

# Computing Education and Learning Technology Research Group



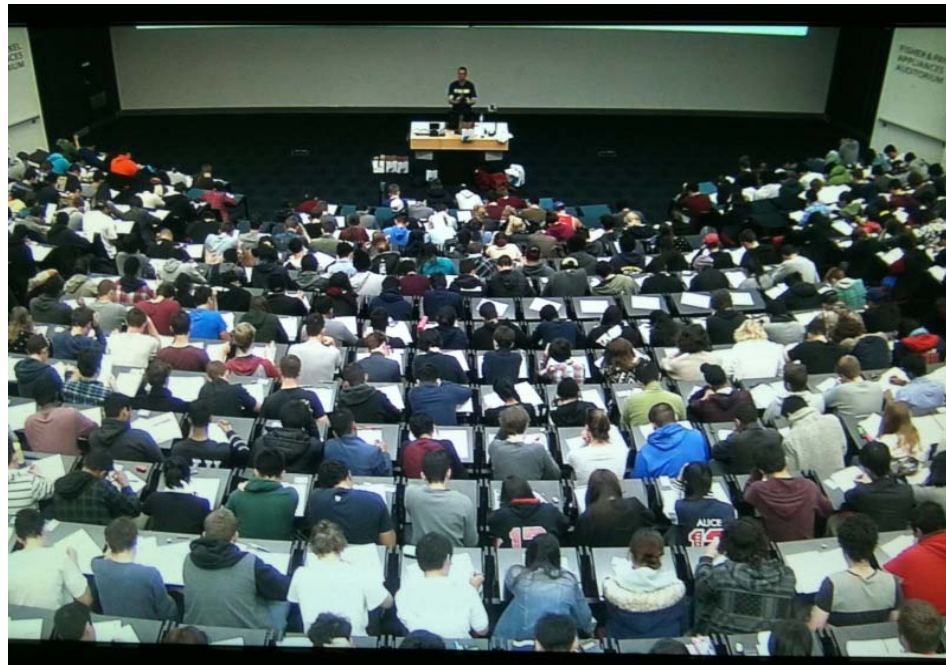
Dr Paul Denny  
Associate Professor  
School of Computer Science  
[p.denny@auckland.ac.nz](mailto:p.denny@auckland.ac.nz)

# Overview

- What is Computing Education & Learning Technology research?
- Why is it an interesting area of research?
- A few examples
  - student projects (including COMPSCI 747)
  - leveraging existing expertise in Computer Science
- Overarching research questions
- Two examples
  - Specific research questions

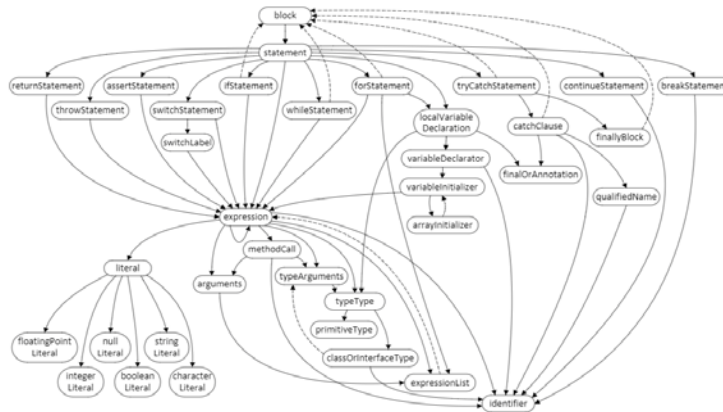
# What is it?

- Distinct from “teaching”
  - Teaching is helping others acquire knowledge and develop skills in a discipline
  - Research is creating new knowledge and exploring new ideas



# What is it?

- Distinct from “teaching”
  - Teaching is helping others acquire knowledge and develop skills in a discipline
  - Research is creating new knowledge and exploring new ideas
- Computing Education Research:
  - the study of how people learn and teach computing
  - the goal is to help students learn, and teachers teach, more effectively



```
int sum = 0;

if(a==2*(a/2)) {
    if(b!=2*(b/2)) {
        sum = a+b;
    }
} else {
    if(b==2*(b/2)) {
        sum = a+b;
    }
}

return sum;
```

# What is it?

- Distinct from “teaching”
  - Teaching is helping others acquire knowledge and develop skills in a discipline
  - Research is creating new knowledge and exploring new ideas
- Computing Education Research:
  - the study of how people learn and teach computing
  - the goal is to help students learn, and teachers teach, more effectively
- Learning Technology Research:
  - designing and evaluating tools for learning
  - covers the broader use of technology in teaching, learning and education across disciplines

# Why research education?

- To have a positive impact in the world
  - Better outcomes for learners

# Why research education?

- To have a positive impact in the world
  - Better outcomes for learners
- Practical application of technology
  - A clear need and an enormous user base

# Why research education?

- To have a positive impact in the world
  - Better outcomes for learners
- Practical application of technology
  - A clear need and an enormous user base
- Big business
  - “Computational Thinking” is an essential 21<sup>st</sup> century skill, yet there are few people who can help others to develop those skills.
    - Constant need to train staff in technology
    - Increasing need for non-CS people to program
    - Increasing integration of programming skills into school curriculum
    - Increasing number of companies involved in technology to support education, and education about CS.



Beehive.govt.nz

The official website of the New Zealand Government

Menu

30 OCTOBER 2018

**International education contributes \$5.1 billion to New Zealand economy**



HON CHRIS HIPKINS

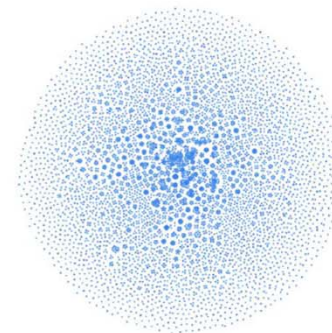
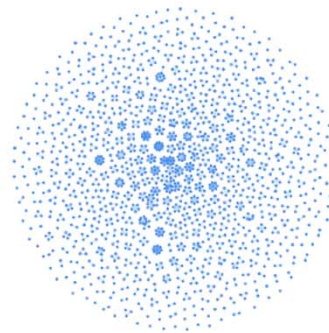
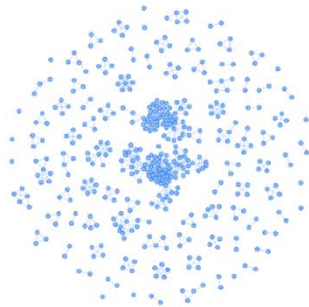
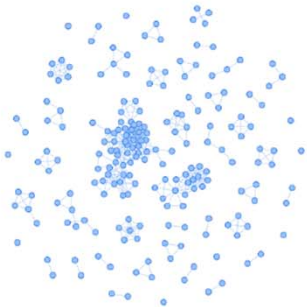
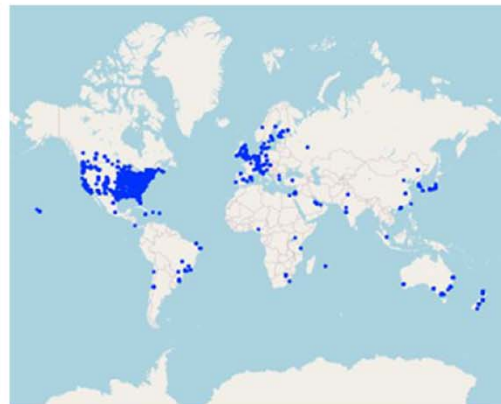
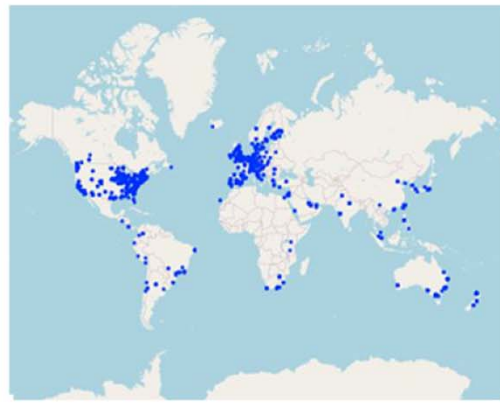


## A few student projects

- Researchers apply their expertise from many areas of CS

# A few student projects

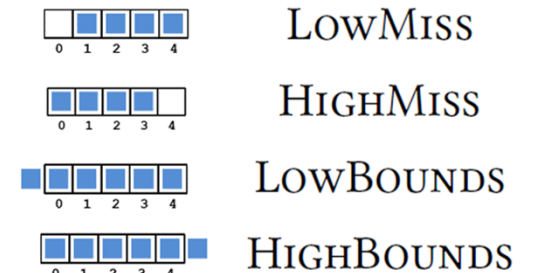
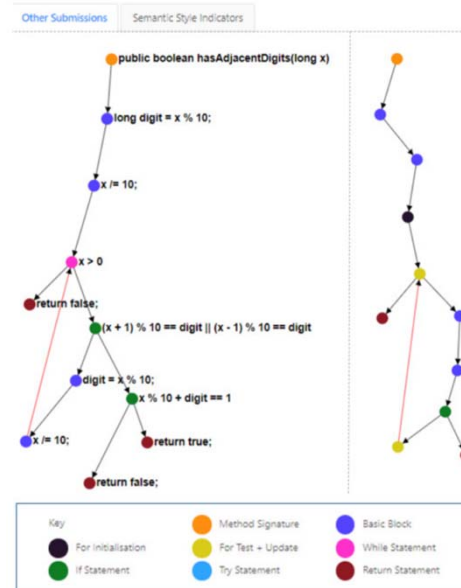
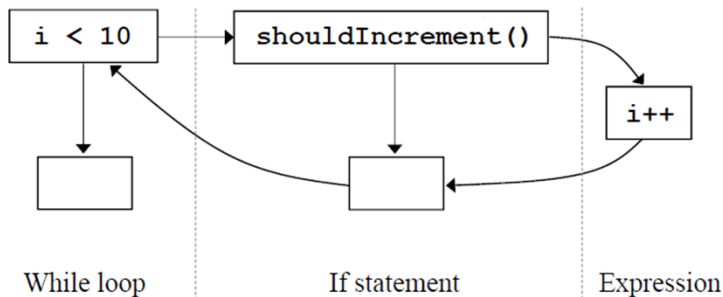
- Researchers apply their expertise from many areas of CS
  - **graph theory** (e.g. bibliometric analysis - [James](#))



# A few student projects

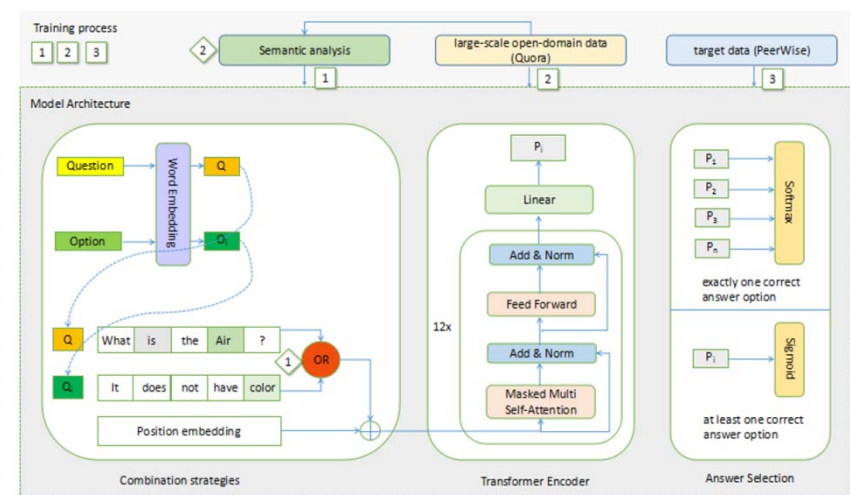
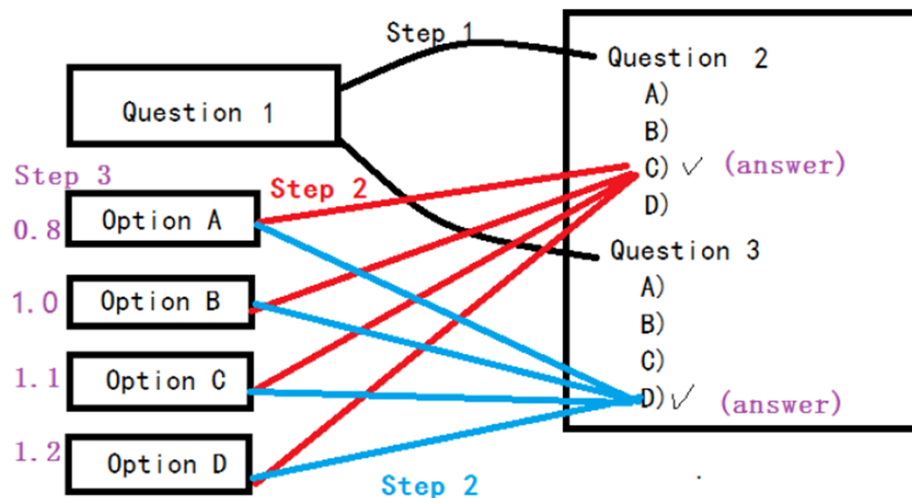
- Researchers apply their expertise from many areas of CS
  - graph theory (e.g. bibliometric analysis - [James](#))
  - **program analysis**
    - static analysis (e.g. generating ASTs for visualising control flow - [Lucy and Robert](#))
    - dynamic analysis (e.g. classification of array access errors - [Liam](#))

```
while (i < 10) {  
    if (shouldIncrement()) {  
        i++;  
    }  
}
```



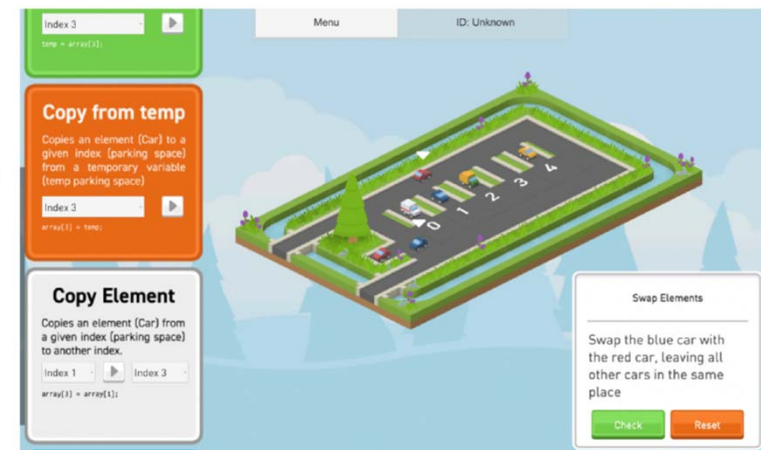
# A few student projects

- Researchers apply their expertise from many areas of CS
  - graph theory (e.g. bibliometric analysis - [James](#))
  - program analysis
    - static analysis (e.g. generating ASTs for visualising control flow - [Lucy and Robert](#))
    - dynamic analysis (e.g. classification of array access errors - [Liam](#))
  - **machine learning/AI** (e.g. NLP for question answering - [Bill](#))



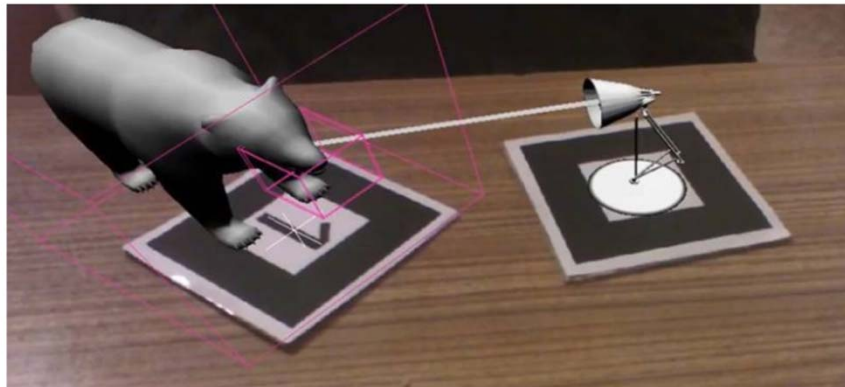
# A few student projects

- Researchers apply their expertise from many areas of CS
  - graph theory (e.g. bibliometric analysis - [James](#))
  - program analysis
    - static analysis (e.g. generating ASTs for visualising control flow - [Lucy and Robert](#))
    - dynamic analysis (e.g. classification of array access errors - [Liam](#))
  - machine learning/AI (e.g. NLP for question answering - [Bill](#))
  - **game design** (e.g. teaching version control, data structures – [Kevin, Simon](#))



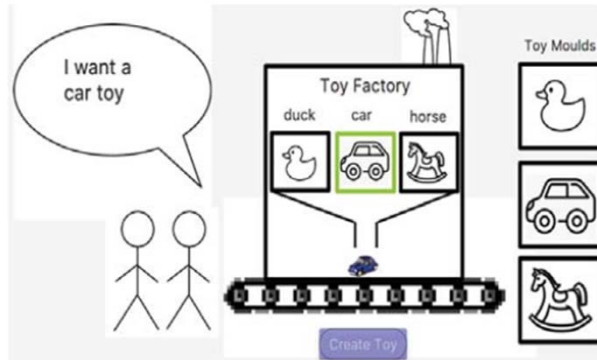
## A few student projects

- Researchers apply their expertise from many areas of CS
  - graph theory (e.g. bibliometric analysis - [James](#))
  - program analysis
    - static analysis (e.g. generating ASTs for visualising control flow - [Lucy and Robert](#))
    - dynamic analysis (e.g. classification of array access errors - [Liam](#))
  - machine learning/AI (e.g. NLP for question answering - [Bill](#))
  - game design (e.g. teaching version control, data structures – [Kevin, Simon](#))
  - **VR/AR** (e.g. visualising object transformations for computer graphics - [Thomas](#))



# A few student projects

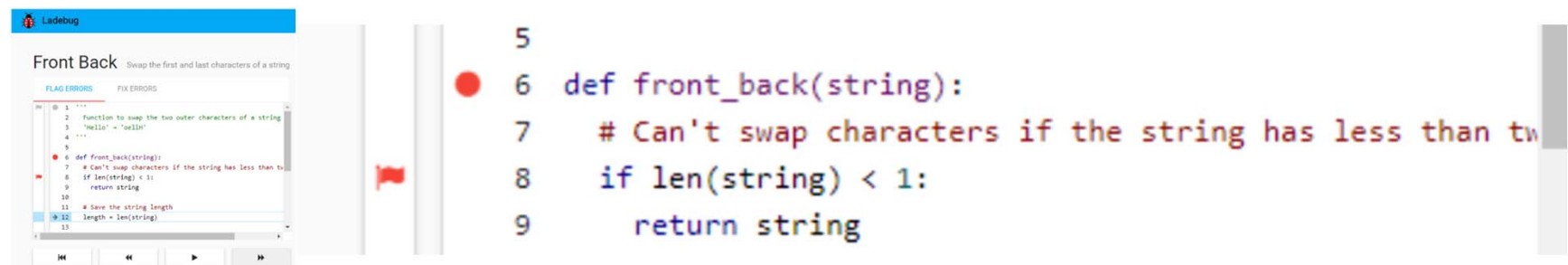
- Researchers apply their expertise from many areas of CS
  - graph theory (e.g. bibliometric analysis - [James](#))
  - program analysis
    - static analysis (e.g. generating ASTs for visualising control flow - [Lucy and Robert](#))
    - dynamic analysis (e.g. classification of array access errors - [Liam](#))
  - machine learning/AI (e.g. NLP for question answering - [Bill](#))
  - game design (e.g. teaching version control, data structures – [Kevin, Simon](#))
  - VR/AR (e.g. visualising object transformations for computer graphics - [Thomas](#))
  - **software design** (e.g. using metaphors for teaching design patterns - [Zain](#))





# A few student projects

- Researchers apply their expertise from many areas of CS
  - graph theory (e.g. bibliometric analysis - [James](#))
  - program analysis
    - static analysis (e.g. generating ASTs for visualising control flow - [Lucy and Robert](#))
    - dynamic analysis (e.g. classification of array access errors - [Liam](#))
  - machine learning/AI (e.g. NLP for question answering - [Bill](#))
  - game design (e.g. teaching version control, data structures – [Kevin, Simon](#))
  - VR/AR (e.g. visualising object transformations for computer graphics - [Thomas](#))
  - software design (e.g. using metaphors for teaching design patterns - [Zain](#))
  - **programming** (e.g. interactive tool for teaching debugging skills - [Emma and Liz](#))





# A few student projects

- Researchers apply their expertise from many areas of CS
  - graph theory (e.g. bibliometric analysis - [James](#))
  - program analysis
    - static analysis (e.g. generating ASTs for visualising control flow - [Lucy and Robert](#))
    - dynamic analysis (e.g. classification of array access errors - [Liam](#))
  - machine learning/AI (e.g. NLP for question answering - [Bill](#))
  - game design (e.g. teaching version control, data structures – [Kevin, Simon](#))
  - VR/AR (e.g. visualising object transformations for computer graphics - [Thomas](#))
  - software design (e.g. using metaphors for teaching design patterns - [Zain](#))
  - programming (e.g. interactive tool for teaching debugging skills - [Emma and Liz](#))
  - **computer systems** (e.g. compiler error messages - [Dave](#))

```
if (a < 0) || (a > 100)
    error = true;
```



```
if (a < 0) || (a > 100)
           ^^
```

Syntax error on token "||", if expected

1 error

# COMPSCI 747

## • Graduate course in Computing Education

### A Miss is as Good as a Mile: Off-By-One Errors and Arrays in an Introductory Programming Course

Liam Rigby  
The University of Auckland  
New Zealand  
lrig521@aucklanduni.ac.nz

Paul Denny  
The University of Auckland  
New Zealand  
paul@cs.auckland.ac.nz

Andrew Luxton-Reilly  
The University of Auckland  
New Zealand  
a.luxton-reilly@auckland.ac.nz

#### ABSTRACT

Loops and arrays are fundamental CS concepts, but ones that can be problematic for novice programmers. In this research, we investigate off-by-one errors – logic errors where loops perform one too few or one too many iterations – in code using an indexed

#### LOWMISS

(first element, at index 0, is missed)

```
for (i = 1; i < length1; i++) {  
    sum += array[i];  
}
```

### A Review of Research on Parsons Problems

Yuemeng Du  
ydu511@aucklanduni.ac.nz  
University of Auckland  
Auckland, New Zealand

Andrew Luxton-Reilly  
andrew@cs.auckland.ac.nz  
University of Auckland  
Auckland, New Zealand

Paul Denny  
paul@cs.auckland.ac.nz  
University of Auckland  
Auckland, New Zealand

#### ABSTRACT

Parsons problems are a type of programming exercise where students rearrange jumbled code blocks of a solution program back into its original form. It is usually implemented as a complement or alternative to traditional programming exercises like code-tracing and code-writing. This paper reviews the existing literature on the Parsons problem in introductory CS education. We find that the flexible nature of the design of Parsons problems has led to many variants, and these have been continuously refined to better address student needs. However, the effectiveness of Parsons problems, both as a question type and as a learning tool in CS education, remains uncertain due to a lack of replicated research in the field.

#### KEYWORDS

Parsons problem, Parsons puzzle, programming exercise

#### ACM Reference Format:

Yuemeng Du, Andrew Luxton-Reilly, and Paul Denny. 2020. A Review of Research on Parsons Problems. In *Proceedings of the Twenty-Second Australian Computing Education Conference (ACE'20)*, February 3–7, 2020, Melbourne, VIC, Australia. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3373165.3373187>

#### 1 INTRODUCTION

Introductory programming courses must necessarily teach students the syntax of programming and mechanical learning to a certain extent. At many universities, this is achieved through exercises that involve code-writing. Research has suggested that reliance on traditional programming exercises often decrease student engagement and motivation [4], which directly influence student performance. Furthermore, tasks such as code-writing can often end up being challenging and time-intensive to students in ways unintended by the instructors. A potential solution to this is to provide a novel way for students to learn to program, one that imposes less cognitive

[1] and the Exploring Computer Science Curriculum [17]. ever, these are mainly informal programming courses that younger (high school) students, serving as preparation for a university study.

At university level, block-based programming exercises are frequently implemented. Nevertheless, the Parsons problem, a and-drop style, block-based program construction exercise been a notable and consistent presence in introductory CS university courses. Unlike traditional code-writing exercises, students are supplied with code fragments that are already written, which relieves a considerable amount of cognitive load. Furthermore, Parsons problems can be easily implemented in an online or n environment, making it easier for students to engage in learn real time.

A survey of existing literature shows that, while there is existing research on the adoption of Parsons problems, the research is scattered and concern themselves with different aspects of education. This paper is a review of the literature surrounding Parsons problems in introductory CS education at the university. It aims to provide an overview of how the Parsons problem evolved over the years, both in terms of the question type and how it is implemented. The following research question addressed in subsequent sections:

- (1) Why are Parsons problems studied in CS education?
- (2) What are the features of Parsons problems?
- (3) How are Parsons problems used in CS education?

#### 2 METHODOLOGY

We conducted a systematic review of the literature using the lines proposed by [26]. The main steps include the identification of research questions, selection of primary studies, assessment of the quality of the studies, data extraction and synthesis.

To identify primary studies, we conducted a search of the

### Mastery Learning in Computer Science Education

James Garner  
University of Auckland  
New Zealand  
jgar569@aucklanduni.ac.nz

Paul Denny  
University of Auckland  
New Zealand  
paul@cs.auckland.ac.nz

Andrew Luxton-Reilly  
University of Auckland  
New Zealand  
andrew@cs.auckland.ac.nz

#### ABSTRACT

Mastery learning is a pedagogical approach in which students must demonstrate mastery of the currently assessed unit of material before being permitted to progress to the next unit. Recent work has suggested that mastery learning may provide a solution to the divergent outcomes observed in introductory computer science (CS) courses. While mastery learning has shown benefits outside of CS, it has received less attention in CS education, and there is no existing overview of the approaches that have been used in

of a student to grasp an early concept will be magnified in assessments involving later concepts. This problem should not arise in a mastery learning approach, since mastery learning does not permit students to attempt a later assessment without first mastering earlier content. Several of the papers included in this review of the 'Learning Edge Momentum' paper as motivation for adopting mastery learning approach.

Despite the relevance of mastery learning to computer science education today, the literature on it is scarce, which influenced

### Transitioning from Block-based to Text-based Programming Languages

Luke Moors  
Department of Computer Science  
The University of Auckland  
Auckland, New Zealand  
lmoor228@aucklanduni.ac.nz

Andrew Luxton-Reilly  
Department of Computer Science  
The University of Auckland  
Auckland, New Zealand  
andrew@cs.auckland.ac.nz

Paul Denny  
Department of Computer Science  
The University of Auckland  
Auckland, New Zealand  
paul@cs.auckland.ac.nz

**Abstract—**Block-based programming environments are becoming increasingly popular as introductory tools for teaching programming to children. These environments differ significantly from their text-based counterparts and have proven to be successful in motivating children and making it easy to start programming. However, several studies have recognised drawbacks of these tools which could potentially be detrimental when students transition to text-based languages. In this paper, the distinguishing features between block-based environments and text-based languages are discussed and the effectiveness of these features are explained. In considering the transition to text-based languages, this paper identifies two significant weaknesses to block-based programming and discusses suggestions for improvement.

**Keywords—**k-12, k12, block-based, introductory programming, novice

#### I. INTRODUCTION

It is widely acknowledged that programming is difficult to learn [1]–[5]. Novice programmers have great difficulty in typical programming tasks such as predicting the output of a program, identifying the correct order of commands, or writing a simple program to solve a task [4]. Many of these tasks are challenging to novices as they are required

environment launched in 2007, there has been an increase from 78,000 to 174,000 monthly active users in the past two years alone [10]. This style of programming implements a block-like structure where blocks of code fit together like jigsaw pieces. These blocks differ in shape and colour to provide cues about how instructions can be assembled and to differentiate between concepts [5]. In addition, the environments typically encourage novice programmers by allowing them to create media-rich content in relation to their own interests. *Storytelling Alice*, in particular, implements a *Storytelling* approach found to be appealing to female students [11], [12]. On the other hand, *App Inventor* encourages novices using a block-based style of programming to provide the ability to create mobile applications in a simplified manner [7]. Many of the environments which use block-based programming (e.g. Scratch, Alice, App Inventor) aim to lower the barriers to programming, making it easier for beginners to start programming [13].

Much of the research related to the subject has reported the benefits of such environments including increased motivation and improved grades [11], [14], [15], however other studies have suggested potential drawbacks to block-based

### A Review of Peer Code Review in Higher Education

THERESIA DEVI INDRIASARI, School of Computer Science, The University of Auckland, New Zealand and Department of Informatics, Universitas Atma Jaya Yogyakarta, Indonesia

ANDREW LUXTON-REILLY and PAUL DENNY, School of Computer Science, The University of Auckland, New Zealand

Peer review is the standard process within academia for maintaining publication quality, but it is also widely employed in other settings, such as education and industry, for improving work quality and for generating actionable feedback to content authors. For example, in the software industry peer review of program source code—or peer code review—is a key technique for detecting bugs and maintaining coding standards. In a programming education context, although peer code review offers potential benefits to both code reviewers and code authors, individuals are typically less experienced, which presents a number of challenges. Some

### Common Logic Errors Made By Novice Programmers

Andrew Ettles  
University of Auckland  
Auckland, New Zealand  
aett072@aucklanduni.ac.nz

Andrew Luxton-Reilly  
University of Auckland  
Auckland, New Zealand  
andrew@cs.auckland.ac.nz

Paul Denny  
University of Auckland  
Auckland, New Zealand  
paul@cs.auckland.ac.nz

#### ABSTRACT

Errors in the logic of a program (sometimes referred to as semantic errors) can be very frustrating for novice programmers to locate and resolve. Developing a better understanding of the kinds of logic error that are most common and problematic for students, and finding strategies for targeting them, may help to inform teaching practice and reduce student frustration.

In this paper we analyse 15,000 code fragments, created by novice programming students, that contain logic errors, and we classify the errors into algorithmic errors, misinterpretations of the problem, and fundamental misconceptions. We find that misconceptions are the most frequent source of logic errors, and lead to the most difficult errors for students to resolve. We list the most common errors of this type as a starting point for designing specific teaching interventions to address them.

#### CCS CONCEPTS

• Social and professional topics → Computing education;

#### KEYWORDS

CS1, logic errors, novice programmers

#### ACM Reference Format:

Andrew Ettles, Andrew Luxton-Reilly, and Paul Denny. 2018. Common Logic Errors Made By Novice Programmers. In *ACE 2018: 20th Australian Computing Education Conference*, January 30–February 2, 2018, Brisbane, QLD, Australia. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/3160489.3160495>

#### 1 INTRODUCTION

Empirical research on identifying and classifying student errors has focused primarily on syntax errors that are detected at compile time, perhaps because static analysis can easily be applied to large data sets [1]. In particular, recent work by Becker [3–5], Denny [7, 8],

inputs. Ahmadzadeh et al. [1] state that such errors are often when the intended meaning of the programmer's code is inconsistent with the language. Debugging logic errors can be frustrating activity for programmers, as often there is no available feedback on the location or nature of the error. Novice programmers may find it particularly difficult to detect logic errors to their relative lack of debugging experience combined with subtle misconceptions in their programming knowledge [9].

In this paper we investigate the following two research questions: RQ1: What are the most common logic errors students make? RQ2: Which logic errors are most problematic for students to identify and fix?

To identify student errors we analysed data taken from a first-year university programming course. Our goal in this study is to highlight some of the challenges that students encounter when learning to program which may inform future teaching practice.

#### 2 RELATED WORK

Hristova et al. [10] found improper casting, not storing data from a call to a non-void method, and flow reaching the end of a non-void method were among the most commonly made errors in Java. This data was collected from a survey of professors and teaching assistants within Computer Science departments and their students. The study found that novices categorised errors in student code through manual analysis of relying on compiler messages. Although they did not provide a detailed breakdown of the most common logic errors, syntax errors were categorised.

Altman and Brown [2] provided a comprehensive analysis of the frequency and time to fix of different error types including semantic errors for over 250,000 novice programmers. They found that the average time students spent

# Example

- Liam Rigby's paper
  - COMPSCI747 project
  - Australasian Computing Education Conference (ACE) 2020
  - Dynamic analysis of large existing code dataset

## A Miss is as Good as a Mile: Off-By-One Errors and Arrays in an Introductory Programming Course

Liam Rigby  
The University of Auckland  
New Zealand  
lrig521@aucklanduni.ac.nz

Paul Denny  
The University of Auckland  
New Zealand  
paul@cs.auckland.ac.nz

Andrew Luxton-Reilly  
The University of Auckland  
New Zealand  
a.luxton-reilly@auckland.ac.nz

### ABSTRACT

Loops and arrays are fundamental CS1 concepts, but ones that can be problematic for novice programmers. In this research, we investigate *off-by-one* errors – logic errors where loops perform one too few or one too many iterations – in code using an indexed loop over an array. We classify off-by-one errors, and explore the prevalence of each type, by analyzing a large set of code submissions from students in a first year programming course as they tackle a sequence of exercises. We describe an approach to reliably identify off-by-one errors through dynamic analysis, and find that off-by-one errors are both common and persist across exercises. We also show that students infrequently choose to iterate over an array in reverse, but when they do they more commonly encounter off-by-one errors. We conclude that teaching material should explicitly focus student attention on boundary cases, and should provide more examples that iterate through arrays in reverse.

### LowMiss

(first element, at index 0, is missed)

```
for (i = 1; i < length; i++) {  
    sum += array[i];  
}
```

### HighMiss

(last element, at index *length-1*, is missed)

```
for (i = 0; i < length - 1; i++) {  
    sum += array[i];  
}
```

### LOWBOUNDS

(invalid element, at index -1, is accessed)

```
for (i = -1; i < length; i++) {  
    sum += array[i];  
}
```



# Example

- Liam Rigby's paper
  - COMPSCI747 project
  - Australasian Computing Education Conference (ACE) 2020
  - Dynamic analysis of large existing code dataset

was reviewing recently...

report arrays and indexing problems, noting on-by-one errors and difficulties setting up the appropriate range. Rigby et al [25] examine off-by-one-errors in which students make logic errors resulting in loops performing too few or too many iterations, and find that such errors are both common and persist across multiple types of exercises. These difficulties have led some computing education researchers to argue for the use of collection objects and their

177  
178  
179  
180  
181  
182  
183  
184  
185

# Example

- Liam's paper
  - CS747 project
  - Published in ACE 2
  - Analysed an existin

was reviewing the  
report arrays and  
difficulties setting  
off-by-one-errors  
loops performin  
such errors are b  
exercises. These  
researchers to a



Liam Rigby

Re: First citation (maybe)



You replied to this message on 28/04/2020 12:16 PM.

Paul Denny



# An exciting time

- Rapid growth in adoption of learning tools
- Vast amounts of data collected on how people learn



**Anant Agarwal** (MIT / EdX)

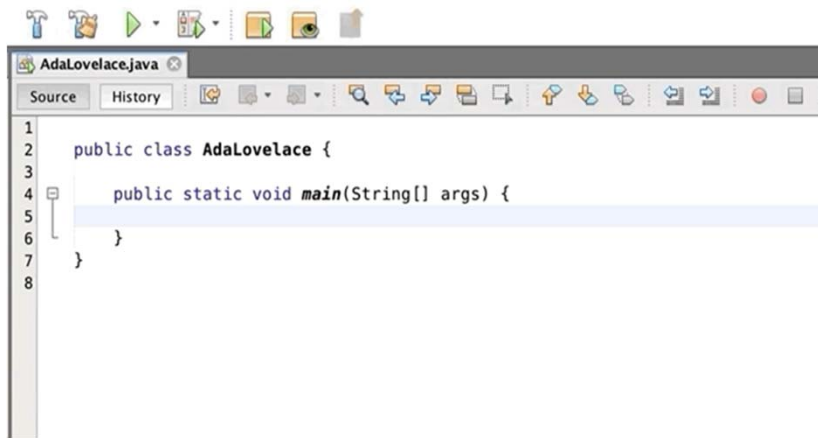
"Data collection for educational research is one of the key goals of EdX.... we gather huge amounts of data.... all this rich data will be available to researchers.... to understand how people really learn and we can help synthesize a better educational experience"



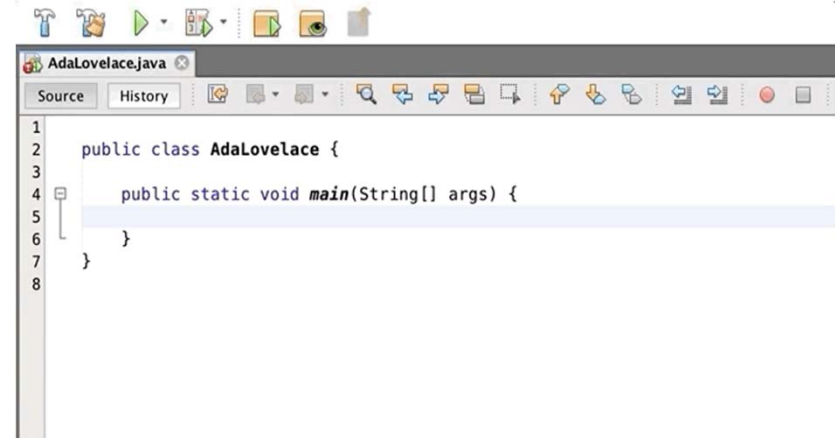
**Daphne Koller** (Stanford / Coursera)

"Tremendous opportunities.... every click, every homework submission, every forum post, from tens of thousands of students.... turn the study of human learning from the hypothesis driven mode to the data driven mode"

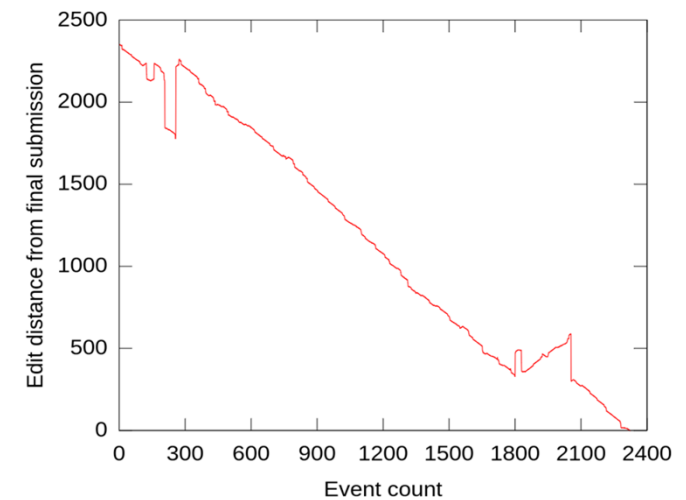
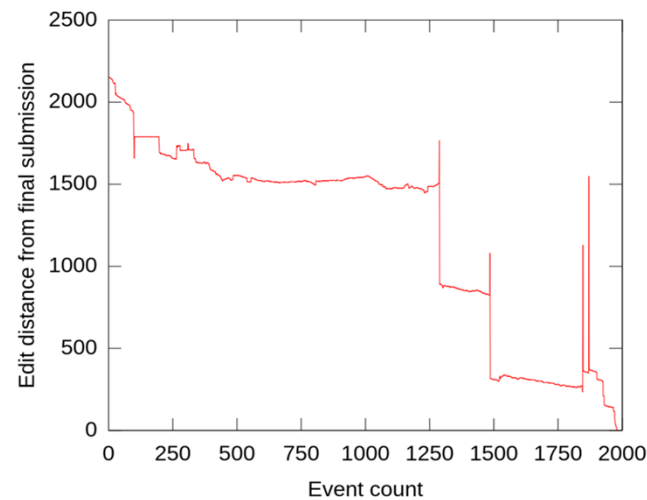
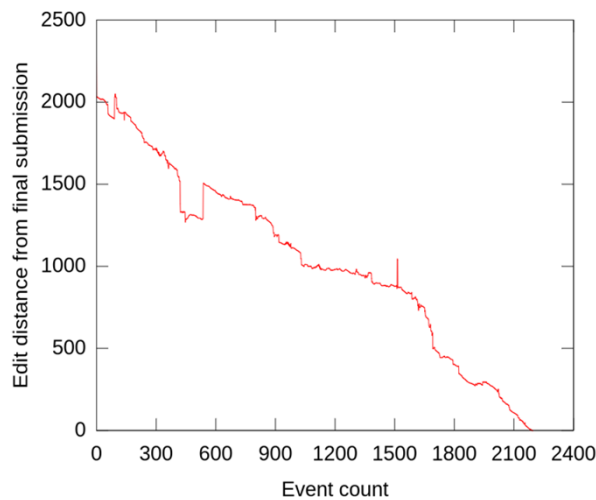
# Example: keystroke vs submission data



```
1 public class AdaLovelace {
2
3
4     public static void main(String[] args) {
5
6     }
7 }
8
```



```
1 public class AdaLovelace {
2     public class AdaLovelace {
3
4     public static void main(String[] args) {
5
6     }
7 }
8
```



Courtesy: Juho Leinonen (University of Helsinki)

# Computing Education Research

- Overarching research questions:
  - How do people learn computing?
  - How do teachers teach and assess computing?
  - How can people learn computing more effectively?
  - How can teachers teach computing more effectively?
  - How can access to computing education be improved?
  - How can computing education be delivered equitably to all?
  - How can learning technologies teach computing?
  - How does computing education affect people's lives?
  - What are the societal costs of computing illiteracy?
  - What does it mean to know computing?
  - What can be taught about computing to learners of different ages?



# Two examples

- Two recent studies
- Feedback and learner behavior
  - Example 1
    - Computing education: *Compiler error messages*
  - Example 2
    - Learning technology: *Influencing (positive) behaviours*

# Example 1: Compiler error messages

## Error Message Readability and Novice Debugging Performance

Paul Denny  
University of Auckland  
Auckland, New Zealand  
paul@cs.auckland.ac.nz

James Prather  
Abilene Christian University  
Abilene, Texas, USA  
james.prather@acu.edu

Brett A. Becker  
University College Dublin  
Dublin, Ireland  
brett.becker@ucd.ie

### ABSTRACT

It is well known that programming error messages can be notoriously difficult for novices to understand, hampering progress and leading to frustration. In response, researchers have explored various approaches for enhancing such messages, yet results from this active strand of research are currently mixed. Direct comparisons of results between studies is challenging as these typically investigate different kinds of message enhancements and report results using different metrics. In addition, many prior studies have involved code writing tasks. In such cases, not all students encounter the same errors and messages, and it is difficult to isolate the time spent interpreting messages and resolving errors from the time spent writing code. In this research, we explore the effects of presenting novices with compiler error messages designed using the most recent collection of published guidelines – specifically, more easily readable, short, positive messages containing resolution hints. To accurately determine the time and effort required to read and respond to the messages, we utilise a debugging task where all students are presented the same code and therefore encounter the same errors. We present results of a randomised controlled experiment ( $n > 700$ ) which shows that, compared to standard error messages, the messages we tested resulted in significantly shorter debugging times and higher self-reported scores of message usefulness for students in the very early stages of learning a new language.

### CCS CONCEPTS

• Social and professional topics → CS1: Computing education; Computer science education; • Human-centered computing → Human computer interaction (HCI).

### KEYWORDS

compiler error messages; CS1; debugging; error message enhancement; novice programmers; programming error messages; readability

### ACM Reference Format:

Paul Denny, James Prather, and Brett A. Becker. 2020. Error Message Readability and Novice Debugging Performance. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '20)*, June 15–19, 2020, Trondheim, Norway. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/3341525.3387384>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).  
ITiCSE '20, June 15–19, 2020, Trondheim, Norway  
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISSN 978-1-4503-6874-2/20/06...\$15.00  
<https://doi.org/10.1145/3341525.3387384>

### 1 INTRODUCTION

One of the challenges faced by any programmer new to a language is understanding the error messages produced by the compiler or interpreter. For students who are also new to programming, learning the syntax of a language alongside the principles of programming can be difficult, particularly when the messages they receive are confusing [17, 46]. Many educators will be familiar with the reality that certain error messages in the language they are teaching lead to particularly cryptic error messages which can be frustrating for students and hamper progress [5, 8].

In recent years there has been growing interest in understanding how students respond to various error messages and how those messages relate to underlying errors in code [7, 29, 37, 41, 44]. However, it is still widely accepted that for many languages, there is much room for improvement in the usefulness of error messages – particularly when concerning novice users [36].

A 2019 ITiCSE Working Group conducted a large-scale review of the research on programming error messages (PEMs) [6]. This report composed a list of design guidelines for improving text-based messages, based on those proposed by various researchers over the past 60 years. The guidelines were classified into ten categories, which included the following four: increase readability, reduce cognitive load, use a positive tone, and show solutions or hints. The Working Group report concluded with a call for additional research to empirically validate the guidelines for producing useful error messages: "Individual guidelines should be examined and then robustly tested to determine their effectiveness" [6, p. 204].

In this research we apply these guidelines to formulate new error messages for a select set of errors. We then measure the effect that these new messages have on students as they work through a simple debugging task. In particular, we explore how students perceive the usefulness of the new messages and how the new messages impact their debugging efforts. We answer the following research questions with respect to the newly formulated error messages:

**RQ1:** *Are the new error messages more readable, using traditional measures of readability, than the corresponding original compiler error messages?*

**RQ2:** *Do students read error messages and, if so, are the new messages perceived as more useful when debugging code in a new language, compared to the messages produced by the compiler?*

**RQ3:** *To what extent do the new messages impact debugging performance, in terms of time and effort?*

Denny, Prather & Becker (2020)

# Example 1: Compiler error messages

## Error Message Readability and Novice Debugging Performance

Paul Denny  
University of Auckland  
Auckland, New Zealand  
paul@cs.auckland.ac.nz

James Prather  
Abilene Christian University  
Abilene, Texas, USA  
james.prather@acu.edu

Brett A. Becker  
University College Dublin  
Dublin, Ireland  
brett.becker@ucd.ie

### ABSTRACT

It is well known that programming error messages can be notoriously difficult for novices to understand, hampering progress and leading to frustration. In response, researchers have explored various approaches for enhancing such messages, yet results from this active strand of research are currently mixed. Direct comparisons of results between studies is challenging as these typically investigate different kinds of message enhancements and report results using different metrics. In addition, many prior studies have involved code writing tasks. In such cases, not all students encounter the same errors and messages, and it is difficult to isolate the time spent interpreting messages and resolving errors from the time spent writing code. In this research, we explore the effects of presenting novices with compiler error messages designed using the most recent collection of published guidelines – specifically, more easily readable, short, positive messages containing resolution hints. To accurately determine the time and effort required to read and respond to the messages, we utilise a debugging task where all students are presented the same code and therefore encounter the same errors. We present results of a randomised controlled experiment ( $n > 700$ ) which shows that, compared to standard error messages, the messages we tested resulted in significantly shorter debugging times and higher self-reported scores of message usefulness for students in the very early stages of learning a new language.

### CCS CONCEPTS

• Social and professional topics → CS1: Computing education; Computer science education; • Human-centered computing → Human computer interaction (HCI).

### KEYWORDS

compiler error messages; CS1; debugging; error message enhancement; novice programmers; programming error messages; readability

### ACM Reference Format:

Paul Denny, James Prather, and Brett A. Becker. 2020. Error Message Readability and Novice Debugging Performance. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education (ITICSE '20)*, June 15–19, 2020, Trondheim, Norway. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/3341525.3387384>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).  
ITICSE '20, June 15–19, 2020, Trondheim, Norway  
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISSN 978-1-4503-6874-2/20/06...\$15.00  
<https://doi.org/10.1145/3341525.3387384>

### 1 INTRODUCTION

One of the challenges faced by any programmer new to a language is understanding the error messages produced by the compiler or interpreter. For students who are also new to programming, learning the syntax of a language alongside the principles of programming can be difficult, particularly when the messages they receive are confusing [17, 46]. Many educators will be familiar with the reality that certain error messages in the language they are teaching lead to particularly cryptic error messages which can be frustrating for students and hamper progress [5, 8].

In recent years there has been growing interest in understanding how students respond to various error messages and how those messages relate to underlying errors in code [7, 29, 37, 41, 44]. However, it is still widely accepted that for many languages, there is much room for improvement in the usefulness of error messages – particularly when concerning novice users [36].

A 2019 ITICSE Working Group conducted a large-scale review of the research on programming error messages (PEMs) [6]. This report composed a list of design guidelines for improving text-based messages, based on those proposed by various researchers over the past 60 years. The guidelines were classified into ten categories, which included the following four: increase readability, reduce cognitive load, use a positive tone, and show solutions or hints. The Working Group report concluded with a call for additional research to empirically validate the guidelines for producing useful error messages: "Individual guidelines should be examined and then robustly tested to determine their effectiveness" [6, p. 204].

In this research we apply these guidelines to formulate new error messages for a select set of errors. We then measure the effect that these new messages have on students as they work through a simple debugging task. In particular, we explore how students perceive the usefulness of the new messages and how the new messages impact their debugging efforts. We answer the following research questions with respect to the newly formulated error messages:

**RQ1:** Are the new error messages more readable, using traditional measures of readability, than the corresponding original compiler error messages?

**RQ2:** Do students read error messages and, if so, are the new messages perceived as more useful when debugging code in a new language, compared to the messages produced by the compiler?

**RQ3:** To what extent do the new messages impact debugging performance, in terms of time and effort?

## II. Background and Objectives

Manufacturer-supplied FORTRAN compilers normally provide rather efficient object code, provide flexible interaction with the operating systems, and have many sophisticated programming features. However, they are inadequate for the needs presented in the area of finding and correcting errors as quickly as possible. In many instances, the description of an error condition lacks resolution and offers the user little assistance in removing the error other than indicating the statement in which the error occurs. A more serious inadequacy is that many error descriptions are given in terms not understandable to a FORTRAN programmer.

DITRAN—a compiler emphasizing diagnostics  
Moulton and Muller  
Communications of the ACM  
January 1967

Denny, Prather & Becker (2020)

# Example 1: Compiler error messages

## Error Message Readability and Novice Debugging Performance

Paul Denny  
University of Auckland  
Auckland, New Zealand  
paul@cs.auckland.ac.nz

James Prather  
Abilene Christian University  
Abilene, Texas, USA  
james.prather@acu.edu

Brett A. Becker  
University College Dublin  
Dublin, Ireland  
brett.becker@ucd.ie

### ABSTRACT

It is well known that programming error messages can be notoriously difficult for novices to understand, hampering progress and leading to frustration. In response, researchers have explored various approaches for enhancing such messages, yet results from this active strand of research are currently mixed. Direct comparisons of results between studies is challenging as these typically investigate different kinds of message enhancements and report results using different metrics. In addition, many prior studies have involved code writing tasks. In such cases, not all students encounter the same errors and messages, and it is difficult to isolate the time spent interpreting messages and resolving errors from the time spent writing code. In this research, we explore the effects of presenting novices with compiler error messages designed using the most recent collection of published guidelines – specifically, more easily readable, short, positive messages containing resolution hints. To accurately determine the time and effort required to read and respond to the messages, we utilise a debugging task where all students are presented the same code and therefore encounter the same errors. We present results of a randomised controlled experiment ( $n > 700$ ) which shows that, compared to standard error messages, the messages we tested resulted in significantly shorter debugging times and higher self-reported scores of message usefulness for students in the very early stages of learning a new language.

### CCS CONCEPTS

• **Social and professional topics** → **CS1: Computing education; Computer science education; Human-centered computing** → **Human computer interaction (HCI).**

### KEYWORDS

compiler error messages; CS1; debugging; error message enhancement; novice programmers; programming error messages; readability

### ACM Reference Format:

Paul Denny, James Prather, and Brett A. Becker. 2020. Error Message Readability and Novice Debugging Performance. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education (ITICSE '20)*, June 15–19, 2020, Trondheim, Norway. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/3341525.3387384>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).  
ITICSE '20, June 15–19, 2020, Trondheim, Norway  
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISSN 978-1-4503-6874-2/20/06...\$15.00  
<https://doi.org/10.1145/3341525.3387384>

### 1 INTRODUCTION

One of the challenges faced by any programmer new to a language is understanding the error messages produced by the compiler or interpreter. For students who are also new to programming, learning the syntax of a language alongside the principles of programming can be difficult, particularly when the messages they receive are confusing [17, 46]. Many educators will be familiar with the reality that certain error messages in the language they are teaching lead to particularly cryptic error messages which can be frustrating for students and hamper progress [5, 8].

In recent years there has been growing interest in understanding how students respond to various error messages and how those messages relate to underlying errors in code [7, 29, 37, 41, 44]. However, it is still widely accepted that for many languages, there is much room for improvement in the usefulness of error messages – particularly when concerning novice users [36].

A 2019 ITICSE Working Group conducted a large-scale review of the research on programming error messages (PEMs) [6]. This report composed a list of design guidelines for improving text-based messages, based on those proposed by various researchers over the past 60 years. The guidelines were classified into ten categories, which included the following four: increase readability, reduce cognitive load, use a positive tone, and show solutions or hints. The Working Group report concluded with a call for additional research to empirically validate the guidelines for producing useful error messages: "Individual guidelines should be examined and then robustly tested to determine their effectiveness" [6, p. 204].

In this research we apply these guidelines to formulate new error messages for a select set of errors. We then measure the effect that these new messages have on students as they work through a simple debugging task. In particular, we explore how students perceive the usefulness of the new messages and how the new messages impact their debugging efforts. We answer the following research questions with respect to the newly formulated error messages:

**RQ1:** *Are the new error messages more readable, using traditional measures of readability, than the corresponding original compiler error messages?*

**RQ2:** *Do students read error messages and, if so, are the new messages perceived as more useful when debugging code in a new language, compared to the messages produced by the compiler?*

**RQ3:** *To what extent do the new messages impact debugging performance, in terms of time and effort?*

Yet, ask any experienced programmer about the quality of error messages in their programming environments, and you will often get an embarrassed laugh. In every environment, a mature programmer can usually point to at least a handful of favourite bad error responses. When they find out that the same environment is being used by novices, their laugh often hardens.

Marceau, Fisler & Krishnamurthi (2011)

Denny, Prather & Becker (2020)



# Example 1: Compiler error messages

## Error Message Readability and Novice Debugging Performance

Paul Denny  
University of Auckland  
Auckland, New Zealand  
paul@cs.auckland.ac.nz

James Prather  
Abilene Christian University  
Abilene, Texas, USA  
james.prather@acu.edu

Brett A. Becker  
University College Dublin  
Dublin, Ireland  
brett.becker@ucd.ie

### ABSTRACT

It is well known that programming error messages can be notoriously difficult for novices to understand, hampering progress and leading to frustration. In response, researchers have explored various approaches for enhancing such messages, yet results from this active strand of research are currently mixed. Direct comparisons of results between studies is challenging as these typically investigate different kinds of message enhancements and report results using different metrics. In addition, many prior studies have involved code writing tasks. In such cases, not all students encounter the same errors and messages, and it is difficult to isolate the time spent interpreting messages and resolving errors from the time spent writing code. In this research, we explore the effects of presenting novices with compiler error messages designed using the most recent collection of published guidelines – specifically, more easily readable, short, positive messages containing resolution hints. To accurately determine the time and effort required to read and respond to the messages, we utilise a debugging task where all students are presented the same code and therefore encounter the same errors. We present results of a randomised controlled experiment ( $n > 700$ ) which shows that, compared to standard error messages, the messages we tested resulted in significantly shorter debugging times and higher self-reported scores of message usefulness for students in the very early stages of learning a new language.

### CCS CONCEPTS

• Social and professional topics → CS1: Computing education; Computer science education; • Human-centered computing → Human computer interaction (HCI).

### KEYWORDS

compiler error messages; CS1; debugging; error message enhancement; novice programmers; programming error messages; readability

### ACM Reference Format:

Paul Denny, James Prather, and Brett A. Becker. 2020. Error Message Readability and Novice Debugging Performance. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education (ITICSE '20)*, June 15–19, 2020, Trondheim, Norway. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/3341525.3387384>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).  
ITICSE '20, June 15–19, 2020, Trondheim, Norway  
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISSN 978-1-4503-6874-2/20/06...\$15.00  
<https://doi.org/10.1145/3341525.3387384>

### 1 INTRODUCTION

One of the challenges faced by any programmer new to a language is understanding the error messages produced by the compiler or interpreter. For students who are also new to programming, learning the syntax of a language alongside the principles of programming can be difficult, particularly when the messages they receive are confusing [17, 46]. Many educators will be familiar with the reality that certain error messages in the language they are teaching lead to particularly cryptic error messages which can be frustrating for students and hamper progress [5, 8].

In recent years there has been growing interest in understanding how students respond to various error messages and how those messages relate to underlying errors in code [7, 29, 37, 41, 44]. However, it is still widely accepted that for many languages, there is much room for improvement in the usefulness of error messages – particularly when concerning novice users [36].

A 2019 ITICSE Working Group conducted a large-scale review of the research on programming error messages (PEMs) [6]. This report composed a list of design guidelines for improving text-based messages, based on those proposed by various researchers over the past 60 years. The guidelines were classified into ten categories, which included the following four: increase readability, reduce cognitive load, use a positive tone, and show solutions or hints. The Working Group report concluded with a call for additional research to empirically validate the guidelines for producing useful error messages: "Individual guidelines should be examined and then robustly tested to determine their effectiveness" [6, p. 204].

In this research we apply these guidelines to formulate new error messages for a select set of errors. We then measure the effect that these new messages have on students as they work through a simple debugging task. In particular, we explore how students perceive the usefulness of the new messages and how the new messages impact their debugging efforts. We answer the following research questions with respect to the newly formulated error messages:

**RQ1:** Are the new error messages more readable, using traditional measures of readability, than the corresponding original compiler error messages?

**RQ2:** Do students read error messages and, if so, are the new messages perceived as more useful when debugging code in a new language, compared to the messages produced by the compiler?

**RQ3:** To what extent do the new messages impact debugging performance, in terms of time and effort?

The screenshot shows the Clang Expressive Diagnostics website. The left sidebar contains navigation links: LLVM Home, Clang Info (Download, About, Features, Related Projects, User's Manual, Language Compatibility, Language Extensions, C++ Status), Clang Development (Get Started, Get Involved, Open Projects, Clang Internals, Hacking on Clang), Clang Tools (Automatic Bug-Finding, Writing Clang Tools), Communication (cfe-users List, cfe-dev List, cfe-commits List, Bug Reports, Planet Clang, IRC, irc.freenet.org), The Code (Check Out Sources, Browse Sources, doxygen), Quick Links (Testing Coverage), Clang Events, and LLVM Meeting.

### Expressive Diagnostics

In addition to being fast and functional, we aim to make Clang extremely user friendly. As far as a command-line compiler goes, this basically boils down to making the diagnostics (error and warning messages) generated by the compiler be as useful as possible. There are several ways that we do this. This section talks about the experience provided by the command line compiler, contrasting Clang output to GCC 4.9's output in some cases.

### Column Numbers and Caret Diagnostics

First, all diagnostics produced by clang include full column number information. The clang command-line compiler driver uses this information to print "point diagnostics". (IDEs can use the information to display in-line error markup.) This is nice because it makes it very easy to understand exactly what is wrong in a particular piece of code.

The point (the green "^" character) exactly shows where the problem is, even inside of a string. This makes it really easy to jump to the problem and helps when multiple instances of the same character occur on a line. (We'll revisit this more in following examples.)

```
$ clang -fsyntax-only format-strings.c
format-strings.c:91:13: warning: '.' specified field precision is missing a matching 'int' argument
printf("%d.%d");
               ^
```

Note that modern versions of GCC have followed Clang's lead, and are now able to give a column for a diagnostic, and include a snippet of source text in the result. However, Clang's column number is much more accurate, pointing at the problematic format specifier, rather than the ) character the parser had reached when the problem was detected. Also, Clang's diagnostic is colored by default, making it easier to distinguish from nearby text.

### Range Highlighting for Related Text

Clang captures and accurately tracks range information for expressions, statements, and other constructs in your program and uses this to make diagnostics highlight related information. In the following somewhat nonsensical example you can see that you don't even need to see the original source code to understand what is wrong based on the Clang error. Because clang prints a point, you know exactly which plus it is complaining about. The range information highlights the left and right side of the plus which makes it immediately obvious what the compiler is talking about. Range information is very useful for cases involving precedence issues and many other cases.

```
$ gcc-4.9 -fsyntax-only t.c
t.c: In function 'int f(int, int)':
t.c:7:19: error: invalid operands to binary + (have 'int' and 'struct A')
return y + func(y ? ((SomeA.X + 40) + SomeA) / 42 + SomeA.X : SomeA.X);
               ^
$ clang -fsyntax-only t.c
t.c:7:19: error: invalid operands to binary expression ('int' and 'struct A')
return y + func(y ? ((SomeA.X + 40) + SomeA) / 42 + SomeA.X : SomeA.X);
               ~~~~~^~~~~~
```

### Precision in Wording

A detail is that we have tried really hard to make the diagnostics that come out of clang contain exactly the pertinent information about what is wrong and why. In the example above, we tell you what the inferred types are for the left and right hand sides, and we don't repeat what is obvious from the point (e.g., that this is a "binary +").

Many other examples abound. In the following example, not only do we tell you that there is a problem with the \* and point to it, we say exactly why and tell you what the type is (in case it is a complicated subexpression, such as a call to an overloaded function). This sort of attention to detail makes it much easier to understand and fix problems quickly.

```
$ gcc-4.9 -fsyntax-only t.c
t.c:6:11: error: invalid type argument of unary '*' (have 'int')
return *SomeA.X;
       ^
$ clang -fsyntax-only t.c
```

<https://clang.llvm.org/diagnostics.html>

Denny, Prather & Becker (2020)

# Example 1: Compiler error messages

```
1  #include <stdio.h>
2
3  #define CENTIMETERS_TO_FEET 0.0328
4  #define CENTIMETERS_TO_INCH 0.3937
5
6  int main(void)
7
8      // Variables for converting metric to imperial
9      int centimeters feet;
10     double inches;
11
12     // Read value into the variable centimeters
13     scanf("%d", centimeters);
14
15     feet = centimeters * CENTIMETERS_TO_FEET;
16     inches = (centimeters - feet / CENTIMETERS_TO_FEET) * CENTIMETERS_TO_INCH;
17     printf("%d centimeters is %d feet and %.2f inches\n", centimeters, feet, inches);
18
19     return 0;
20 }
```

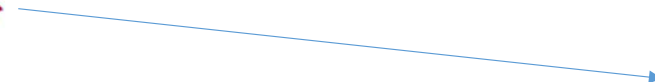
# Example 1: Compiler error messages

```
1  #include <studio.h>
2
3  #define CENTIMETERS_TO_FEET 0.0328
4  #define CENTIMETERS_TO_INCH 0.3937
5
6  int main(void)
7
8      // Variables for converting metric to imperial
9      int centimeters feet;
10     double inches;
11
12     // Read value into the variable centimeters
13     scanf("%d", centimeters);
14
15     feet = centimeters * CENTIMETERS_TO_FEET;
16     inches = (centimeters - feet / CENTIMETERS_TO_FEET) * CENTIMETERS_TO_INCH;
17     printf("%d centimeters is %d feet and %.2f inches\n", centimeters, feet, inches);
18
19     return 0;
20 }
```

**1:10: fatal error: studio.h:  
No such file or directory**

# Example 1: Compiler error messages

```
1  #include <studio.h>
2
3  #define CENTIMETERS_TO_FEET 0.0328
4  #define CENTIMETERS_TO_INCH 0.3937
5
6  int main(void)
7
8      // Variables for converting metric to imperial
9      int centimeters feet;
10     double inches;
11
12     // Read value into the variable centimeters
13     scanf("%d", centimeters);
14
15     feet = centimeters * CENTIMETERS_TO_FEET;
16     inches = (centimeters - feet / CENTIMETERS_TO_FEET) * CENTIMETERS_TO_INCH;
17     printf("%d centimeters is %d feet and %.2f inches\n", centimeters, feet, inches);
18
19     return 0;
20 }
```



**1:10: fatal error: studio.h:  
No such file or directory**



# Example 1: Compiler error messages

```
1  #include <stdio.h>
2
3  #define CENTIMETERS_TO_FEET 0.0328
4  #define CENTIMETERS_TO_INCH 0.3937
5
6  int main(void)
7
8      // Variables for converting metric to imperial
9      int centimeters feet;
10     double inches;
11
12     // Read value into the variable centimeters
13     scanf("%d", centimeters);
14
15     feet = centimeters * CENTIMETERS_TO_FEET;
16     inches = (centimeters - feet / CENTIMETERS_TO_FEET) * CENTIMETERS_TO_INCH;
17     printf("%d centimeters is %d feet and %.2f inches\n", centimeters, feet, inches);
18
19     return 0;
20 }
```

**9:21: error: expected '=', ';;', 'asm' or '\_\_attribute\_\_' before 'feet'.**

# Example 1: Compiler error messages

```
1  #include <stdio.h>
2
3  #define CENTIMETERS_TO_FEET 0.0328
4  #define CENTIMETERS_TO_INCH 0.3937
5
6  int main(void)
7
8      // Variables for converting metric to imperial
9      int centimeters feet;
10     double inches;
11
12     // Read value into the variable centimeters
13     scanf("%d", centimeters);
14
15     feet = centimeters * CENTIMETERS_TO_FEET;
16     inches = (centimeters - feet / CENTIMETERS_TO_FEET) * CENTIMETERS_TO_INCH;
17     printf("%d centimeters is %d feet and %.2f inches\n", centimeters, feet, inches);
18
19     return 0;
20 }
```

**9:21: error: expected '=', ';;', 'asm' or '\_\_attribute\_\_' before 'feet'.**

# Example 1: Compiler error messages

```
1  #include <stdio.h>
2
3  #define CENTIMETERS_TO_FEET 0.0328
4  #define CENTIMETERS_TO_INCH 0.3937
5
6  int main(void)
7
8      // Variables for converting metric to imperial
9      int centimeters feet;
10     double inches;
11
12     // Read value into the variable centimeters
13     scanf("%d", centimeters);
14
15     feet = centimeters * CENTIMETERS_TO_FEET;
16     inches = (centimeters - feet / CENTIMETERS_TO_FEET) * CENTIMETERS_TO_INCH;
17     printf("%d centimeters is %d feet and %.2f inches\n", centimeters, feet, inches);
18
19     return 0;
20 }
```

**13:5: error: expected  
declaration specifiers before  
'scanf'**

# Example 1: Compiler error messages

```
1  #include <stdio.h>
2
3  #define CENTIMETERS_TO_FEET 0.0328
4  #define CENTIMETERS_TO_INCH 0.3937
5
6  int main(void)
7
8      // Variables for converting metric to imperial
9      int centimeters feet;
10     double inches;
11
12     // Read value into the variable centimeters
13     scanf("%d", centimeters);
14
15     feet = centimeters * CENTIMETERS_TO_FEET;
16     inches = (centimeters - feet / CENTIMETERS_TO_FEET) * CENTIMETERS_TO_INCH;
17     printf("%d centimeters is %d feet and %.2f inches\n", centimeters, feet, inches);
18
19     return 0;
20 }
```

**13:5: error: expected  
declaration specifiers before  
'scanf'**

# Example 1: Compiler error messages

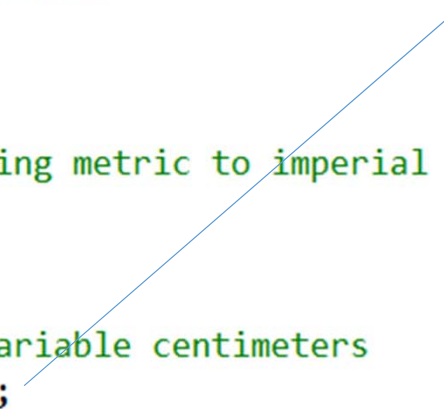
```
1  #include <stdio.h>
2
3  #define CENTIMETERS_TO_FEET 0.0328
4  #define CENTIMETERS_TO_INCH 0.3937
5
6  int main(void)
7
8      // Variables for converting metric to imperial
9      int centimeters feet;
10     double inches;
11
12     // Read value into the variable centimeters
13     scanf("%d", centimeters);
14
15     feet = centimeters * CENTIMETERS_TO_FEET;
16     inches = (centimeters - feet / CENTIMETERS_TO_FEET) * CENTIMETERS_TO_INCH;
17     printf("%d centimeters is %d feet and %.2f inches\n", centimeters, feet, inches);
18
19     return 0;
20 }
```

**13:13: error: format '%d' expects argument of type 'int \*', but argument 2 has type 'int'.**

# Example 1: Compiler error messages

```
1  #include <stdio.h>
2
3  #define CENTIMETERS_TO_FEET 0.0328
4  #define CENTIMETERS_TO_INCH 0.3937
5
6  int main(void)
7
8      // Variables for converting metric to imperial
9      int centimeters;
10     double inches;
11
12     // Read value into the variable centimeters
13     scanf("%d", centimeters);
14
15     feet = centimeters * CENTIMETERS_TO_FEET;
16     inches = (centimeters - feet / CENTIMETERS_TO_FEET) * CENTIMETERS_TO_INCH;
17     printf("%d centimeters is %d feet and %.2f inches\n", centimeters, feet, inches);
18
19     return 0;
20 }
```

**13:13: error: format '%d' expects argument of type 'int \*', but argument 2 has type 'int'.**





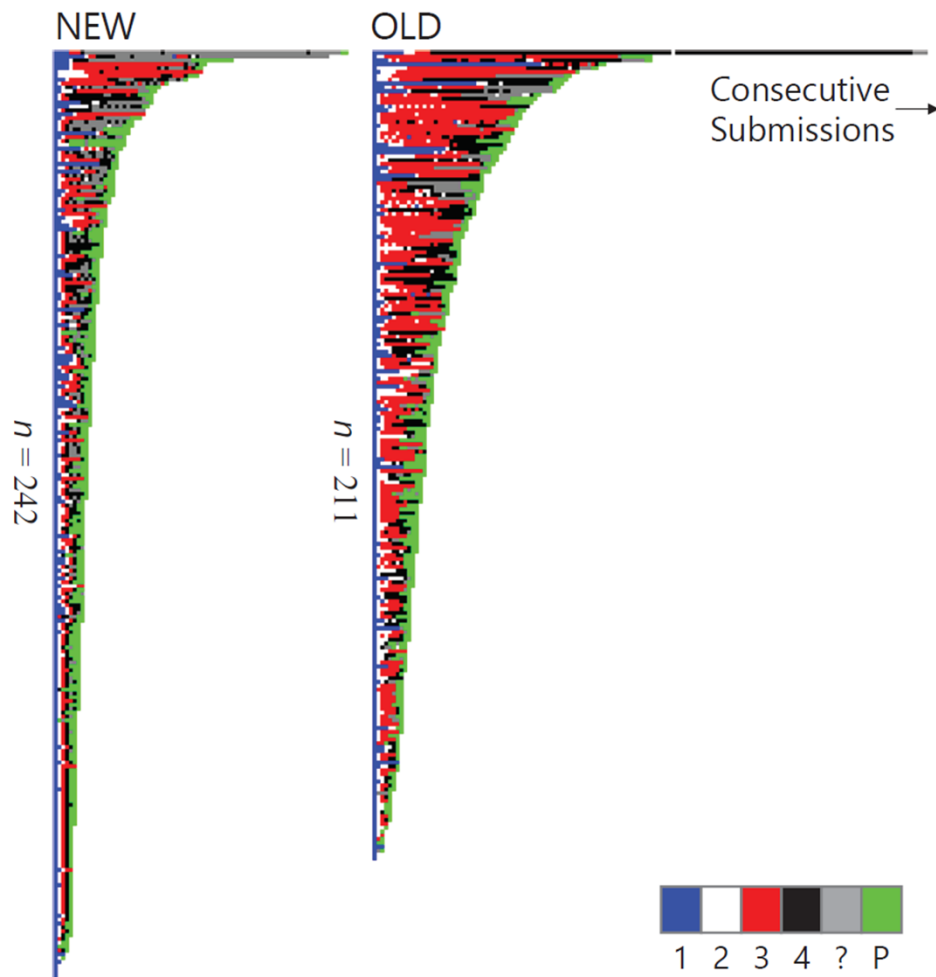
# Example 1: Compiler error messages

```
1  #include <stdio.h>
2
3  #define CENTIMETERS_TO_FEET 0.0328
4  #define CENTIMETERS_TO_INCH 0.3937
5
6  int main(void)
7
8      // Variables for converting metric to imperial
9      int centimeters;
10     double inches;
11
12     // Read value into the variable centimeters
13     scanf("%d", &centimeters);
14
15     feet = centimeters * CENTIMETERS_TO_FEET;
16     inches = (centimeters - feet / CENTIMETERS_TO_FEET) * CENTIMETERS_TO_INCH;
17     printf("%d centimeters is %d feet and %.2f inches\n", centimeters, feet, inches);
18
19     return 0;
20 }
```

**Could we provide more useful error messages?**

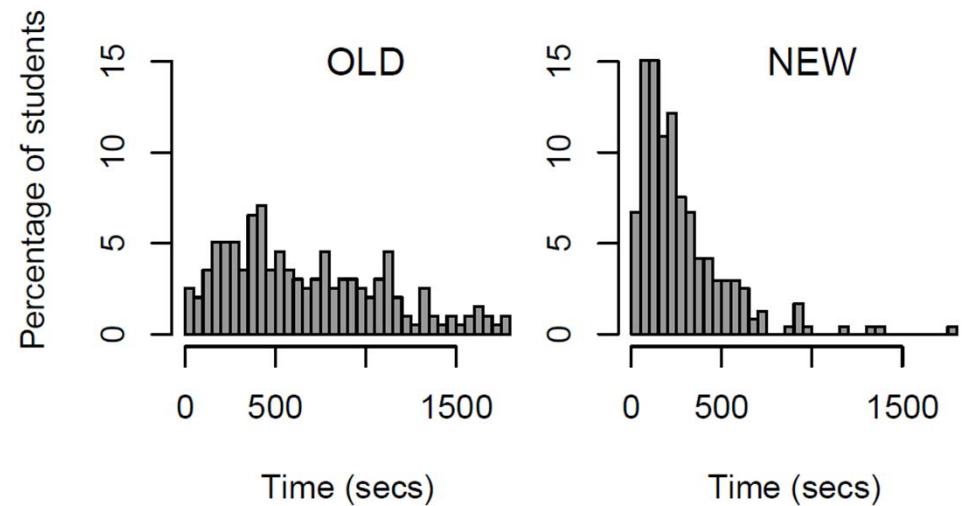
**What sort of difference would that make?**

# Example 1: Compiler error messages



**Could we provide more useful error messages?**

**What sort of difference would that make?**





# Example 2: Influencing (positive) behaviours

CHI 2018 Honourable Mention

CHI 2018, April 21–26, 2018, Montréal, QC, Canada

## Empirical Support for a Causal Relationship Between Gamification and Learning Outcomes

Paul Denny  
University of Auckland  
Auckland, New Zealand  
paul@cs.auckland.ac.nz

Fiona McDonald,  
Ruth Empson, Philip Kelly  
University of Otago  
Dunedin, New Zealand  
{fiona.mcdonald, ruth.empson,  
philip.kelly}@otago.ac.nz

Andrew Petersen  
University of Toronto  
Mississauga, Canada  
andrew.petersen@utoronto.ca

### ABSTRACT

Preparing for exams is an important yet stressful time for many students. Self-testing is known to be an effective preparation strategy, yet some students lack motivation to engage or persist in self-testing activities. Adding game elements to a platform supporting self-testing may increase engagement and, by extension, exam performance. We conduct a randomized controlled experiment ( $n=701$ ) comparing the effect of two game elements – a points system and a badge system – used individually and in combination.

We find that the badge system elicits significantly higher levels of voluntary self-testing activity and this effect is particularly pronounced amongst a relatively small cohort. Importantly, this increased activity translates to a significant improvement in exam scores. Our data supports a causal relationship between gamification and learning outcomes, mediated by self-testing behavior. This provides empirical support for Landers' theory of gamified learning when the gamified activity is conducted prior to measuring learning outcomes.

### ACM Classification Keywords

H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous; K.3.0 Computers and Education: General

### Author Keywords

gamification; points; badges; self-testing; PeerWise

### INTRODUCTION

A growing number of online platforms are incorporating game-like elements to motivate users and generate higher levels of activity. Commonly referred to as “gamification,” this approach employs elements that are typically seen in games in non-game contexts [18]. Educational tools have followed this trend, with many including features such as points [10], leaderboards [4], levels [43] and virtual achievements [15].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

DOI: <https://doi.org/10.1145/3159458.3159501>

This raises the question, “Can gamification positively impact student behavior and learning outcomes?”

In a comprehensive review of the literature, Hamari et al. report that three particular elements: points, leaderboards and badges are the most commonly used in empirical studies of gamification [28]. Their review concluded that most published gamification studies reported some positive effects, but they identified a number of methodological limitations that may have contributed to varying results. These limitations included small sample sizes, lack of control groups and very short experiment timeframes. A fourth limitation was that multiple game elements were often investigated in combination, but not individually, making it impossible to establish whether individual elements had measurable effects.

In this work we investigate two of the most common gamification elements, points and badges, as used in an online learning tool. Our context is a large first-year anatomy and physiology course (701 participants), where we investigate student engagement with the tool over an entire 15 week semester and relate engagement to subsequent exam performance. We examine the effects of the game elements both individually and in combination, relative to a control group.

We explore two related research questions. Our primary question tests the hypothesis that gamifying an online study tool will have a causal effect on subsequent exam performance. Landers' theory of gamified learning provides strong theoretical support for this hypothesis [34]. Our secondary research question tests the hypothesis that a combination of game elements will have a greater effect on student behavior than either element used on its own. We measure the individual effects of our implemented points and badge systems, and we determine if their simultaneous use is beneficial in our context.

### BACKGROUND

Education is an increasingly common application area for gamification [2, 33, 51]. This has been driven by the potential for gamification to address challenges around student motivation and to positively impact learning [8, 36]. This latter outcome is of particular importance in educational contexts. The relationship between gamification and learning outcomes may be mediated by behaviors, such as time-on-task, that

Denny, McDonald, Empson, Kelly & Petersen (2018)

# Example 2: Influencing (positive) behaviours

CHI 2018 Honourable Mention

CHI 2018, April 21–26, 2018, Montréal, QC, Canada

## Empirical Support for a Causal Relationship Between Gamification and Learning Outcomes

Paul Denny  
University of Auckland  
Auckland, New Zealand  
paul@cs.auckland.ac.nz

Fiona McDonald,  
Ruth Empson, Philip Kelly  
University of Otago  
Dunedin, New Zealand  
{fiona.mcdonald, ruth.empson,  
philip.kelly}@otago.ac.nz

Andrew Petersen  
University of Toronto  
Mississauga, Canada  
andrew.petersen@utoronto.ca

### ABSTRACT

Preparing for exams is an important yet stressful time for many students. Self-testing is known to be an effective preparation strategy, yet some students lack motivation to engage or persist in self-testing activities. Adding game elements to a platform supporting self-testing may increase engagement and, by extension, exam performance. We conduct a randomized controlled experiment ( $n=701$ ) comparing the effect of two game elements – a points system and a badge system – used individually and in combination.

We find that the badge system elicits significantly higher levels of voluntary self-testing activity and this effect is particularly pronounced amongst a relatively small cohort. Importantly, this increased activity translates to a significant improvement in exam scores. Our data supports a causal relationship between gamification and learning outcomes, mediated by self-testing behavior. This provides empirical support for Landers' theory of gamified learning when the gamified activity is conducted prior to measuring learning outcomes.

### ACM Classification Keywords

H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous; K.3.0 Computers and Education: General

### Author Keywords

gamification; points; badges; self-testing; PeerWise

### INTRODUCTION

A growing number of online platforms are incorporating game-like elements to motivate users and generate higher levels of activity. Commonly referred to as “gamification,” this approach employs elements that are typically seen in games in non-game contexts [18]. Educational tools have followed this trend, with many including features such as points [10], leaderboards [4], levels [43] and virtual achievements [15].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM 978-1-4503-2652-9/18/04/0000...\$15.00.

This raises the question, “Can gamification positively impact student behavior and learning outcomes?”

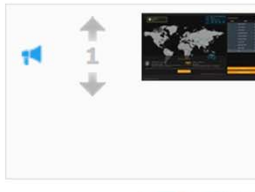
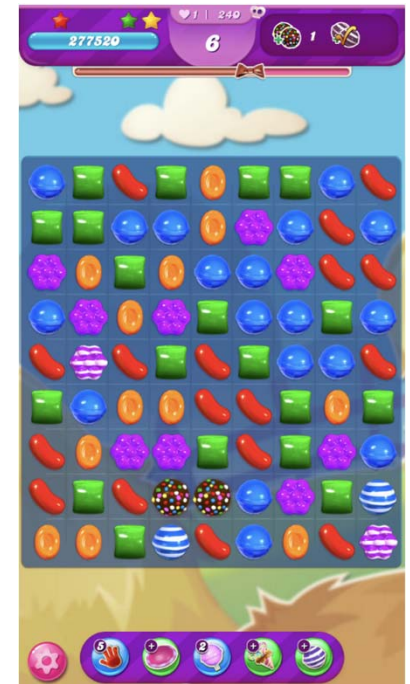
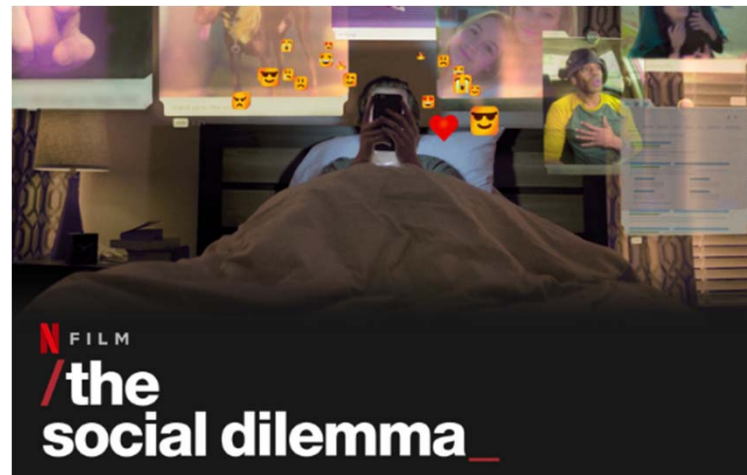
In a comprehensive review of the literature, Hamari et al. report that three particular elements: points, leaderboards and badges are the most commonly used in empirical studies of gamification [28]. Their review concluded that most published gamification studies reported some positive effects, but they identified a number of methodological limitations that may have contributed to varying results. These limitations included small sample sizes, lack of control groups and very short experiment timeframes. A fourth limitation was that multiple game elements were often investigated in combination, but not individually, making it impossible to establish whether individual elements had measurable effects.

In this work we investigate two of the most common gamification elements, points and badges, as used in an online learning tool. Our context is a large first-year anatomy and physiology course (701 participants), where we investigate student engagement with the tool over an entire 15 week semester and relate engagement to subsequent exam performance. We examine the effects of the game elements both individually and in combination, relative to a control group.

We explore two related research questions. Our primary question tests the hypothesis that gamifying an online study tool will have a causal effect on subsequent exam performance. Landers' theory of gamified learning provides strong theoretical support for this hypothesis [34]. Our secondary research question tests the hypothesis that a combination of game elements will have a greater effect on student behavior than either element used on its own. We measure the individual effects of our implemented points and badge systems, and we determine if their simultaneous use is beneficial in our context.

### BACKGROUND

Education is an increasingly common application area for gamification [2, 33, 51]. This has been driven by the potential for gamification to address challenges around student motivation and to positively impact learning [8, 36]. This latter outcome is of particular importance in educational contexts. The relationship between gamification and learning outcomes may be mediated by behaviors, such as time-on-task, that



You're probably a kick-ass developer... but are you a secure developer? Try our gamified challenges, climb the leaderboard, and win a FREE T-shirt! (securecodewarrior.com)

promoted by SecCodeWarrior

[promoted](#) [save](#) [give award](#) [report](#)

Denny, McDonald, Empson, Kelly & Petersen (2018)

## Example 2: Influencing (positive) behaviours

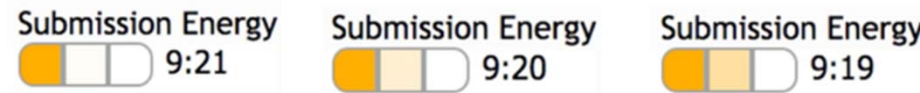
- Or, avoiding (negative) behaviours

Paper Session: CS1 Metacognitive

CompEd '19, May 17–19, 2019, Chengdu, Sichuan, China

### Can Mobile Gaming Psychology Be Used to Improve Time Management on Programming Assignments?

Michael S. Irwin and Stephen H. Edwards  
Department of Computer Science  
Virginia Tech  
Blacksburg, VA, USA  
mikesir@vt.edu, edwards@cs.vt.edu



**Figure 1: Submission energy bar, showing countdown and the animated fading of the next available unit being regenerated.**






Irwin & Edwards (2019)

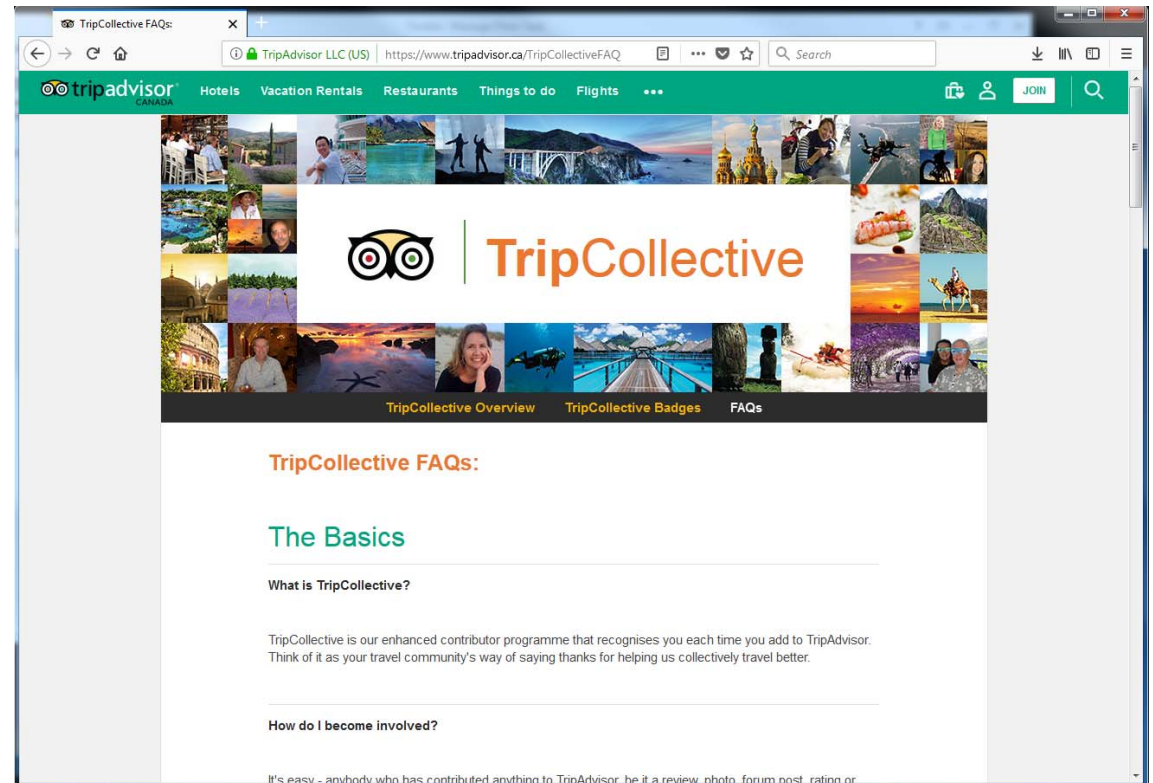
# Example 2: Influencing (positive) behaviours

## The Programme

### How do I receive points?

Every time you contribute to TripAdvisor, you receive TripCollective points. Here's a list of what you can contribute, and how much it's worth.

	Review	100 points
	Photo	30 points
	Video	30 points
	Forum Post	20 points
	Rating	5 points



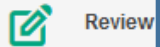


## Example 2: Influencing (positive) behaviours

### The Programme

#### How do I receive points?

Every time you  
TripCollective p  
contribute, and



Review



Photo



Video



Forum Post

20 points

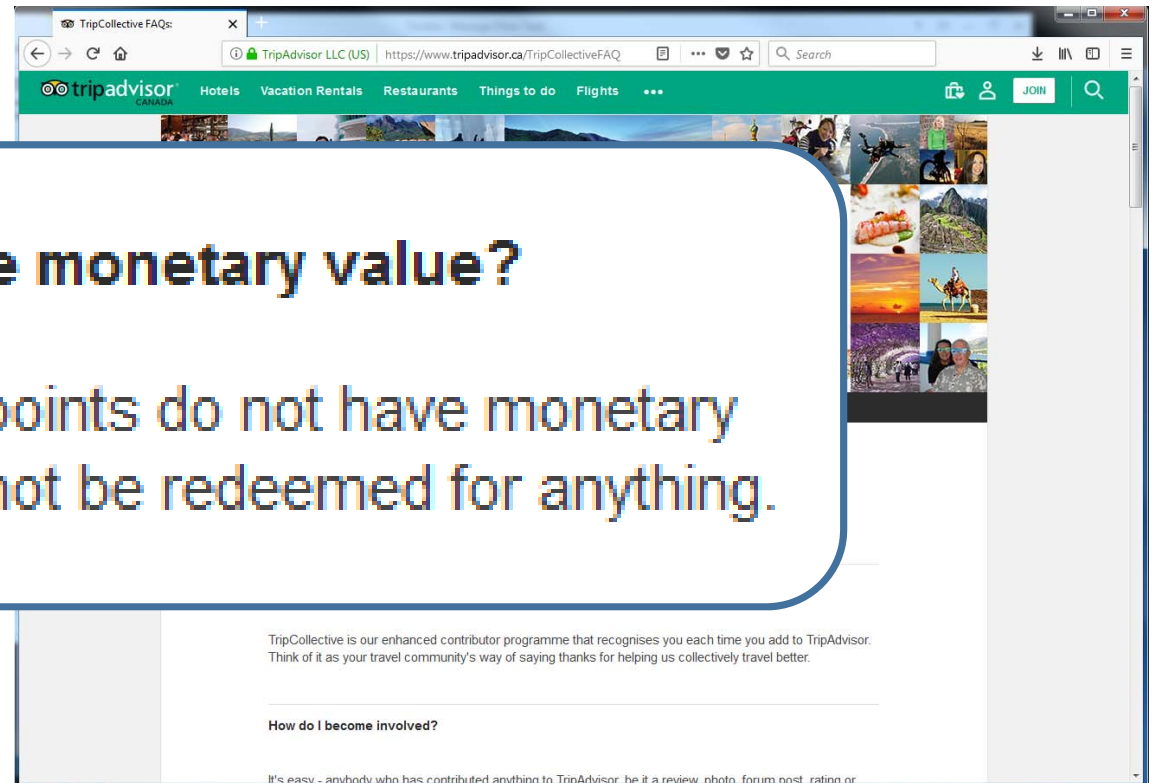


Rating

5 points

### Do points have monetary value?

TripCollective points do not have monetary value and cannot be redeemed for anything.



## Example 2: Influencing (positive) behaviours

“Empirical research on the effectiveness of incorporating game elements in learning environments is still scarce”

[Dicheva et al.; 2015]



# Example 2: Influencing (positive) behaviours

Topical Article

## Generation and Retrieval Practice Effects in the Classroom Using PeerWise

Matthew R. Kelley<sup>1</sup>, Elizabeth K. Chapman-Orr<sup>2</sup>,  
Susanna Calkins<sup>3</sup>, and Robert J. Lemke<sup>4</sup>

### Abstract

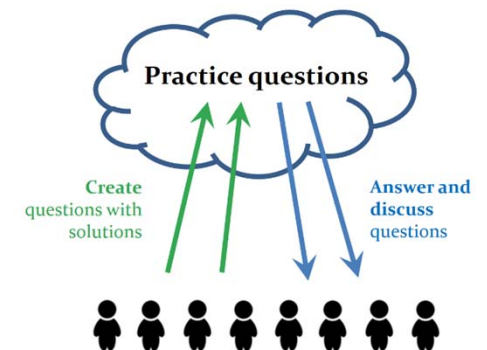
The present study explored the generation and retrieval practice effects within a college classroom using a free, online tool called PeerWise (PW). PW allows students to create their own multiple-choice questions, share them with peers, and answer the shared questions written by their peers. Forty students from two sections of an upper level cognitive psychology course authored and answered multiple-choice questions as part of a semester-long assignment. Analyses showed reliable generation and retrieval practice effects following PW usage, along with a significant improvement in exam performance.

### Keywords

generation effect, retrieval practice, PeerWise



Teaching of Psychology  
2019, Vol. 46(2) 121-126  
© The Author(s) 2019  
Article reuse guidelines:  
sagepub.com/journals-permissions  
DOI: 10.1177/0098628319834174  
journals.sagepub.com/home/top



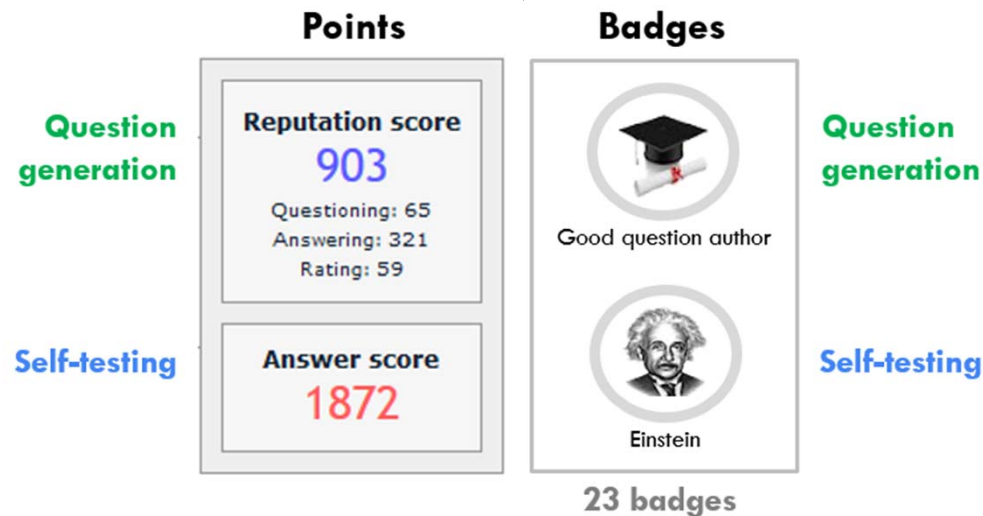
The "generation" effect  
[Slamecka & Graf, 1978]

The "testing" effect  
[Karpicke & Blunt, 2011]

Kelley, Chapman-Orr, Calkins, Lemke. **Generation and Retrieval Practice Effects in the Classroom Using PeerWise**,  
Teaching of Psychology, March 1, 2019.

## Example 2: Influencing (positive) behaviours

Desired behavior	Example reward
Rate questions early and fairly	Reputation score (rating component)
Answer questions correctly	Answer score
Spaced practice sessions	Commitment badge
Create good questions	Good question author badge
....	....




# Example 2: Influencing (positive) behaviours

[Home](#) | [Main menu](#)


**Your questions**  
[view »](#) You have created **5** questions

**Answered questions**  
[view »](#) You have answered **207** questions

**Unanswered questions**  
[view »](#) There are **146** questions for you to answer

  
[Provide feedback](#)

[Home](#) | [Main menu](#)

 Congratulations - you've earned a new badge! [View](#)




**Your questions**  
[view »](#) You have created **5** questions

**Answered questions**  
[view »](#) You have answered **207** questions

**Unanswered questions**  
[view »](#) There are **146** questions for you to answer

**Reputation score**  
**903**  
Questioning: 65  
Answering: 321  
Rating: 59

**Answer score**  
**1872**

 [View points](#)  [View badges](#)  [Provide feedback](#)

control

vs.

game

Denny, McDonald, Empson, Kelly, Petersen. 2018. **Empirical Support for a Causal Relationship Between Gamification and Learning Outcomes**. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18). ACM, New York, NY, USA

## Example 2: Influencing (positive) behaviours



*“Personally I tried really hard to get the ‘Leader’ badge, where I had to gain at least one follower! This was really motivating, and made me think more carefully and creatively when writing a question.”*

---

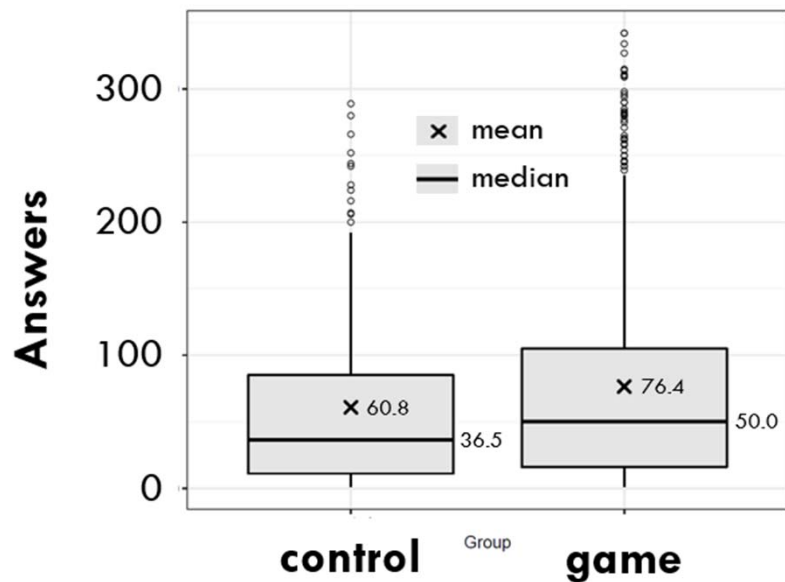


*“I didn't think I was ‘badge’ type of person, but I did enjoy getting badges (I was the first one to get the ‘obsessed badge’ - yay!). It did help motivate me to do extra and in doing so, I believe I have learnt more effectively.”*

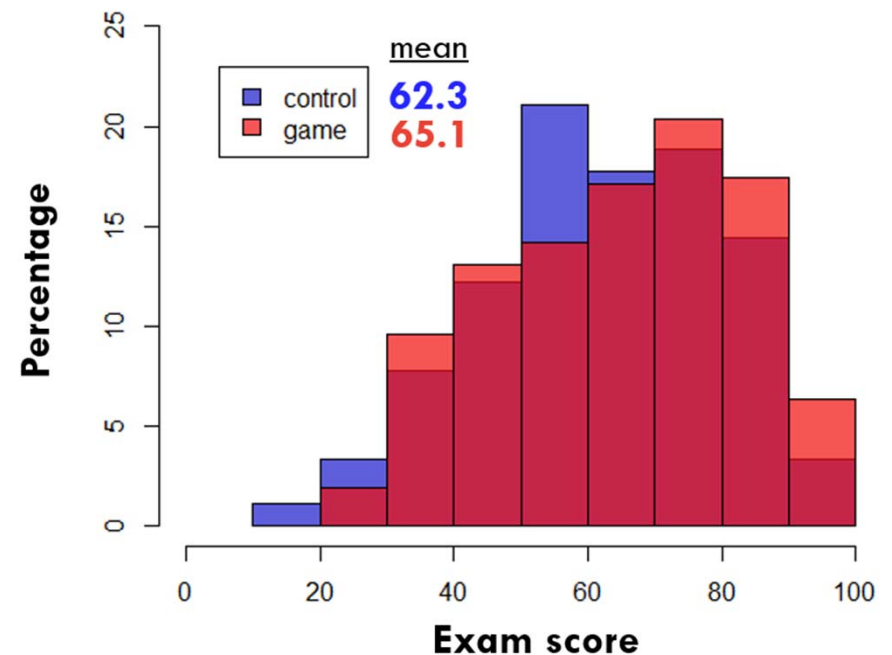
## Example 2: Influencing (positive) behaviours

### Game elements caused:

- twice as many students to create questions (6.7% vs. 11.5%)
- nearly 40% increase in answering activity



**37% increase in median number of answers**  
( $p = .016$ )



**4.5% increase in mean exam score**  
( $p = .038$ )

## Summary

- The goal of Computing Education research is to help students learn computer science more effectively
- The goal of Learning Technology research is the same, but applies more broadly across disciplines
- These are interesting areas of research, to which a range of computer science skills can be applied, and with the potential of large impact
- Our School's graduate course is COMPSCI 747!