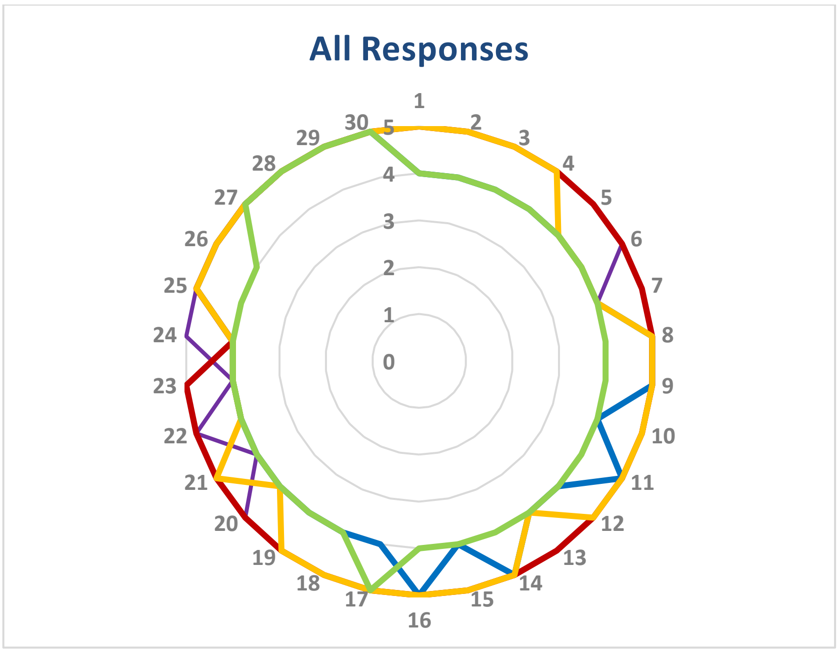


Figure A.22 All Responses Superimposed

Appendix 12: Consent Form

Study Title:

PILeT and CS Circles: Usability Study

Researcher Details:
Bedour Alshaigy, Research student (PhD)
12012361@brookes.ac.uk
Department of Computing and Communication Technologies, Turing Bldg.
Oxford Brookes University, Wheatley Campus, Wheatley, Oxford OX33 1HX
United Kingdom

	Please initial box
I confirm that I have read and understand the information sheet for the above study and have had the opportunity to ask questions.	☐
I understand that my participation is voluntary and that I am free to withdraw at any time, without giving reason.	☐
I agree to take part in the above study.	☐
I agree that my data gathered in this study may be stored (after it has been anonymised) in a specialist data centre and may be used for future research.	☐

Please tick box

	Yes	**No**
I agree to the use of anonymised quotes in publications	☐	☐

Name of Participant	**Date**	**Signature**
Name of Researcher	**Date**	**Signature**

Appendix 13: Usability Survey

On a scale from 1 to 5, where

1. Strongly disagree
2. Disagree
3. Neither agree nor disagree
4. Agree
5. Strongly agree

Please rate your agreement with each statement.

A. Navigation Design

Number	Statement	CS Circles Rating	PILeT Rating
1	I found it easy to navigate [website name here].		
2	It was easy to know which page I was on and what other pages I visited.		
3	It was easy to locate certain pages of [website name here]		

B. Page Layout

Number	Statement	CS Circles Rating	PILeT Rating
1	I am happy with the amount of information on the page.		
2	I am happy with the arrangement of the interactive features.		
3	I am happy with the table of content.		

C. Content Design

Number	Statement	CS Circles Rating	PILeT Rating
1	I found the font easy to read.		
2	I found the page of acceptable length.		
3	I found the page legible.		
4	I liked the typographical aspects of the page.		
5	I found it easy to scan through the page.		
6	I liked the colours of the page.		

For the final question, please circle the right answer.

D. Interactive Features

Number	**Statement**	**CS Circles**		**PILeT**	
1	I knew the purpose of	*Console*	Yes No	*ActiveCode*	Yes No
2	I found the feature useful	*Console*	Yes No	*ActiveCode*	Yes No
3	I knew the purpose of	*Codelens*	Yes No	*CodeLens*	Yes No
4	I found the feature useful	*Codelens*	Yes No	*CodeLens*	Yes No
5	I knew the purpose of	*Code scrambler*	Yes No	*Parson's Programming Puzzles*	Yes No
6	I found the feature useful	*Code scrambler*	Yes No	*Parson's Programming Puzzles*	Yes No

Appendix 14: System Usability Scale

On a scale from 1 to 5, where

6. Strongly disagree
7. Disagree
8. Neither agree nor disagree
9. Agree
10. Strongly agree

Please rate your agreement with each statement.

Number	Statement	Rating
1	I think that I would like to use this system frequently.	
2	I found the system unnecessarily complex.	
3	I thought the system was easy to use.	
4	I think that I would need the support of a technical person to be able to sue the system.	
5	I found the various functions in this system were well integrated.	
6	I thought there was too much inconsistency in this system.	
7	I would imagine that most people would learn to use this system very quickly	
8	I found the system cumbersome to use.	
9	I felt very confident using the system.	
10	I needed to learn a lot of things before I could get going with this system.	

Appendix 15: SUS Questionnaire Results for CS Circles

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Score
1	3	2	5	1	4	3	4	4	5	1	75
2	3	2	3	2	3	3	3	3	4	3	57.5
3	3	3	4	1	4	1	3	4	4	3	65
4	4	2	3	2	5	3	2	3	4	1	67.5
5	4	2	3	2	3	3	3	3	4	1	65
6	2	2	4	1	3	2	4	4	3	2	62.5
7	1	4	2	3	3	3	2	4	3	3	35
8	1	5	1	3	2	4	1	3	1	5	15
9	2	3	3	4	2	5	4	4	2	3	35
10	1	2	2	4	3	3	1	3	1	4	30
11	2	3	2	3	2	2	2	3	3	3	42.5
12	2	3	3	4	2	2	2	3	3	2	45
13	2	3	3	3	3	2	3	2	2	2	52.5
14	3	2	4	2	4	3	4	3	4	1	70
15	2	3	3	5	2	4	2	2	3	4	35
16	4	2	4	2	4	2	2	2	3	4	62.5
17	2	3	3	2	3	3	1	3	2	3	42.5
18	1	3	3	1	3	3	2	5	3	4	40
19	1	1	1	1	1	1	1	1	1	1	50
20	3	3	4	1	3	4	3	5	3	2	52.5
21	1	4	2	3	2	4	2	4	1	4	22.5
22	1	4	1	3	2	3	1	2	2	3	30
23	2	2	2	3	2	2	2	3	2	3	42.5
24	1	3	2	2	2	3	2	5	1	4	27.5
Total											**1122.5**
AVG											**46.7**

Appendix 16: SUS Questionnaire Results for PILeT

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Score
1	4	4	4	3	3	5	4	2	5	3	57.5
2	5	1	5	1	5	1	5	1	5	1	100
3	4	4	3	1	5	3	4	3	3	1	67.5
4	5	1	5	1	5	1	5	1	5	1	100
5	1	5	4	1	2	4	2	5	3	1	40
6	5	5	5	5	5	5	5	5	5	5	50
7	5	1	5	1	4	3	5	2	5	3	85
8	4	2	4	5	4	2	1	1	3	4	55
9	5	1	5	1	5	1	5	1	5	1	100
10	5	3	4	2	3	3	3	3	5	3	65
11	2	2	3	2	2	2	1	2	2	1	52.5
12	3	3	3	3	3	3	3	3	3	3	50
13	5	1	5	1	5	1	5	1	5	1	100
14	3	3	3	3	3	3	3	3	4	4	50
15	5	5	5	1	4	1	4	2	4	1	80
16	5	1	4	1	4	1	5	1	5	1	95
17	4	1	5	1	4	3	5	2	5	3	82.5
18	4	3	4	3	3	3	4	3	4	3	60
19	4	2	5	1	5	3	2	2	5	2	77.5
20	2	2	5	1	4	2	4	3	3	2	70
21	2	3	3	2	3	2	2	3	3	2	52.5
22	4	2	4	1	5	2	4	2	4	2	80
23	4	1	5	1	5	3	5	2	5	3	85
24	5	2	5	2	3	3	5	2	4	2	77.5
25	4	2	5	1	5	3	2	2	5	2	77.5
Total											**1810**
AVG											**72.4**

Appendix 17: Selection Statements Quiz

Please answer the following questions:

Q1. Write a program that asks the user for their age. If they are 18 or over then display the message “Congratulations, you are allowed to vote”, otherwise display the message “Sorry, you are not old enough to vote”. **(2 marks)**

Q2. Write a program that reads a temperature in Celsius and prints the state of water whether it is liquid, solid, or gaseous at that temperature at sea level. **(2 marks)**

Q3. Write a program that reads a numerical mark and translates into a grade. Here you have the grades: **(2 marks)**

Grade A: mark >= 70
Grade B+: 60 <= mark <= 69
Grade B: 50 <= mark <= 59
Grade C: 40 <= mark <= 49
Grade Resit: 30 <= mark <= 39
Grade Fail: mark

Mark out of 6:

Appendix 18: Loops Quiz

Please answer the following questions:

Q1. Print the statement (I love Python) 5 times using a for loop. **(2 marks)**

Q2. Print the statement (I love Python) 5 times using a while loop. **(2 marks)**

Q3. Use a for loop to print the sum of 5 numbers beginning from 5. **(2 marks)**

Q4. Use a while loop to print the sum of 5 numbers beginning from 5. **(2 marks)**

Mark out of 8:

Appendix 19: Results Part 1

Selection Statements Results

Participant No	Control Group	CS Circles	PILeT
1	6	3	6
2	6	6	6
3	6	1	6
4	6	6	6
5	6	2	4
6	5	3	6
7	6	1	2
8	6	3	6
9	6	6	6
10	6	4	6
11		3	3
12		3	0
13		4	6
14		3	6
15		3	3
16		3	4
17		4	6
18		4	4
19		3	3

Loops Results

Participant No	Control Group	CS Circles	PILeT
1	7	6	6
2	8	0	7
3	6	0	8
4	4	0	5
5	7	0	5
6	4	0	6
7	4	0	7
8	2	0	6
9	6	0	6
10	5	6	4
11		2	4
12			4
13			8
14			4

Appendix 20: Index of Learning Styles Questionnaire

Circle "a" or "b" to indicate your answer to every question. Please choose only one answer for each question. If both "a" and "b" seem to apply to you, choose the one that applies more frequently.

1. I understand something better after I
(a) try it out.
(b) think it through.
2. I would rather be considered
(a) realistic.
(b) innovative.
3. When I think about what I did yesterday, I am most likely to get
(a) a picture.
(b) words.
4. I tend to
(a) understand details of a subject but may be fuzzy about its overall structure.
(b) understand the overall structure but may be fuzzy about details.
5. When I am learning something new, it helps me to
(a) talk about it.
(b) think about it.
6. If I were a teacher, I would rather teach a course
(a) that deals with facts and real life situations.
(b) that deals with ideas and theories.
7. I prefer to get new information in
(a) pictures, diagrams, graphs, or maps.
(b) written directions or verbal information.
8. Once I understand
(a) all the parts, I understand the whole thing.
(b) the whole thing, I see how the parts fit.
9. In a study group working on difficult material, I am more likely to
(a) jump in and contribute ideas.
(b) sit back and listen.
10. I find it easier
(a) to learn facts.
(b) to learn concepts.
11. In a book with lots of pictures and charts, I am likely to
(a) look over the pictures and charts carefully.
(b) focus on the written text.
12. When I solve math problems
(a) I usually work my way to the solutions one step at a time.
(b) I often just see the solutions but then have to struggle to figure out the steps to get to them.
13. In classes I have taken
(a) I have usually gotten to know many of the students.
(b) I have rarely gotten to know many of the students.

14. In reading nonfiction, I prefer
(a) something that teaches me new facts or tells me how to do something.
(b) something that gives me new ideas to think about.
15. I like teachers
(a) who put a lot of diagrams on the board.
(b) who spend a lot of time explaining.
16. When I'm analyzing a story or a novel
(a) I think of the incidents and try to put them together to figure out the themes.
(b) I just know what the themes are when I finish reading and then I have to go back and find the incidents that demonstrate them.
17. When I start a homework problem, I am more likely to
(a) start working on the solution immediately.
(b) try to fully understand the problem first.
18. I prefer the idea of
(a) certainty.
(b) theory.
19. I remember best
(a) what I see.
(b) what I hear.
20. It is more important to me that an instructor
(a) lay out the material in clear sequential steps.
(b) give me an overall picture and relate the material to other subjects.
21. I prefer to study
(a) in a study group.
(b) alone.
22. I am more likely to be considered
(a) careful about the details of my work.
(b) creative about how to do my work.
23. When I get directions to a new place, I prefer
(a) a map.
(b) written instructions.
24. I learn
(a) at a fairly regular pace. If I study hard, I'll "get it."
(b) in fits and starts. I'll be totally confused and then suddenly it all "clicks."
25. I would rather first
(a) try things out.
(b) think about how I'm going to do it.
26. When I am reading for enjoyment, I like writers to
(a) clearly say what they mean.
(b) say things in creative, interesting ways.
27. When I see a diagram or sketch in class, I am most likely to remember
(a) the picture.
(b) what the instructor said about it.
28. When considering a body of information, I am more likely to
(a) focus on details and miss the big picture.
(b) try to understand the big picture before getting into the details.
29. I more easily remember
(a) something I have done.

(b) something I have thought a lot about.
30. When I have to perform a task, I prefer to
(a) master one way of doing it.
(b) come up with new ways of doing it.
31. When someone is showing me data, I prefer
(a) charts or graphs.
(b) text summarizing the results.
32. When writing a paper, I am more likely to
(a) work on (think about or write) the beginning of the paper and progress forward.
(b) work on (think about or write) different parts of the paper and then order them.
33. When I have to work on a group project, I first want to
(a) have "group brainstorming" where everyone contributes ideas.
(b) brainstorm individually and then come together as a group to compare ideas.
34. I consider it higher praise to call someone
(a) sensible.
(b) imaginative.
35. When I meet people at a party, I am more likely to remember
(a) what they looked like.
(b) what they said about themselves.
36. When I am learning a new subject, I prefer to
(a) stay focused on that subject, learning as much about it as I can.
(b) try to make connections between that subject and related subjects
37. I am more likely to be considered
(a) outgoing.
(b) reserved.
38. I prefer courses that emphasize
(a) concrete material (facts, data).
(b) abstract material (concepts, theories).
39. For entertainment, I would rather
(a) watch television.
(b) read a book.
40. Some teachers start their lectures with an outline of what they will cover. Such outlines are
(a) somewhat helpful to me.
(b) very helpful to me.
41. The idea of doing homework in groups, with one grade for the entire group,
(a) appeals to me.
(b) does not appeal to me.
42. When I am doing long calculations,
(a) I tend to repeat all my steps and check my work carefully.
(b) I find checking my work tiresome and have to force myself to do it.
43. I tend to picture places I have been
(a) easily and fairly accurately.
(b) with difficulty and without much detail.
44. When solving problems in a group, I would be more likely to
(a) think of the steps in the solution process.
(b) think of possible consequences or applications of the solution in a wide range of areas.

Appendix 21: Results Part 2

Selection Statements Results

Participant No	Control Group	PILeT
1	6	6
2	6	6
3	4	6
4	5	6
5	4	4
6	5	6
7	4	5
8	6	6
9	6	6
10	6	6
11	2	4
12	5	6

Loops Results

Participant No	Control Group	PILeT
1	7	8
2	8	7
3	6	8
4	4	6
5	7	7
6	4	6
7	4	7
8	2	7
9	6	6
10	5	8
11	4	8
12	6	8
13		8