

## **UC Irvine**

### **UC Irvine Electronic Theses and Dissertations**

#### **Title**

An Experiment in Microtask Crowdsourcing Software Design

#### **Permalink**

<https://escholarship.org/uc/item/2904d747>

#### **Author**

Lopez, Consuelo

#### **Publication Date**

2015

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA,  
IRVINE

An Experiment in Microtask Crowdsourcing Software Design

THESIS

submitted in partial satisfaction of the requirements  
for the degree of

MASTER OF SCIENCE

in Software Engineering

by

Consuelo López

Thesis Committee:  
Professor André van der Hoek, Chair  
Associate Professor James Jones  
Assistant Professor Thomas LaToza

2016



# DEDICATION

To all those crowds which, collectively working, make things happen, especially when it is impossible through individual efforts. But there is one specific crowd, to which I want to especially dedicate this thesis: the one of all those girls and women who battle against gender inequality every single day.



# TABLE OF CONTENTS

	Page
<b>LIST OF FIGURES</b>	<b>v</b>
<b>LIST OF TABLES</b>	<b>vii</b>
<b>ACKNOWLEDGMENTS</b>	<b>viii</b>
<b>ABSTRACT OF THE THESIS</b>	<b>x</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Related Work</b>	<b>6</b>
2.1 Crowdsourcing . . . . .	6
2.2 Crowdsourcing Models . . . . .	8
2.3 Microtask Crowdsourcing . . . . .	9
2.4 Crowdsourcing in Software Engineering . . . . .	10
2.5 Crowdsourcing and Design . . . . .	12
<b>3 Experiment Design</b>	<b>15</b>
3.1 Experimental Conditions . . . . .	16
3.2 Participant Recruitment . . . . .	17
3.3 Experiment Time . . . . .	18
3.4 Qualification Tests . . . . .	19
3.5 Tasks . . . . .	19
3.6 CrowdDesign Platform . . . . .	20
3.7 Questionnaire . . . . .	23
3.8 Review Procedure . . . . .	25
3.9 Compensation . . . . .	26
<b>4 Data Analysis</b>	<b>27</b>
4.1 Diversity . . . . .	27
4.2 Quality . . . . .	28
4.3 Difficulty . . . . .	30
4.4 Collaboration . . . . .	30

<b>5</b>	<b>Results</b>	<b>33</b>
5.1	General Results . . . . .	33
5.2	User Interface Design Without Examples (UIplain) . . . . .	39
5.2.1	Diversity . . . . .	39
5.2.2	Quality . . . . .	41
5.2.3	Difficulty . . . . .	44
5.3	Internal Code Design Without Examples (ICplain) . . . . .	45
5.3.1	Diversity . . . . .	45
5.3.2	Quality . . . . .	47
5.3.3	Difficulty . . . . .	49
5.4	Influence of Examples (UIexamples and ICexamples) . . . . .	50
5.4.1	Diversity . . . . .	50
5.4.2	Quality . . . . .	54
5.4.3	Difficulty . . . . .	57
5.5	Borrowing . . . . .	58
5.5.1	Use of ‘Duplicate’ and ‘Copy’ Features . . . . .	58
5.5.2	Solution Categories Distances . . . . .	61
5.6	Additional Analyses . . . . .	64
5.6.1	Finding Subgroups within the Solution Alternatives . . . . .	64
5.6.2	Finding Subgroups within the Crowd . . . . .	68
<b>6</b>	<b>Discussion</b>	<b>73</b>
<b>7</b>	<b>Threats to Validity</b>	<b>78</b>
7.1	Internal Threats to Validity . . . . .	78
7.2	External Threats to Validity . . . . .	79
<b>8</b>	<b>Conclusion and Future Work</b>	<b>81</b>
	<b>Bibliography</b>	<b>84</b>
	<b>Appendices</b>	<b>90</b>
A	Consent Form and Demographics Questionnaire . . . . .	90
B	Qualification tests . . . . .	93
B.1	User Interface Design Qualification Tests . . . . .	93
B.2	Internal Code Design qualification Tests . . . . .	97
C	Tasks description . . . . .	100
C.1	User Interface Design Tasks . . . . .	100
C.2	Internal Code Design Tasks . . . . .	104
D	Diversity per Worker . . . . .	108
D.1	User Interface Decision Points . . . . .	108
D.2	Internal Code Decision Points . . . . .	112
E	Cumulative Number of Unique Categories Created by Workers . . . . .	116
E.1	User Interface Decision Points . . . . .	116
E.2	Internal Code Decision Points . . . . .	119

# LIST OF FIGURES

	Page
1.1 Morphological chart for a vegetable collection system. . . . .	4
3.1 Stages of the experiment. . . . .	16
3.2 HIT description for UIplain experiment. . . . .	18
3.3 Task description for ‘representing cars’ decision point. . . . .	21
3.4 CrowdDesign platform. . . . .	22
3.5 Copy and duplicate features in CrowdDesing platform. . . . .	23
3.6 Exit questionnaire. . . . .	24
3.7 Quit questionnaire. . . . .	24
3.8 CrowdDesign admin tool. . . . .	25
4.1 Affinity diagrams constructed during data analysis. . . . .	28
4.2 Example of the use of ‘duplicate’ feature. . . . .	31
5.1 Example of a solution alternative which did not show honest effort. . . . .	34
5.2 Designs submitted by workers for UI design task. . . . .	37
5.3 Designs submitted by workers for IC design task. . . . .	38
5.4 Diversity per worker for ‘creating maps’ design (UIplain). . . . .	40
5.5 Cumulative number of unique categories provided by workers for ‘creating maps’ design (UIplain). . . . .	40
5.6 Example of a top solution provided for ‘setting timing of traffic lights’ decision point (UIplain). . . . .	43
5.7 Diversity per worker for ‘moving cars’ design (ICplain). . . . .	46
5.8 Cumulative number of unique categories provided by workers for ‘moving cars’ design (ICplain). . . . .	47
5.9 Diversity per worker for ‘creating maps’ decision point (UIexamples). . . . .	52
5.10 Diversity per worker for ‘moving cars’ decision point (ICexamples). . . . .	52
5.11 Cumulative number of unique categories created by workers for ‘creating maps’ design (UIexamples). . . . .	53
5.12 Cumulative number of unique categories created by workers for ‘moving cars’ design (ICexamples). . . . .	53
5.13 Cumulative average quality score for ‘creating maps’ decision point in the UIexamples experiment (left), and ‘moving cars’ decision point in the ICexamples experiment (right). . . . .	55

5.14	Occurrences of each category in a chronological order (‘creating maps’ decision point, UIexamples experiment). . . . .	62
5.15	Example of a solution within the “map only” category (‘creating maps’ decision point, UIplain). . . . .	66
5.16	Example of a solution within the “automated using input” category (‘creating maps’ decision point, UIplain). . . . .	67

# LIST OF TABLES

	Page	
5.1	Number of workers at each step of the experiment. . . . .	34
5.2	Demographics. . . . .	35
5.3	Number of solution alternatives submitted per decision point (UI design). . .	36
5.4	Number of solution alternatives submitted per decision point (IC design). . .	36
5.5	Number of resulting categories per decision point for UIplain experiment. . .	39
5.6	Quality of solution alternatives (UIplain). . . . .	41
5.7	Number of requirements met (UIplain). . . . .	42
5.8	Crosstabulation of quality scores and requirements met. . . . .	42
5.9	Number of resulting categories per decision point for ICplain experiment. . .	45
5.10	Quality of solution alternatives (ICplain). . . . .	47
5.11	Number of requirements met (ICplain). . . . .	48
5.12	Crosstabulation of quality scores and requirements met (ICplain). . . . .	49
5.13	Number of resulting categories per decision point (UIexamples). . . . .	51
5.14	Number of resulting categories per decision point (ICexamples). . . . .	51
5.15	Quality of solution alternatives for UIexamples (top) and ICexamples (bottom). .	54
5.16	Number of requirements met for UIexamples (top) and ICexamples (bottom). .	56
5.17	Crosstabulation of quality scores and requirements met (UIexamples). . . . .	57
5.18	Crosstabulation of quality scores and requirements met (ICexamples). . . . .	57
5.19	Exit questionnaire results across the four experiments and comparisons. . . .	58
5.20	Use of copy and duplicate feature (ICexamples). . . . .	59
5.21	Use of copy and duplicate feature (UIexamples). . . . .	60
5.22	Number of sketches with $x$ distance for each of the four experiments. . . . .	63
5.23	Resulting average quality, average number of requirements met, and total diversity of the subset of ‘top designs’ (all experiments). . . . .	64
5.24	Average quality and average number of requirements met for categories iden- tified for ‘creating maps’ decision point. . . . .	65
5.25	Resulting average quality, average number of requirements met, and total diversity when filtering out solutions with $x$ requirements met (all experiments). .	66
5.26	Average quality, average number of requirements met, and total diversity per demographic group. . . . .	68
5.27	Demographics of the group of workers who generated the best designs. . . .	70
5.28	Performance of workers with $x$ correct answers out of five during the qualifi- cation test per experiment. . . . .	71

# ACKNOWLEDGMENTS

Fortunately, there are many people I need to thank for helping me succeed on this adventure, personally, academically, and professionally. First and foremost, thank you to my family (V́ctor, Adriana, Emilia and Toḿas), who put up with the distance, and gave me all their love and encouragement, which has been essential to succeed in my endeavor.

Mati, I could not imagine a way to surf this wave without you. Undoubtedly, you flavored this experience; you gave it colors, you gave it red, you gave it magic and enthusiasm. Thanks for helping me when I bumbled, when I did not believe in myself, and when I was breathless. Thanks for celebrating my triumphs, for encouraging me in every new project and sharing with me the little (and big) things that make me happy. Thank you for making me feel at home. You are a wonderful person, a magnificent professional, and, most of all, the best adventure partner.

My deepest gratitude to Andr  van der Hoek, my advisor, for these two years of guidance. Thank you for giving me this opportunity; your support and confidence were essential for me to come to UCI. Thank you for sharing your wisdom and your hard work, for inspiring me to seek out new challenges, and for helping me when I stumbled. Thank you for always pushing me to do my best and for sparking my passion for research.

I am very thankful to my committee members. Thank you Jim Jones, you were the first Professor I had at UCI, and the one who taught me how to read papers, an essential skill without which I would not have been able to write this thesis. Thank you Thomas LaToza for feeding my curiosity about crowdsourcing.

Many thanks to BEC.AR and the Government of Argentina for trusting me, supporting my stay in the United States, and for believing that education, science and technology are decisive to make a better society. A special thanks to the Fulbright Commission in Argentina who made my life immensely easier, and assisted me every single time that I needed. This would not have been possible without them.

This is not an individual work but the effort of a lot of people. A very special thanks to my research group. Sahand Nayebaziz, thanks for your curiosity and your patience. Fernando Spanghero, thanks for sharing this ride with me. I want to especially thank Edgar Weidema, with whom I closely worked together during this project. Edgar, thank you for your encouragement, trust, patience, and especially for your (now, at a distance) constant support.

Many thanks to all the anonymous workers from Amazon Mechanical Turk who participated in the experiments, for their great feedback and dedicated work. Without them, this work would not have been possible.

To everyone in the SDCL research group, thank you for making my stay a memorable and wonderful experience. Nicolas Mangano, thank you for all the hours you spent helping me during my initial steps in research. Thank you all the people who contributed to this work

with their time and effort. Thanks to Ankita Raturi, Christian Adriano, Lee Martie, Matias Giorgio, Mengyao Zhao, Sara Triplet, and Thomas Kwak for spending many hours scoring designs, clustering solution alternatives on the wall, and mainly, for supporting my work with invaluable feedback. My thanks to Arturo di Lecce, Fabio Ricci, and Martin Medina who made my stay a joyful experience.

I would like to express my appreciation to my first academic mentors: Judith Meles and Daniel Battistelli. Judith, I will always be very thankful for encouraging me to continue studying and sharing my knowledge. Daniel, you made me realize that no class is large enough to scare me, and that we need to push ourselves to become better on what we do.

I have been lucky to meet fantastic people in the last two years. Coming alone and living in a different country has been, indeed, a challenge and an adventure. Thanks to all the friends I made here, you have made my stay much, much happier. I want to especially thank Asal Askarizadeh, initially my roommate, now my friend, for your help, generosity, and care.

Thank you very much to those unique women who are sisters to me. Agus, Leti, Lety, Nanu, Sara, Sofi, Tefi, Vero, and Vale, thank you for sharing my achievements and drying my tears. Especially, thank you for making me feel that distance does not exist.

Thank you, University of California, Irvine for this wonderful experience and for enriching me as a person and as a professional. I hope this is not a 'goodbye', but just a 'see you later'.

# ABSTRACT OF THE THESIS

An Experiment in Microtask Crowdsourcing Software Design

By

Consuelo López

Master of Science in Software Engineering

University of California, Irvine, 2016

Professor André van der Hoek, Chair

Microtask crowdsourcing is a form of crowdsourcing in which work is decomposed into a set of small, self-contained tasks, which each can typically be completed in a matter of minutes. The approach has been used to address a number of different problems, ranging from labeling images to planning travel. To date, however, little is known about the potential of microtask crowdsourcing in software engineering.

This thesis explores microtask crowdsourcing as applied to software design work. We particularly conducted a large study with Amazon Mechanical Turk workers, who each provided one or more solution alternatives for a small, partial software design problem. We included two experimental conditions: (1) user interface design work versus internal code design work, and (2) workers operating independently versus workers being shown previous designs from other workers.

We report on various results concerning solution diversity, solution quality, and perceived task difficulty, across the different experimental conditions. Our primary findings show that: (1) it is feasible for a crowd to generate a broad range of solution alternatives for a software design problem, (2) solutions alternatives range all over the quality spectrum, and (3) many workers perceived the task as difficult.



# 1

## Introduction

The use of crowdsourcing as an approach to performing diverse kinds of tasks has rapidly increased in popularity over the last few years [22]. It was Jeff Howe, in 2006, who first used the term crowdsourcing and defined it as “the act of a company or institution taking a function once performed by employees and outsourcing it to an undefined (and generally large) network of people in the form of an open call” [34]. Since then, crowdsourcing has become an accepted approach for tasks such as translation of video subtitles [3, 26], general graphical design [1], content moderation [19], and audio transcription [14]. Academia too, has begun to study crowdsourcing, for instance exploring how to use it for taxonomy creation [16], itinerary planning [90], language translation [89], or writing [7, 42].

Crowdsourcing is also gaining popularity as a means of supporting software engineering work (see [58]). To date, various commercial platforms are available to support different software engineering tasks. Examples of these platforms include, among others, Bugcrowd [12] for bug bounties, Stack Overflow [73] for expert exchange knowledge, uTest [83] for usability and system testing, and Topcoder [81] for programming competitions. Beyond practice, academia has devoted time and effort to the exploration of the application of crowdsourcing

to tasks such as requirements gathering (e.g., [11]) or software verification (e.g., [54]).

Different crowdsourcing models exist. Returning to the definition of crowdsourcing by Howe, several characteristics stand out and help differentiate these models. Particularly, the way these characteristics play out in practice may vary, for instance, in the form of the open call, the way a task is broken down, or how workers collaborate. Several canonical models have emerged, with LaToza and van der Hoek [53] describing some of the crowdsourcing models that are more prevalent in software engineering, including peer production, competition, and microtasking.

This thesis contributes to the exploration of microtask crowdsourcing for software engineering, particularly in the context of software design work. To date, the form of crowdsourcing that is explored the most in software design is the competition model. This model has shown promise, not only in software (e.g., Topcoder [81]), but also in other domains such as apparel design (e.g., [78]), architecture (e.g., [6]), video production (e.g., [80]), and music creation (e.g., [20]). Despite these successes, the effectiveness of the competition model has been questioned [75], mainly because of the linear nature of its underlying process: participants work independently on their tasks and the “aggregation mechanism” is simply to select a winner [48]. In doing so, the diversity of the crowd and the work it produces is not fully leveraged in that the effort of the losing contestants is wasted [15, 48]. Some platforms compensate by applying a two-phased approach, enabling borrowing from other designs (e.g., [48]), but even then a large amount of futile and unpaid work is performed by individual workers.

To explore a different approach, we focus in this thesis on microtask crowdsourcing. While not yet widely explored for software engineering, microtasking has had success when applied to complex problems [42], showing that it can match the quality of professionals [89]. For instance, *CrowdForge* [42], a general purpose framework for accomplishing complex and interdependent tasks using microtask markets, was used for article writing, obtaining higher

quality rates than individually produced articles. Furthermore, microtasks allow the use of a large and diverse crowd, potentially improving the exploration of the design space at a reduced cost [33]. Finally, because many more people participate, the loss of individual work if not selected is less of an issue for the individual, though collectively, of course, there still is some serious lost of effort. Interestingly, too, is that microtask crowd work, as long as it represents serious effort, is typically paid regardless of whether or not it is used.

The approach we explore in this thesis builds upon the concept of a morphological chart [72], a widely used design technique in engineering. A morphological chart, also known as a concept combination table [24] or a function-means table [23], consists of a set of main decision points (one per row) and solution alternatives for each decision point (multiple per row). Solution alternatives are meant to be derived in a structural manner, independent from the other decision points. For each decision point, its solution alternatives should represent as much of the space of possible solutions for that decision point only. A full morphological chart, then, consists of many partial solutions per decision point, from which a designer builds a complete (or near complete) design by choosing one alternative per decision point in such a way that the full set is both as compatible and as functional as possible. Figure 1.1 shows an example of a morphological chart for the design of a vegetable collection system [30].

Examining whether a morphological chart can be adapted to software design work by parallelizing its construction through microtasks leads to three questions:

1. *Can a crowd identify key decision points?* Given a prompt (set of requirements), is it possible for a crowd to identify the main decision points that, when taken together, represent the ‘heart of the design problem’ to be solved? Moreover, can the crowd specify those decision points sufficiently clearly so that they can serve as input into the next phase?


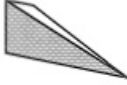


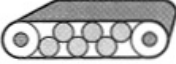


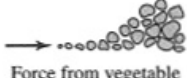
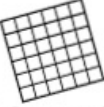



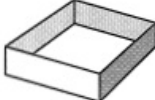

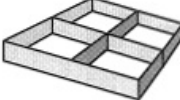
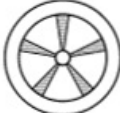


	Option 1	Option 2	Option 3	Option 4
Vegetable picking device		 Triangular plow	 Tubular grabber	 Mechanical picker
Vegetable placing device	 Conveyor belt	 Rake	 Rotating mover	 Force from vegetable accumulation
Dirt sifting device	 Square mesh	 Water from well	 Slits in plow or carrier	
Packaging device				
Method of transportation		 Track system	 Sled	
Power source	Hand pushed	Horse drawn	Wind blown	Pedal driven

Figure 1.1: Morphological chart for a vegetable collection system.

2. *Can a crowd identify solution alternatives?* Given a set of decision points, is it possible for a crowd to generate a diverse set of solution alternatives for each decision point? Moreover, are at least some of those solution alternatives of a sufficiently high quality that they actually solve the various partial design problems?
3. *Can a crowd assemble a design from the individual solution alternatives?* Given a set of decision points and solution alternatives for each of those decision points, is it possible for a crowd to select a set of alternatives that together form a complete or near-complete design?

In this work, we focus on the second question: given a set of specified decision points, can a crowd of workers generate a diverse set of solution alternatives with at least some set of those alternatives being of high quality? We chose this question first, because it is here that we believe the greatest benefit from the crowd may lie, as it is the question that is least

easily addressed by a single person, yet most easily parallelized.

To begin to formulate an answer to this question, we conducted four separate experiments on Amazon Mechanical Turk [4] that together explore two experimental conditions: (1) user interface (UI) design work versus internal code (IC) design work, and (2) workers operating independently versus workers being shown previous designs from other workers. In each experiment, we asked workers to provide solutions for four small, partial design problems, each representing a decision point in a large design task. To perform their work, crowdworkers used a special-purpose tool with which they were able to sketch a set of possible solution alternatives and provide explanatory descriptions for each of these solution alternatives.

We evaluated the resulting diversity, quality, and perceived difficulty for each of the four experiments. Our primary findings reveal that: (1) it is important for diversity to involve multiple workers, since individual workers did not create diverse sets of solution alternatives, (2) quality of the solutions varied considerably, (3) the task is seen as difficult by many workers. In addition, we observed a variety of additional phenomena upon which we report.

The remainder of the thesis is organized as follows. Chapter 2 presents background material on crowdsourcing, crowdsourcing models, crowdsourcing for software engineering, and crowdsourcing for design broadly speaking. Chapter 3 describes the various elements involved in our experiment design. Chapter 4 outlines the procedure we followed to analyze the results. Chapter 5 presents the results of our study. Chapter 6 reviews the key findings of our research. Chapter 7 discusses issues that arose throughout the work. Chapter 8 concludes by revisiting our contributions and outlining a number of directions for future work.

# 2

## Related Work

### 2.1 Crowdsourcing

Whereas the term ‘crowdsourcing’ was popularized to describe Internet-based activities [10], the history of crowdsourcing dates back to more than 300 years ago. The list of the most famous examples of the use of crowdsourcing includes the Longitude Prize in 1714 (when the British government offered the public a monetary prize to whoever came up with the most simple and practical method for the precise determination of a ship’s longitude at sea), the first publication of the Oxford English Dictionary in 1884 (when 800 volunteers catalogued words to create the first fascicle), and the design competition for the Sydney Opera House in 1957 (see [21] for a complete timeline of the crowdsourcing history). Not in the least because of the rise of digital connectivity in the world, the pace of the use of crowdsourcing has accelerated over the past few years. Dawson and Bynghall [21] attribute this acceleration to the influence of factors such as the rise of collaboration tools, the global awareness of crowdsourcing, the comfort with remote work, and results being produced at a competitive cost with great efficiency. To date, crowdsourcing is a practice driven by

Web 2.0 technologies [70], characterized by a customer, or *requester*, who advertises tasks on a crowdsourcing platform that are processed by members of the crowd, or *crowdworkers*, mostly for a fixed remuneration [39].

Implicit in the idea of crowdsourcing is the ability to create value that transcends individual contributions, articulating collective insights through structured aggregation [21]. According to Surowiecki [76], there are four criteria that empower the wisdom of a crowd: *diversity of opinion* (each person should have private information even if it is just an eccentric interpretation of the known facts), *independence* (people’s opinions are not determined by the opinions of those around them), *decentralization* (people are able to specialize and draw upon local knowledge), and *aggregation* (some mechanism exists for turning private judgments into a collective decision). Surowiecki states that, if a group satisfies those conditions, its judgment is likely to be accurate.

Today, crowdsourcing utilizes advanced Internet technologies to take advantage of the collective knowledge of the community or to exploit the crowd to directly produce goods and services [69]. Many online platforms are available to perform tasks in a wide variety of domains. For instance, *Wordy* [85] offers crowdsourced real-time copy-editing and proofreading services. Another example is *CastingWords* [14], which provides audio transcription services through the crowd. Finally, *HYVE Crowd* [37] is an example of a platform which runs creative competitions seeking designs for objects, ranging from a car trunk, to a hair product, to a new individual coffee machine.

The constant increase in fields that adopt crowdsourcing [10, 21, 22, 40, 70] and the success of existing crowdsourcing platforms provide initial evidence that crowdsourcing might offer benefits over traditional ways of working. The research community has reaffirmed and explained some of these benefits. Beyond many studies describing successful cases of crowdsourcing (e.g., [5, 22, 45]), other research seeks to document common lessons learned across such case studies. Kittur et al. [42], for instance, state that crowdwork has the potential to

support a flexible workforce and mitigate challenges such as shortages of experts in specific areas. As another example, Stol and Fitzgerald [75] highlight cost reduction, faster time to market, higher quality through broad participation and creativity, and open innovation as benefits of the use of crowdsourcing. Other benefits are described elsewhere (e.g., [21, 33]).

## 2.2 Crowdsourcing Models

Crowdsourcing is not a single strategy, but concerns an umbrella of approaches [35]. According to Doan et al. [22], any crowdsourcing system should address four main challenges: (1) how to recruit certain users, (2) what contributions users can make, (3) how to combine user contributions to solve the target problem, and (4) how to evaluate users and their contributions. There are naturally multiple ways to approach these challenges. Different authors describe, then, various crowdsourcing models that have emerged as relatively common.

Howe [35] identified, as early as 2008, four models of crowdsourcing. He suggests that, depending on what is being attempted to be achieved, a requester can select one of the following: harnessing the collective intelligence or crowd wisdom, using the crowd to sift through things and vote, using the crowd to create what you want to sell, and tapping into the crowd's collective financial resource. He identified these models as different because of the rather different underlying purpose of each.

Saxton et al. [69] describe a different categorization, one based on the underlying business model. Their taxonomy includes the intermediary model, the citizen media production model, the collaborative software development model, the digital goods sales model, the product design model, the peer-to-peer social financing model, the consumer report model, the knowledge based building model, and the collaborative science project model.

Dawson and Byngall [21] offer yet another categorization of crowdsourcing models. They



identified no fewer than 22 categories, which are clustered into seven types of crowd business models plus non-profit ventures. The list of categories includes, among others, competition markets, microtasks, innovation prizes, and innovation markets.

For the purposes of this thesis, we work with the categorization proposed by LaToza and van der Hoek [53]. They focus on crowdsourcing as models determined by eight characteristics of how different crowds work together, focusing on the underlying collaborative nature of crowdsourcing. They describe three main models: peer production (open source development is a well known example of this model), competitions (TopCoder [81] and 99designs [1] are examples of platforms embedding this model), and microtasking (typified by Amazon Mechanical Turk [4]). As this thesis explores the application of microtask crowdsourcing to software engineering, and software design in particular, the next section details this model further.

## 2.3 Microtask Crowdsourcing

Microtask crowdsourcing envisions a radically different model of work compared to other crowdsourcing models [22, 49]. Unique is that workers are recruited using an open call and are assumed to be transient, working on short, self-contained tasks [49, 53]. Frequently, indeed, a microtask can be completed in a matter of minutes [49, 53]. A solution to a more complex task is achieved by combining the set of self-contained microtasks through an aggregation mechanism (e.g., majority voting) [53].

Microtasking offers many potential advantages. Many authors agree on this model's primary benefit: its extreme scalability [33, 49, 53, 62]. By dividing tasks into self-contained microtasks, microtask crowdsourcing dramatically increases the potential for parallelism, enabling the work to be distributed to large crowds and leading to potentially very fast completion of

large tasks [52, 53]. Additionally, microtasking has been observed to offer benefits such as a more fluid labor force, fast worker recruitment, and, due to mass participation, the creation of diverse ideas from the crowd [49, 53].

There are some kinds of work which are naturally more suitable than others to be approached through microtasking: those that can be easily partitioned, distributed, and worked on. For instance, the most common use of microtasking involves tasks such as labelling images [84] or transcribing video [47]. These tasks can easily be broken down into smaller parts, require minimal context, and can be completed in a matter of seconds. Microtasking as an approach, however, starts facing more challenges when the complexity of the work increases, as such work often leads to interdependencies among microtasks that must be carefully orchestrated. Work has begun to explore the use of microtask crowdsourcing for tasks that require such coordination, and the resulting need for knowledge sharing [49], giving rise to the creation of workflows to coordinate crowdworkers' work [16, 33, 42, 44, 47, 62].

## 2.4 Crowdsourcing in Software Engineering

The field of software engineering is also investing effort in exploring the role that crowdsourcing can play. Mao et al. [58] tailored the original crowdsourcing definition by Howe to make it more specific to software engineering: “Crowdsourced Software Engineering is the act of undertaking any external software engineering tasks by an undefined, potentially large group of online workers in an open call format”. There are a number of crowdsourcing platforms specifically targeting software engineering work. For instance, TopCoder organizes competitions for tasks such as algorithm development and software design [81]. Bountify runs programming competitions, but for small self-contained coding tasks [9]. uTest [83], 99tests [2], Passbrains [64], and Testbirds [77] are examples of several crowdsourcing platforms for software testing. These testing platforms match client's needs with crowdworkers,

providing a wide range of testing services, such as functional testing, usability testing, and performance testing.

The research community has developed various crowdsourcing tools as well, in order to explore the possibilities of crowdsourcing in addressing more complex software engineering tasks (see [58] for an exhaustive literature survey up to May 2015). As a few examples, Lim et al. introduced *StakeSource*, a tool that uses crowdsourcing to automate stakeholder analysis [55] (available online at [74]); Tillmann et al. created *Code Hunt*, a gaming platform for coding contests to practice programming skills [17, 79]; Xue explored *CrowdBlaze*, a system that combines crowdsourced human testing efforts with automatic testing tools to improve testing coverage for Android apps [87]; and Li et al. developed *CrowdMine*, a system that, through gamification, recruits non-expert humans who can assist in the formal verification process of a piece of code.

Despite all of its potential benefits, the use of microtasks to approach software engineering work has been barely explored in the last few years, with only a few software engineering activities that have been approached through microtasks. For instance, for software development, *CrowdCode* [51] and *Collabode* [28, 29] are two web IDEs that each use a somewhat different mechanism to break down programming into microtasks, allowing a large crowd of developers to code various aspects of a program in parallel. For software debugging, *CrowdOracles* [65] recruits workers on Amazon Mechanical Turk to check and fix unit test assertions. For bug fixing, *HelpMeOut* [32] is a social recommender system that aids the debugging of error messages by suggesting solutions that peers have applied in the past. Notably absent in this set of examples is software design, the subject of this thesis.

## 2.5 Crowdsourcing and Design

This thesis studies crowdsourcing as it is applied to software design. Compared to other kinds of crowdsourcing work, design represents a task with significant more complexity. Yet, today's the popularity of crowdsourcing as an approach to design should not be surprising: diversity of thinking leads to creativity [88], a phenomenon long observed by social science research on creativity and brainstorming [13, 56, 60]. Design work, then, draws on crowdsourcing benefits such as broad participation and the potential to generate diverse ideas, stimulating the production of creative and innovative approaches to solve problems [27, 40, 88].

Outside of software, many commercial platforms show evidence of early success of design work through crowdsourcing. 99designs [1] excels in running design competitions for graphic and web design. Tongal [80] (audiovisual content), Slogan Slings [71] (slogan creation for advertisement and marketing), Arcbazar [6] (architectural design), and CrowdStudio [20] (music creation), are just a few of many examples of crowdsourcing platforms running design contests for different purposes. Design competitions create temporary arenas of exploration where innovative solutions can emerge at far lower cost than similar efforts in traditional settings [46], offering a wide range of quality alternatives in a matter of one or a few weeks [45].

The research community has explored different workflows to enable a crowd to produce high quality design work as well. *Flash Teams* [67] is a framework for dynamically assembling crowdsourced small expert teams, by linking modular tasks in order to build a sequence of tasks that, together, address more complex work, such as design prototyping and animation. *Crowd vs. Crowd (CvC)* [63] is a design crowdsourcing method in which several design teams made up of designers and assorted crowds compete with each other. Yu and Nickerson [88] proposed a sketch combination system, in which a large crowd participates in an iterative

process of design, evaluation, and combination to create a chair for children. In all of these cases, findings suggest that crowd based design processes may be effective in obtaining quality design outcomes with high levels of satisfaction for stakeholders.

In addition to generating designs, crowdsourcing has been used to provide feedback and critique existing designs for further improvement. Several studies have explored different workflows and platforms for this purpose. For instance, *CrowdCrit* [57] is a system that allows designers to receive design critiques from non-expert crowd workers. In several studies with the tool, it was found that: (1) the quality of crowd critiques approached that of expert critiques, (2) designers who received crowd feedback perceived that it improved their design process, and (3) designers were enthusiastic about crowd critiques and used them to change their design. As another example, *Voyant* [86] is a system which gives users access to a non-expert crowd to receive perception-oriented feedback on their designs. The authors highlighted the utility of the feedback generated by a crowd-based system for users and their designs.

In software design, crowdsourcing has primarily been used for user interface design. For instance, Huang et al. [36] propose a crowd-based method for creating mobile UI design pattern galleries, so designers can explore examples in the wireframing stage of the design process. As another example, Lasecki et al. [47] present *Apparition*, a crowdsourcing system which helps designers to create working interface prototypes in real-time by sketching and describing its functionality in natural language. As a final example, Nebeling et al. [61] introduce an approach for the development of web systems, involving crowds in composing data-driven web interfaces in a plug-and-play manner.

Crowdsourcing for internal software design (e.g. architecture, modules, code) is less explored, both in practice and in research. TopCoder [81] is one of the few platforms that supports internal design, through its competitions for general software design. Despite its popularity, some studies highlight limitations in the process it follows [48, 75]. Particularly, the com-

petition model presumes a waterfall process, requires clients to be intimately involved, and evaluates quality only late in the process. LaToza et al. [48] propose some improvements to the competition model, introducing a recombination phase during which participants had access to other participants' designs produced in the first phase. They could therefore borrow ideas from those other designs to improve their own work. Work quality improved overall, though the study also offers several input at recommendations to further improve software design competitions.

What is notable is that, in all of these examples of crowdsourcing software design, microtask crowdsourcing is notably absent as an approach. Our study aims to begin to fill this gap, and specifically explores the application of microtask crowdsourcing to software design, both for user interface design and for internal code design.

# 3

## Experiment Design

Chapter 1 introduced our overall vision of using a morphological chart [23, 24, 72] to support microtask crowdsourcing for software design. It also scoped the problem addressed to a subquestion of the overall vision: *Can a crowd identify solution alternatives?* To begin answering this question, we designed the experiment that is detailed in this chapter.

Returning to the morphological chart, remember that what we are trying to achieve is to get crowdworkers to generate solution alternatives for given decision points. This is, a design problem needs to first be decomposed into a set of small, partial design decisions that need to be made in order to tackle the overall problem; we term these design decisions *decision points*. Once the decision points have been determined, the challenge is to involve a crowd to generate solution alternatives for each of the decision points. We mapped decision points into tasks, with each worker being provided access only to the information for the particular decision point they are assigned. They do not have access to the whole design problem, nor to the complete morphological chart.

We conducted four separate experiments on Amazon Mechanical Turk. We posted one human intelligence task (HIT) per experiment. For any of the four experiments, once a

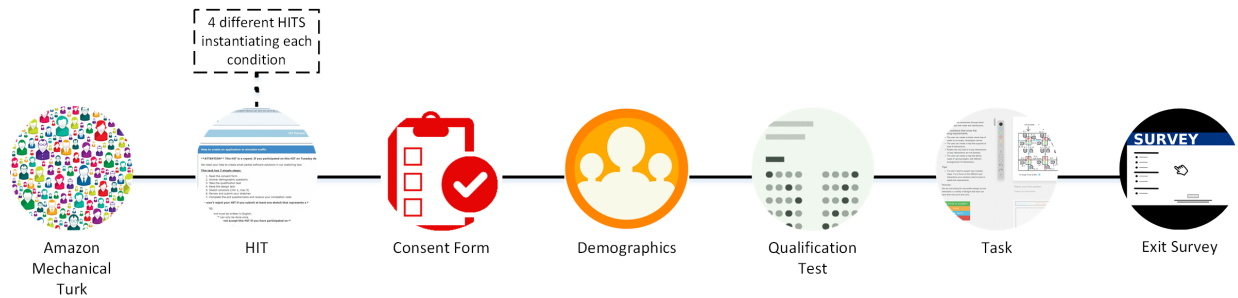


Figure 3.1: Stages of the experiment.

worker decided to participate in the HIT, they had to follow a link to our platform, on which they undertook the sequence of steps illustrated in Figure 3.1.

First, they had to read and ‘sign’ a consent form. Second, they had to provide basic demographic information. Third, they had to qualify for the experiment by passing a test consisting of five multiple choice questions. Finally, if they passed the test, the worker was given access to the actual task. After the worker finished providing solution alternatives, they were asked to complete a questionnaire about their experience, and were provided the option to give any additional feedback. After that, the worker was provided with a unique completion code, to be submitted on Amazon Mechanical Turk to facilitate payment.

In the following subsections, we detail each of these steps.

### 3.1 Experimental Conditions

Our first experimental condition is about the type of task. Naturally, when designing a piece of software, design takes place at multiple levels, including architecture, user interface, internal components, classes, algorithms, etc. We decided to make focus on two of those, namely *user interface* (UI) design and *internal code* (IC) design, one being more outwardly oriented and the other more internally oriented. Each, too, requires different expertise and the kind of work may be different enough to reveal differences in how the designs are created



by a crowd.

Our second experimental condition is about the impact of examples on the resulting quality and diversity. Specifically, we wanted to explore whether the availability of designs made by others influenced the workers in choosing how to approach their tasks. This is because previous research [48] reveals that designers often benefit from borrowing ideas from others. For this reason, we allowed workers in one condition to see designs from other workers and copy parts or entire sketches from other workers, and workers in the other condition to not see any examples at all. We of course did not permit the submission of identical solutions.

Each of the four resulting experiments arises from one of the four possible combinations between the two experimental conditions. The experiments were performed in sequence, one after the other, starting from UI design without showing other workers' work (*UIplain*), followed by UI design with revealing others' work (*UIexamples*), and repeating the same two conditions for internal code design (*ICplain* and *ICexamples*, respectively).

## 3.2 Participant Recruitment

All participants were recruited via Amazon Mechanical Turk. We considered other platforms (Topcoder [81] and Upwork [82]), but decided to use Amazon Mechanical Turk because of its ubiquity. The HIT was posted as an open call, without any stipulated restrictions. We did not promote the HIT, but rather let workers discover it on their own.

Workers were able to participate only once in our experiment. We set this restriction for two reasons. First, we wanted to recruit a diverse pool of workers, so to obtain solution alternatives from many workers instead of receiving many designs created by just few workers. Second, we wanted to avoid a learning effect in workers.

Sketch and explain small solutions for a software program (Bonus can earn you up to \$9.50)			
Requester: SDCL UCL	Reward: \$2.00 per HIT	HITs available: 0	Duration: 3 Hours
Qualifications Required: None			

**Help to create an application to simulate traffic**

**\*\*ATTENTION\*\* This HIT is a repost. If you participated on this HIT on Tuesday do NOT participate on this batch.**

We need your help to create small partial software solutions in our sketching tool.

**This task has 7 simple steps:**

1. Read the consent form
2. Answer demographic questions
3. Take the qualification test
4. Read the design task
5. Sketch solutions (min 1, max 5)
6. Review and submit your sketches
7. Complete the exit questionnaire and receive your completion code

**We won't reject your HIT if you submit at least one sketch that represents a thoughtful solution to the task.**

Attention:

- All text must be written in English.
- This HIT can only be done once.
- **Please do not accept this HIT if you have participated on this HIT on Tuesday, I cannot accept your work if you do.**

Technology Requirements:

- Chrome (v46) or latest version of Safari (OSX version)

Two bonus criteria:

- For each sketch that represents a honest attempt to solve the solution you earn an extra \$0.50 bonus.
- In addition, if your sketch covers at least 3 out of 4 of the given requirements you earn an extra \$1.00 bonus for a total of \$1.50, for each sketch.

**Make sure to leave this window open as you complete the task.** When you are finished, you will return to this page to paste the code into the box.

<b>Survey link:</b>	<a href="http://dellserver.lcs.ucl.edu:8080/crowddesign/ConsentForm.jsp?AR">http://dellserver.lcs.ucl.edu:8080/crowddesign/ConsentForm.jsp?AR</a>
<b>Provide the task code here:</b>	<input style="width: 80%;" type="text" value="e.g. 123456"/>

Figure 3.2: HIT description for UPlain experiment.

Figure 3.2 shows the HIT description as participants saw it in Amazon Mechanical Turk. The HIT described the steps, requirements, payment information, and provided a link to complete the task.

### 3.3 Experiment Time

Our goal for each of the four experiments was to collect solution alternatives from 80 workers (20 per each of four decision points). We decided upon this number of participants, because we were interested in analyzing both quality and diversity of the solution alternatives produced. To guarantee we would obtain a reasonable sample per decision point, we aimed for 20 workers per decision point. Because workers could submit from one to five solution

alternatives, we would be guaranteed a minimum of 20 solution alternatives and a maximum of 100. We set a maximum time of one week per experiment, and we decided to post the task twice within that timeframe. The reason why we opted to post the HIT twice instead of keeping a single HIT running for seven days is because newer HITs are more likely to be discovered by workers among the extensive list of tasks available in Amazon Mechanical Turk.

### **3.4 Qualification Tests**

Workers were eligible to participate in the task only if they passed a qualification test, so to ensure that each worker actually had some knowledge related to the task (i.e., some UI design knowledge or some coding knowledge). Workers were included based on their score on the qualification test. If they answered at least three out of five questions correctly, they could actually take the HIT and work on the task.

We prepared two different types of qualification tests, both of them following the same multiple choice format. For the user interface experiments, questions in the qualification test were about user interface design principles. For the internal code design, experiments the test consisted of five questions about a Java code snippet. We prepared multiple tests for each, to be randomly assigned to workers. Appendix B includes all qualification tests that we used.

### **3.5 Tasks**

To ensure breadth in our experimentation and analysis, and not accidentally bias the experiment, we used four decision points per experiment. Each decision point committed a worker

to design a different aspect of the same software. Each single HIT, thus, had underneath it four microtasks, one of which a worker would be randomly assigned to work on. To generate these tasks, we examined existing (complete) designs that were previously created by professional software designers for an educational traffic light simulator (see [66] for an extensive treatment of the design prompt). From these existing designs, we created a list of thirty key decision points, out of which we selected eight to use: four related to user interface design and four related to internal code design. Within the user interface decision points, we chose: map creation, setting of traffic light timings, visualization of the state of the simulation, and determining the flow of traffic. For the internal code design decision points, we selected the following: how to represent cars, the algorithm by which cars move, how to internally represent the road system, and the algorithm by which the traffic lights colors change.

All tasks followed the same structure. Each consisted of a brief description of its goal, four precise requirements, a couple of hints, and a reminder of the overall goal of the HIT being to generate solution alternatives. Figure 3.3 shows the task for the ‘representing cars’ decision point (see Appendix C for all eight decision points).

The task asked a worker to provide at least one and up to five different solution alternatives for a given decision point. We chose not to ask for just a single solution alternative, because we were curious whether individual workers would contribute more than one solution alternative and, when they did provide multiple solution alternatives, whether those alternatives were diverse (especially as compared to contributions made by other coworkers).

## 3.6 CrowdDesign Platform

To complete the HIT, workers had to use our proprietary CrowdDesign platform, which is shown in Figure 3.4. On the left of the screen (marked with (1)), workers were provided

## Task:

Sketch and describe the internal design through which cars are represented by the code, including their location and direction of driving on a given map. We are not looking for the user interface design, but the design of the code that captures the state of the simulation.

## Your design should cover the following:

- Cars drive on roads.
- Cars travel through intersections, possibly changing direction.
- A map has multiple entry and exit points for cars; cars enter at one, and exist at another.
- Only one type of car exists. No other vehicles types are present.

## Tips:

- Try to focus on the different parts of the internal design and the information each of the parts needs to capture to satisfy the requirements.
- To illustrate your solution, you may use any notation, from a simple drawing or sketch of the code, to pseudocode, to one or more UML diagrams, or any other representation you find convenient. This is up to you; you are free to explain your solution in any way that you believe would help a programmer to understand and implement your design.

## Reminder:

We are not looking for one perfect design but are interested in a variety of designs that each can have their own pro's and con's.

Figure 3.3: Task description for 'representing cars' decision point.

with all of the instructions necessary to complete the task. In the middle, the platform provides a set of basic sketching features (2), which allow the worker to produce a sketch illustrating their solution alternative on an empty canvas on the right (3). Note that the canvas has two associated text fields: one for the name of the solution alternative, and the other for a brief explanation of the solution alternative (4). Even though we were expecting short descriptions, the textual field did not have any limit on the amount workers could write, enabling workers to provide as much or as little detail as they considered necessary. Scrolling down reveals four additional canvases and associated textual fields, to be used for up to four additional solution alternatives (5). Once workers are content with their work or

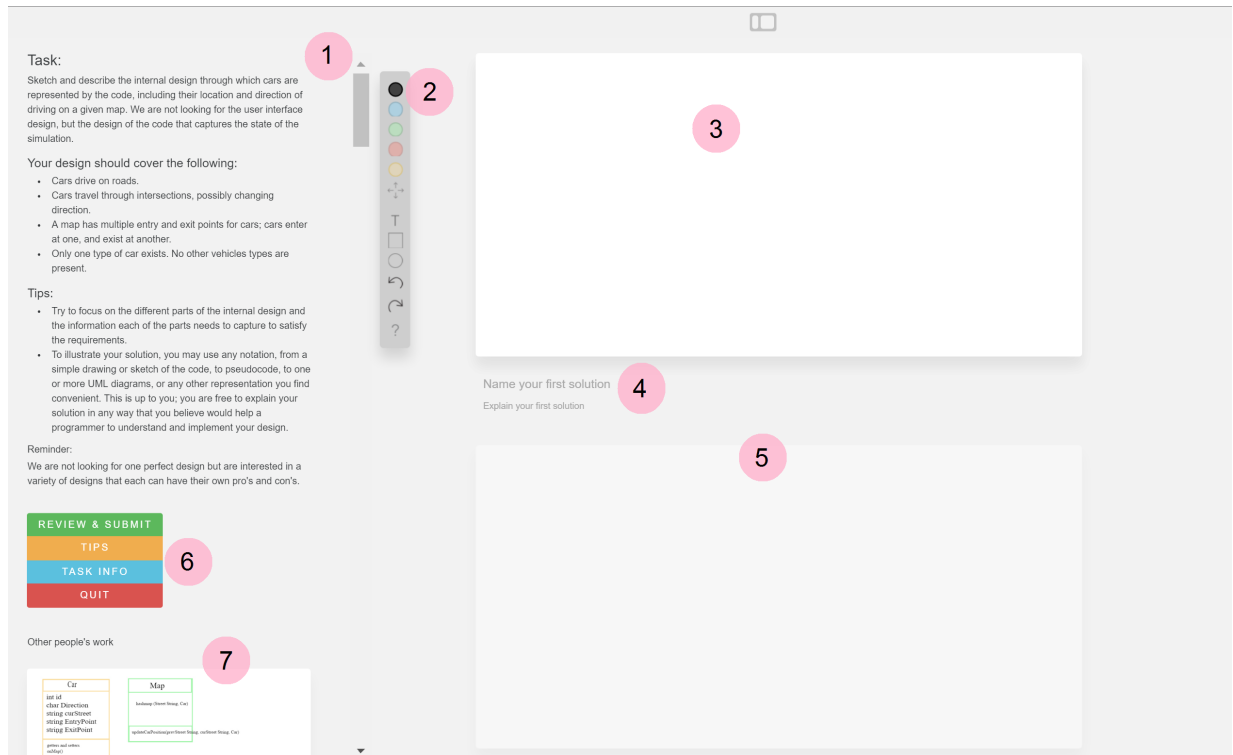


Figure 3.4: CrowdDesign platform.

otherwise feel they are done, they use the “Review & Submit” button (6) to submit their work and access the final survey.

Below the “Review & Submit” button is a “Quit” button. Workers had the option, at any time, to abandon the task. They could simply close the browser if they decided they no longer wanted to complete the task. However, by clicking the “Quit” button, they were redirected to a questionnaire. In this way, we were able to collect information from people who quit and analyze the reasons a worker had to abandoned the work.

Only for the two experiments where workers had access to their coworkers’ work (UIexamples and ICexamples), the prototype has an additional section ((7) in Figure 3.4). On scrolling down in this area, a worker can see other workers’ solution alternatives. This area not only displays the previous work, but workers could select and copy parts of any example sketch and include it in any of the five canvas as part of their own solution alternatives. They could

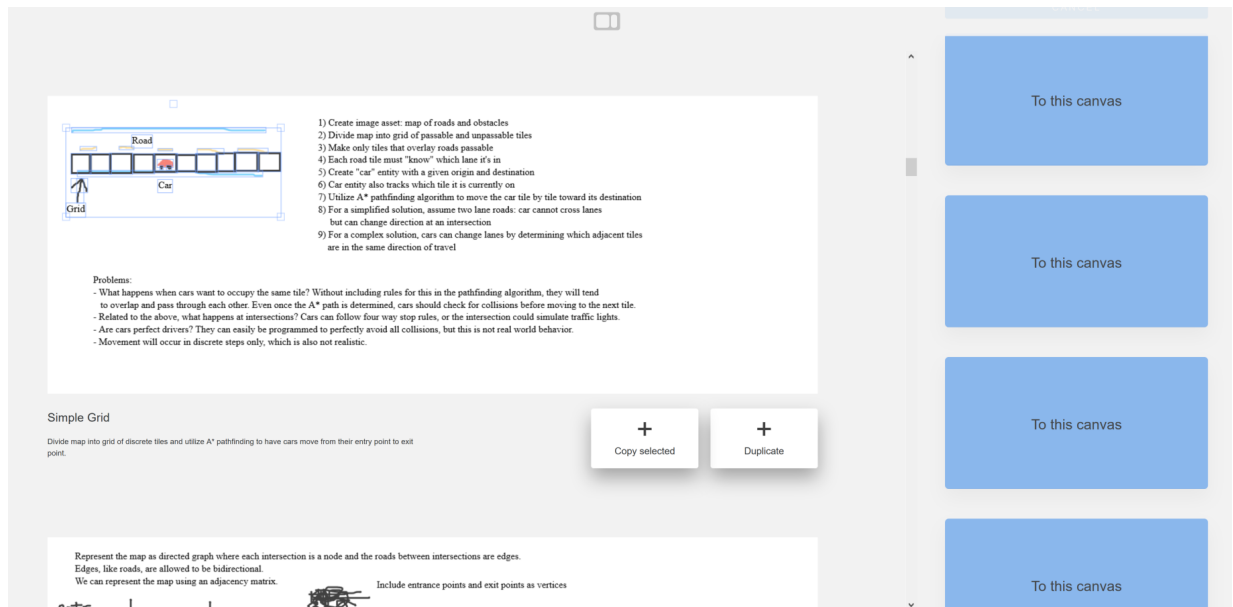


Figure 3.5: Copy and duplicate features in CrowdDesing platform.

even copy entire sketches. Figure 3.5 shows how a worker can use the copy and duplicate features. After selecting (part of) the source sketch, the user can select any of the five available canvases as the destination. If content already existed, the copied content would be added.

### 3.7 Questionnaire

After completing the task, workers were asked to complete a survey with four questions (see Figure 3.6). The first three questions asked them to rate on a one (easy) to seven (difficult) scale three aspects: ability to complete the entire task, difficulty level of the decision point, and adequacy of support by the tool. The fourth question was open ended, and asked workers for any general feedback they might have.

Those workers who decided to click on the “Quit” button to abandon the task were redirected to a different questionnaire (see Figure 3.7). In this case, we asked only two questions. The first question asked the worker why they quit the task. We pre-defined four options capturing

**Exit Questionnaire**  
please answer the following questions

How difficult was it to complete this task?  
very easy -  1  2  3  4  5  6  7 - very hard

How difficult was it to design for this decision point?  
very easy -  1  2  3  4  5  6  7 - very hard

How difficult was it to use this design tool?  
very easy -  1  2  3  4  5  6  7 - very hard

Do you have any feedback to improve this HIT?

[GET YOUR COMPLETION CODE](#)

Figure 3.6: Exit questionnaire.

**Are you sure you want to quit this HIT?** ✕

Why are you quitting this task?

- The task is too easy
- The task is too hard
- It is not clear to me what I need to do
- The tool does not work like I would expect
- Other (explained below)

Do you have any feedback to improve this HIT?

Figure 3.7: Quit questionnaire.

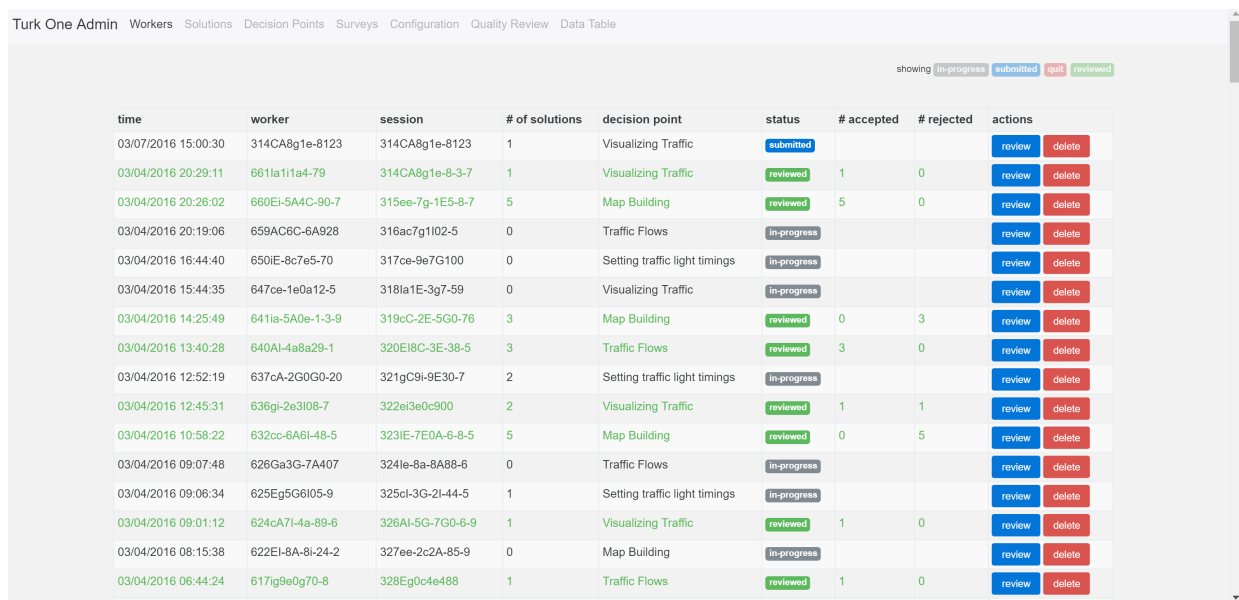
the most common reasons for a worker to abandon a task (to lower the hurdle of collecting this information). The second question was an open-ended question, where workers had the chance to give more detail about the selected reason and provide additional feedback.



### 3.8 Review Procedure

We performed the review and payment process after workers submitted their work. To support the review process, we built a complementary admin tool (shown in Figure 3.8). Its main interface lists all of the workers who entered the tool, as updated in real time. The admin tool, for internal access only, serves three main purposes. First, through the main view, we could monitor the submissions and work distribution. Second, a different view gave us access to each of the workers' work in detail, a feature that was used to evaluate and judge each of the solution alternatives. Last, once we completed judging the work of a worker, the tool allowed us to keep track of rejections and payments.

For the payment process, we worked with the Amazon Mechanical Turk dashboard. Through this dashboard we had access to the completion codes submitted by the workers. We verified in the dashboard that each provided completion code corresponded to a worker's set of solutions. Once the set of solutions was identified in our tool, we checked the judgment and paid the bonuses accordingly. Fake codes were rejected.



The screenshot shows the 'Turk One Admin' interface with a navigation menu at the top: 'Workers', 'Solutions', 'Decision Points', 'Surveys', 'Configuration', 'Quality Review', and 'Data Table'. The main content area displays a table of worker submissions with columns for time, worker, session, # of solutions, decision point, status, # accepted, # rejected, and actions. The status column uses color-coded labels: 'submitted' (blue), 'reviewed' (green), and 'in-progress' (grey). The actions column contains 'review' (blue) and 'delete' (red) buttons for each row.

time	worker	session	# of solutions	decision point	status	# accepted	# rejected	actions
03/07/2016 15:00:30	314CA8g1e-8123	314CA8g1e-8123	1	Visualizing Traffic	submitted			review delete
03/04/2016 20:29:11	661a111e4-79	314CA8g1e-8-3-7	1	Visualizing Traffic	reviewed	1	0	review delete
03/04/2016 20:26:02	660Ei-5A4C-90-7	315ee-7g-1E5-8-7	5	Map Building	reviewed	5	0	review delete
03/04/2016 20:19:06	659AC6C-6A928	316ac7g1102-5	0	Traffic Flows	in-progress			review delete
03/04/2016 16:44:40	650IE-8c7e5-70	317ce-9e7G100	0	Setting traffic light timings	in-progress			review delete
03/04/2016 15:44:35	647ce-1e0a12-5	3181a1E-3g7-59	0	Visualizing Traffic	in-progress			review delete
03/04/2016 14:25:49	641ia-5A0e-1-3-9	319cC-2E-5G0-76	3	Map Building	reviewed	0	3	review delete
03/04/2016 13:40:28	640Al-4a8a29-1	320EIBC-3E-38-5	3	Traffic Flows	reviewed	3	0	review delete
03/04/2016 12:52:19	637cA-2G0G0-20	321gC9i-9E30-7	2	Setting traffic light timings	in-progress			review delete
03/04/2016 12:45:31	636gi-2e3108-7	322ei3e0c900	2	Visualizing Traffic	reviewed	1	1	review delete
03/04/2016 10:58:22	632cc-6A6I-48-5	323IE-7E0A-6-8-5	5	Map Building	reviewed	0	5	review delete
03/04/2016 09:07:48	626Ga3G-7A407	324Ie-8a-8A88-6	0	Traffic Flows	in-progress			review delete
03/04/2016 09:06:34	625Eg5G6I05-9	325cl-3G-2I-44-5	1	Setting traffic light timings	in-progress			review delete
03/04/2016 09:01:12	624A7I-4a-89-6	326Al-5G-7G0-6-9	1	Visualizing Traffic	reviewed	1	0	review delete
03/04/2016 08:15:38	622EI-8A-8i-24-2	327ee-2c2A-85-9	0	Map Building	in-progress			review delete
03/04/2016 06:44:24	617ig9e0g70-8	328Eg0c4e488	1	Traffic Flows	reviewed	1	0	review delete

Figure 3.8: CrowdDesign admin tool.

### 3.9 Compensation

A worker who provided a valid completion code and demonstrated honest effort (i.e., clearly attempting to address the design problem) was paid \$2.00. Additionally, a worker was given a bonus of \$0.50 per valid sketch (i.e., a sketch that meaningfully illustrated the solution alternative). Further, to encourage workers, we gave an extra \$1.00 bonus for each solution alternative that met at least three out of its four task requirements. Therefore, workers were able to earn up to \$9.50 per task by providing five complete sketches. This compensation is based on the California minimal wage (\$9.00 per hour) and our estimation that completing five solution alternatives takes approximately an hour (given that these are small, partial designs, not full designs).

# 4

## Data Analysis

We performed a variety of analyses on the collected data. The key analysis, of course, concerned the diversity and quality of the solution alternatives. That is to say, we studied how many different conceptual approaches can be found within the set created both by the as attributed to the crowd and to individual workers, and what is the overall quality of these designs. However, we performed additional analyses, which helped us in understanding how the crowd operated during the experiment and where the main difficulties for workers lied.

Bellow we detail how we approached each of our analyses.

### 4.1 Diversity

First and foremost, we were interested in whether a crowd can generate a diverse set of decision points for a given solution alternative. To assess whether the solution alternatives are diverse, we applied affinity diagramming [31] to sort and group the solution alternatives within each of the decision points. For this purpose, we first printed all of the solutions for each of the decision points of the four experiments. We partitioned the activity in

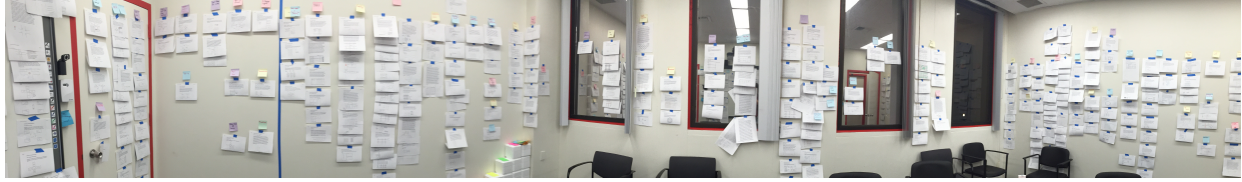


Figure 4.1: Affinity diagrams constructed during data analysis.

two, starting with the designs generated during both UI design experiments (UIplain and UIexamples), and repeating the same procedure with the solutions generated during the IC design experiments (ICplain and ICexamples). Six researchers (including the author of this thesis) engaged in this activity, which consisted of iteratively clustering designs that addressed the problem using a similar approach (again, per decision point), which then gave rise to general and overarching categories. Figure 4.1 shows affinity diagrams constructed during the analysis of the solution alternatives produced during UIplain and UIexamples.

The principal characteristic of affinity diagramming is that, instead of grouping notes in predefined categories, the work is done from the bottom up, where categories emerge from the designs themselves. For example, “Click and drag”, “Nodes”, and “Blocks” were categories that emerged for the ‘creating maps’ decision point during the UI experiments.

The more categories were found for a decision point, the more diverse the set of solution alternatives. We both examined the diversity of the full set of solution alternatives and the diversity within each individual worker’s contributions.

## 4.2 Quality

Second, we were interested in the quality of the solution alternatives. Diversity is important, but if none of the solution alternatives are of high (or even just reasonable) quality, then our approach is not an appropriate way of designing software. To determine the quality of the set of solution alternatives, each solution alternative was independently assessed by

an expert panel. We formed two panels, the first one to evaluate the solution alternatives generated during the UIplain and UIexamples experiments, and the second one to assess solution alternatives produced during the ICplain and ICexamples experiments. Each of the panels was composed of three researchers, in addition to the author of this thesis. All assessors had a background in design and were extensively familiar with the traffic simulator design problem.

Each solution alternative was rated on a one (poor) to seven (excellent) scale in three criteria: understandability, feasibility, and usability for UI design tasks, and understandability, feasibility, and elegance for IC design tasks. An overall score was calculated by averaging across the three criteria.

In addition, the panel assessed the completeness of each of the solution alternatives by tallying how many of the four requirements of the relevant decision point were met. We considered a requirement met if at least three out of four assessors declared it was met.

The process to evaluate the solution alternatives was the following. Each rater had access to all solution alternatives they needed to rate, organized in buckets per decision point. Each of the buckets contained solution alternatives from the two experiments under analysis, mixed in a random order without any identification of the experimental condition or worker. That is, all the solutions generated during UIplan and UIexamples were mixed, and the same for the solution alternatives produced during the IC experiments. In order to ensure inter-rater reliability, each of the panels had an initial meeting. During these meetings, we randomly selected five solution alternatives for each decision point. Each of the raters gave a separate score for each of the criteria, depending on the kind of task under evaluation (understandability, feasibility, and usability for UI solution alternatives, versus understandability, feasibility, and elegance for IC solution alternatives). During the same assessment, raters had to indicate for each of the requirements in the task, if it was met by the solution alternative under evaluation. We disclosed the scores after individually going through the five designs, and

discussed disagreements among raters. The remaining solutions were scored individually and reported to the research team at the end.

As a result of the evaluation, each solution alternative was judged with a quality score (numeric value from one to seven) and a completeness score (numeric value from zero to four).

### **4.3 Difficulty**

Third, we were interested in how difficult the workers found the HIT to be. For this, we looked at the reasons they provided for quitting (if they chose not to complete the HIT and simply exited our tool), examined the scores workers provided to the numerical questions on the questionnaire, and, once again, performed affinity diagramming on the open-ended feedback question to identify possible common themes in the responses. For example, “Did not understand the task” and “Did not have time to complete the task” were categories we identified during this analysis.

Second, we repeated the same procedure for workers who completed their tasks, examining both the textual feedback and numerical scores workers provided at the end of the task. In this case, “Problems with the tool” and “Task improvement suggestion” are examples of some of the categories that emerged from the analysis of workers’ responses.

### **4.4 Collaboration**

Fourth, we were interested in the degree to which workers borrowed ideas from previous participants and how the exposure to solution alternatives produced by coworkers impacted quality and diversity. We performed two different analyses. The first one looked at the use

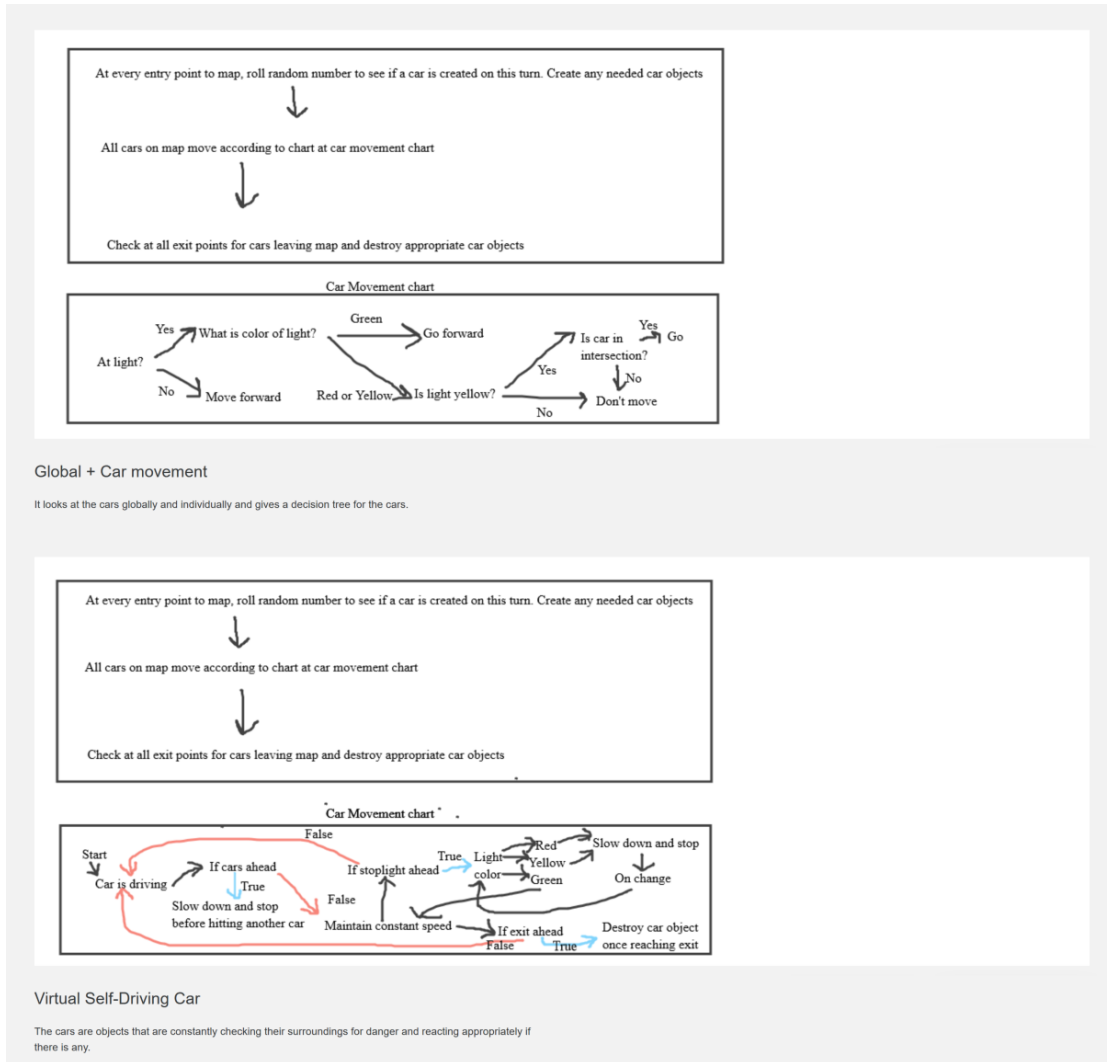


Figure 4.2: Example of the use of ‘duplicate’ feature.

The top solution alternative is the source design. The bottom solution is the design from a second worker who copied and improved their coworkers solution alternative.

of the copy and duplicate functions. The second analyzed the distance between appearances of solution alternatives that belong to the same category.

In our first analysis, we employed a conservative approach in order to examine whether copy and duplicate actually happened. Figure 4.2 illustrates an example of how a worker made use of the ‘duplicate’ function to choose a design that they then improved.

First, we identified workers, such as the one in the example, who used the copy or duplicate

features, and paired their solution alternatives with the source solution. Through our admin tool, we were able to track these instances because our platform logs all of these actions). We then looked at the quality score and requirements met for each of the solutions (both source and destination) and compared the values.

We also analyzed other aspects, such as, for instance, the average score of the source solution alternatives to see whether workers were able to identify the good designs among all of the available solution alternatives. As another example, we also calculated the distance between source and destination solutions, in order to understand how much research do workers do when looking for solution alternatives from which to copy.

Our second analysis looked at the entire set of solution alternatives generated during the UIexamples and ICexamples experiments. We were interested in, independently of the explicit use of the ‘copy’ or ‘duplicate’ feature, the degree to which previous work influenced the generation of solution alternatives. We used as input of this second analysis the analysis that looked at the diversity generated by individual workers. In this case we inspected, for each of the categories identified for a certain decision point, the distance (in number of workers) between a solution within that category and the next appearance. In doing this for each experimental condition, we sought to find possible patterns in workers behaviours.



# 5

## Results

This chapter presents the results of each of the four experiments we performed on Amazon Mechanical Turk. The first section describes general results across the four experiments. After that, we unwrap each of the experiments, starting with UIplain, followed by ICplain, and ending with the two experiments in which we revealed other workers' work (UIexamples and ICexamples). Last, we present additional analyses regarding demographics, results of qualification tests, and an analysis about the 'top designs' for each of the four experiments.

### 5.1 General Results

Table 5.1 shows the participation across the four experiments. Despite the fact that between 19% and 27% of the participants did not continue after signing the consent form, for all of the experiments the largest drop is in the qualification test (49% to 60%). An additional small drop took place with workers who passed the test but did not enter into the tool (1.25% drop on average). Between 72 and 94 workers eventually submitted their work. From these workers, some did not submit any work of honest effort (again, work that clearly attempts

	<b>UIplain</b>	<b>UIexamples</b>	<b>ICplain</b>	<b>ICexamples</b>
<b>Signed consent form</b>	<b>1069</b>	<b>875</b>	<b>1474</b>	<b>836</b>
<b>Quit before taking qualification test</b>	205 (19%)	191 (22%)	375 (25%)	229 (27%)
<b>Failed the test</b>	580 (54%)	431 (49%)	886 (60%)	456 (55%)
<b>Passed the test</b>	284 (27%)	253 (29%)	232 (16%)	151 (18%)
<b>Entered the platform</b>	282 (26%)	225 (26%)	220 (15%)	151 (18%)
<b>Submitted completion code</b>	88 (8%)	87 (10%)	94 (6%)	72 (9%)
<b>Accepted work</b>	78 (7%)	80 (9%)	76 (5%)	66 (8%)

Table 5.1: Number of workers at each step of the experiment.

name: accident

explanation: sue to sudden break of yellow car there is a pile up on road

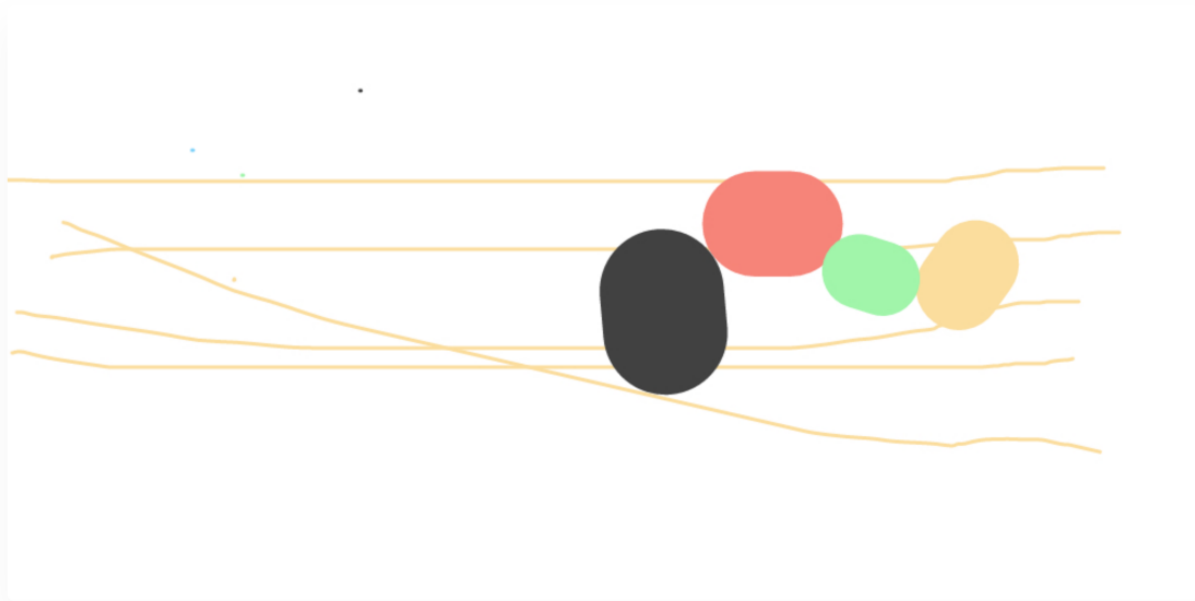


Figure 5.1: Example of a solution alternative which did not show honest effort. The worker provided this solution for experiment ICplain, for the ‘moving cars’ decision point.

to address the given design problem), meaning we rejected their work. An example of such a rejected solution alternative is shown in Figure 5.1. The final row of the table, then, shows the number of workers whose work we accepted, because they created at least one viable solution alternative.

Participants whose work was accepted had different backgrounds. Table 5.2 shows the demographics across the four experiments. We provided the following options to workers: Undergraduate Student, Graduate Student, Professional Developer, Professional UI/UX de-

	<b>UIplain</b>	<b>UIexamples</b>	<b>ICplain</b>	<b>ICexamples</b>
<b>Undergraduate student</b>	12 (15%)	12 (15%)	13 (17%)	13 (20%)
<b>Graduate Student</b>	11 (14%)	7 (9%)	7 (9%)	2 (3%)
<b>Professional Developer</b>	17 (22%)	12 (15%)	22 (29%)	26 (39%)
<b>Professional UI/UX Designer</b>	2 (3%)	2 (3%)	1 (1%)	0 (0%)
<b>Hobbyist</b>	24 (31%)	36 (45%)	24 (32%)	19 (29%)
<b>Other</b>	12 (15%)	11 (14%)	9 (12%)	6 (9%)
<b>Total</b>	<b>78</b>	<b>80</b>	<b>76</b>	<b>66</b>

Table 5.2: Demographics.

signer, and Hobbyist. If any worker did not feel represented by these options, they could select “Other” and detail their experience. Some of the most frequent examples of what workers indicated in this field are professor (22 along the four experiments), system administrator (17), software tester (10), business analyst (9), and researcher (7). In most of the experiments, the majority of our participants were hobbyists. The exception is experiment ICexamples, for which 39% of the participants were self-declared professional developers. There was also remarkable participation of professional developers and designers in the three remaining experiments (25%, 18%, and 30%, respectively) and a notable difference in software developer participation between the UI design tasks and the IC design tasks (higher in IC design). Of note is, too, that we hoped for more UI professionals to participate.

Each UI experiment lasted seven days, during which 181 and 187 solution alternatives were collected, respectively. The IC experiments lasted 11 days and 14 days, respectively, during which 158 and 115 solution alternatives were collected. On average, each worker provided two-and-a-half solutions for the UI tasks and two solutions for the IC tasks. Tables 5.3 and 5.4 show the distribution of how many accepted solution alternatives were produced by how many workers during each experiment. We note that the majority of workers produced one or two accepted solution alternatives. A good number (around 32% per experiment) of workers provided three or more solution alternatives. This is not true for the last experiment (ICexamples), where only 10 (15%) workers submitted two or more accepted solution alternatives.

		#workers with $x$ solutions accepted per experiment											
		UIplain					UIexamples						
		1	2	3	4	5	total #solutions	1	2	3	4	5	total #solutions
<b>dp 1.1</b>	<b>map creation</b>	5	5	3	2	6	<b>62</b>	2	5	3	6	7	<b>80</b>
<b>dp 1.2</b>	<b>setting of traffic light timings</b>	9	6	2	1	3	<b>46</b>	11	2	3	0	1	<b>29</b>
<b>dp 1.3</b>	<b>determining the flow of traffic</b>	6	4	2	0	3	<b>35</b>	9	3	3	0	2	<b>34</b>
<b>dp 1.4</b>	<b>visualization of the state of the simulation</b>	12	5	2	0	2	<b>38</b>	11	6	4	1	1	<b>44</b>
<b>Total</b>		<b>32</b>	<b>20</b>	<b>9</b>	<b>3</b>	<b>14</b>	<b>181</b>	<b>33</b>	<b>16</b>	<b>13</b>	<b>7</b>	<b>11</b>	<b>187</b>

Table 5.3: Number of solution alternatives submitted per decision point (UI design).

		#workers with $x$ solutions accepted per experiment											
		ICplain					ICexamples						
		1	2	3	4	5	total #solutions	1	2	3	4	5	total #solutions
<b>dp 2.1</b>	<b>representing cars</b>	9	5	4	0	1	<b>36</b>	11	4	1	1	3	<b>41</b>
<b>dp 2.2</b>	<b>moving cars</b>	10	2	3	1	4	<b>47</b>	12	2	1	0	0	<b>19</b>
<b>dp 2.3</b>	<b>representing the road system</b>	10	5	3	1	0	<b>33</b>	6	6	0	0	1	<b>23</b>
<b>dp 2.4</b>	<b>changing the colors of traffic lights</b>	7	5	2	1	3	<b>42</b>	9	6	2	0	1	<b>32</b>
<b>Total</b>		<b>36</b>	<b>17</b>	<b>12</b>	<b>3</b>	<b>8</b>	<b>158</b>	<b>38</b>	<b>18</b>	<b>4</b>	<b>1</b>	<b>5</b>	<b>115</b>

Table 5.4: Number of solution alternatives submitted per decision point (IC design).

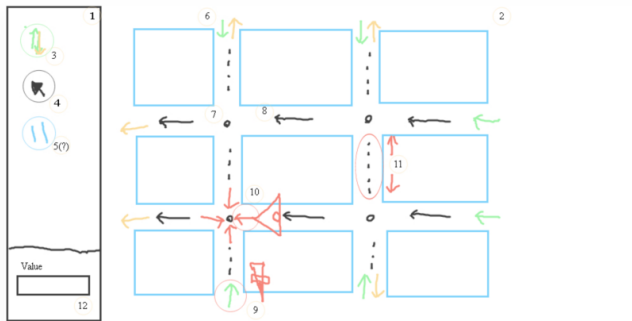
Each worker took, on average, six minutes for each provided solution for the UI experiments and five minutes for the IC experiments. Despite the average being only five to six minutes, a few workers spent between 30 and 45 minutes solving the task.

The kind of work each worker produced varied from worker to worker. Some workers chose to draw a very simple sketch and provide a very long and detailed textual description (see Figure 5.2 (A)). Others opted for sketching a very detailed mockup or diagram together with minimal text (see Figure 5.2 (B)). Yet others were more balanced.

Within the UI design experiments, we mainly found drawings with UI elements such as buttons, panels, and radio buttons. Some workers drew just elements necessary to describe their solutions, whereas some others provided an entire depiction of what the screen would look like. In the case of the IC experiments, workers chose to express their solutions through class diagrams, flow diagrams, pseudocode, state machines, other informal notations, and combinations thereof (see Figures 5.3 (A) and 5.3 (B)).

name: Graphical Traffic Flow Editor

explanation: I goofed and drew a 3x3 grid with 4 intersections instead of a 3x4 grid with 6, but the principles should hold up the same regardless of map size or shape. (1) - Toolbar (2) - Map editing area (3) - Tool for placing entrances and exits, possibly left-click to place entrance, right-click to place exits or something to that effect (4) - Typical object select tool (5) - Tool for placing roads. I wasn't sure whether this interface is intended to be able to construct maps or simply change the traffic flows on existing maps, so functionality of this tool won't be expanded on much. Maybe a simple line tool type mechanism, where you click to place two nodes or select existing intersection nodes and a road is created between them (6) - Green arrows represent placed entrances and yellow arrows represent exits (7) - Intersection nodes. Should be automatically defined wherever two roads intersect. Maybe there could be a tool to manually place them as well? Not sure what impact or use that could have, except the possibility to create "garages" in areas between intersections, which I go over later (8) - Marker representing a segment of road. Dashed lines in this sketch represent 2-way streets, lines with arrows are one-way (9) - When an entrance is selected, a slider appears allowing for adjustment of the amount of traffic that will enter through that entrance. Maybe it also makes the arrow grow bigger/thicker the higher the factor so that it's easy to tell at a glance the relative traffic of all entrances without having to select each one and check (10) - When an intersection is selected, arrows appear from each oncoming traffic direction (that one on the left side isn't actually valid because it's a one-way street. It probably wouldn't be a bad idea for the system to detect this, but it shouldn't do any harm if it doesn't since that arrow will simply affect the Zero cars coming that way). When you select an arrow, a UI element appears that allows you to adjust the probability that approaching cars will head in a certain direction. In this case if the slider is moved to one of the points of the triangle, that means a 100% chance cars will move in that direction, while leaving it in the middle of the right face of the triangle means an equal probability of any direction. This could be expanded for an N-way intersection to be any (N-1)-gon, but that could get messy and unintuitive pretty quick. Then again, anything more than a 5-way intersection is pretty unlikely in a realistic road map. Turn probability should NOT overwrite one-way streets however, unless a goal of the program is to model reckless/law-breaking drivers as well. (11) - When a road segment is selected, you can click and drag parallel to the road to make it a one-way street. A one-way street can be made two-way again by dragging in the opposite direction. Alternatively there could actually be two UI arrows with a dot in between. Possibility for more traffic flow control here, but most of what I can think of is already covered by entrances and intersections. (Other than vehicle speed, maybe? That's something I didn't consider at all until just now, hmm...) (12) - As with most editing programs, the option to look at and manually edit the actual values of whatever you're adjusting would be provided too, so you can be more precise with your flow settings if you like. -- Not shown in the sketch, but there could be a possibility of adding entrances and exits in between intersections as well, essentially garages on the blocks in between. These would simply be placed with the entrance/exit tool by clicking on the desired spot. Entrances would have the same entrance rate control slider, but in addition both entrances and exits could have an intersection-like probability controller that determines which way cars will turn when coming out of entrances, and the likelihood that passing cars will turn to take an exit they pass by. -- The ability to place entrances and exits and one-way streets so freely certainly presents the possibility of the user inducing conflict, but that might not be such a bad thing - If the map breaks somewhere, you should wind up seeing where it breaks sooner or later. But there could also be some kind of conflict handler, that's not really the part I'm meant to be worrying about here anyway. -- This interface might not be the most precise but it should be fairly approachable and intuitive to use. Theoretically there shouldn't be any major issues with non square-grid-based road systems either, and curved roads should be handled in the same way as straight road segments as far as directions, intersections, etc



name: Graphical representation of traffic at 6 traffic lights

explanation: I decided that the user has to see what is going on in order to make a decision about what to do with the traffic light timings. So, I decided to graphically represent it in a way that would make it easier to see what was happening.

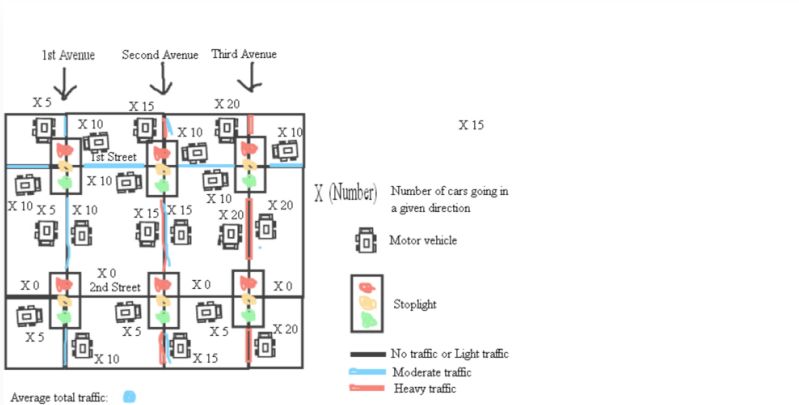


Figure 5.2: Designs submitted by workers for UI design task.

name: Multi-threaded logic - pseudo code

explanation: This pseudo code instantiates a new thread for each car. Each car travels towards next intersection where it determines its direction (left, right, straight, U-turn) based on its intended destination. It then checks if the traffic signal is green for the direction it needs to go, waiting if necessary. This continues until the car reaches its destination. Each thread exits when the destination for the corresponding car is reached.

```
Initialize Starting Point and Destination of all Cars
Start a new Thread of execution for each Car

Car Thread Logic (one thread per car)
While not reached Destination
    Move forward till next intersection or Destination reached
    If Destination Reached
        End this Thread
    Based on destination, determine direction to take at this intersection
    While Traffic Light for required direction is not Green
        Wait for a second
    Start moving in the required direction, turning if required
```

A

name: Intersections as the center of the world (Intersectioncentrism)

explanation: Overview: Map contains list of roads and intersections. Each road has a length and traffic load value, as well as a pointer to each of the two intersections it comes to. In addition, a speed limit value helps determine average time it takes to cross the stretch of road. Each intersection has a pointer to each of the roads it connects to (assuming North, South, East, West) as well as four traffic lights for each direction. Each traffic light has its own current state (green, yellow, red) as well as a timer value to help change to the next required value. Rather easy first go. Both roads and intersections are full out objects which carry their own workload.

```
MAP
ROAD Roads[]
INTER Intersections[]
```

```
ROAD
Int distance
Int speed_limit
Int traffic_load
INTER* first, second
```

```
INTER
ROAD* r_north, r_east,
    r_west, r_south
TLIGHT tl_north, tl_east,
    tl_west, tl_south
```

```
TLIGHT
STRING state
Int timer
```

Overview:

- Map contains list of roads and intersections.
- Each road has a length and traffic load value, as well as a pointer to each of the two intersections it comes to. In addition, a speed limit value helps determine average time it takes to cross the stretch of road.
- Each intersection has a pointer to each of the roads it connects to (assuming North, South, East, West) as well as four traffic lights for each direction.
- Each traffic light has its own current state (green, yellow, red) as well as a timer value to help change to the next required value.

B

Figure 5.3: Designs submitted by workers for IC design task.

## 5.2 User Interface Design Without Examples (UIplain)

### 5.2.1 Diversity

Table 5.5 presents the results of a first analysis with respect to diversity. Each decision point had at least ten categories of conceptually different solution alternatives, which is an important result as it shows that the primary benefit of competitions (generating alternatives) can be preserved in a microtask setting.

	<b>decision point description</b>	<b>#categories</b>
<b>dp 1.1</b>	<b>creating maps</b>	11
<b>dp 1.2</b>	<b>setting timing of traffic lights</b>	13
<b>dp 1.3</b>	<b>determining the flow of traffic</b>	10
<b>dp 1.4</b>	<b>visualizing the state of the simulation</b>	14

Table 5.5: Number of resulting categories per decision point for UIplain experiment.

Our second analysis looked at diversity of the solution alternatives per worker, per decision point. Figure 5.4 shows the result for the ‘creating maps’ decision point. Each of the rows represents a worker, with the name and color of each cell (up to five) illustrating the category of the corresponding solution alternative. For instance, worker pMB5 submitted five solution alternatives, with three of them belonging to the same conceptual approach – creating the map in a “Pencil-Like (Draw)” manner. Most, though not all, of the workers who submitted multiple solution alternatives ended up submitting a homogeneous set. Workers pMB2, pMB11, and pMB21 are exceptions, with especially worker pMB2 submitting four conceptually very different solution alternatives. This confirms that it is important for diversity to involve multiple workers.

At the same time, Figure 5.5 brings up the question if asking for fewer alternative solutions per worker might lead to the same result in terms of overall diversity. A careful examination of Figure 5.4, however, reveals that pMB6 introduces a unique new category with solution

Worker ID	Category				
pMB1	Blocks				
pMB2	Click and drag	Build as you go	Blocks	GPS	
pMB3	Traffic Simulation				
pMB4	Blocks	Assisted Drawing			
pMB5	Click and drag	Pencil-like (Draw)	Pencil-like (Draw)	Pencil-like (Draw)	UI layout and extra features
pMB6	UI layout and extra features	UI layout and extra features	Automated using input	UI layout and extra features	UI layout and extra features
pMB7	Map Only	Map Only	Map Only	Map Only	Map Only
pMB8	Build as you go				
pMB9	Nodes	Nodes			
pMB10	Click and drag	Click and drag			
pMB11	Pencil-like (Draw)	Blocks	Blocks	Build as you go	
pMB12	Click and drag	Click and drag			
pMB13	Map Only	Map Only	Map Only	Map Only	Map Only
pMB14	Nodes	Nodes	Click and drag		
pMB15	UI layout and extra features	UI layout and extra features	UI layout and extra features	UI layout and extra features	Click and drag
pMB16	Click and drag				
pMB17	Assisted Drawing	Assisted Drawing	Pencil-like (Draw)		
pMB18	Blocks	UI layout and extra features			
pMB19	Blocks	UI layout and extra features	UI layout and extra features	UI layout and extra features	UI layout and extra features
pMB20	Map Only				
pMB21	Build as you go	Blocks	Pencil-like (Draw)		

Figure 5.4: Diversity per worker for ‘creating maps’ design (UIplain).

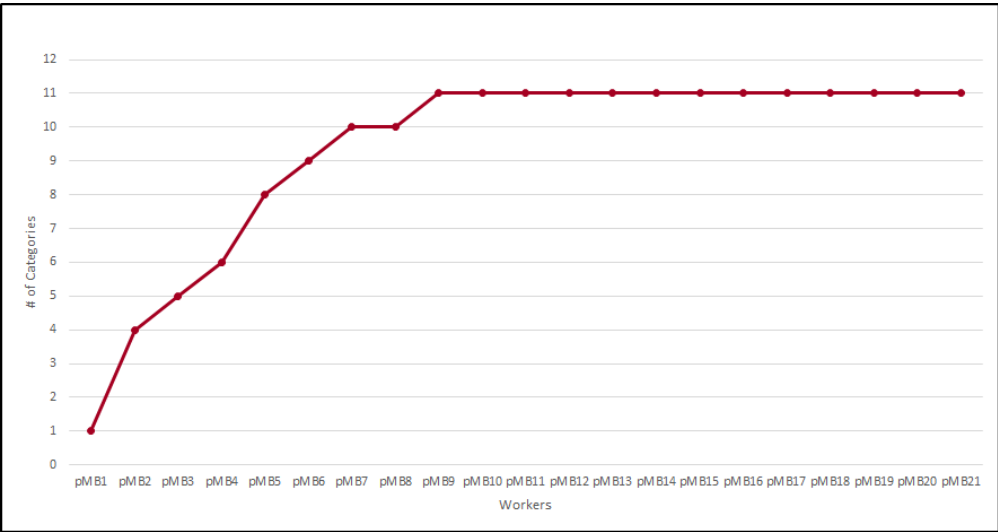


Figure 5.5: Cumulative number of unique categories provided by workers for ‘creating maps’ design (UIplain).

alternative 3 and pMB2 another new category with solution alternative 4. Repeating the analysis for all decision points, to retain the same number of categories, a minimum of 4, 5, 3, and 4 solution alternatives per worker is needed, respectively. These results suggest that, in addition to involve multiple workers, there is some value in asking individual workers for multiple solution alternatives. Diversity also benefits from some key workers who provide multiple conceptual approaches when contributing with more than one solution alternative.



We also wondered if fewer workers could provide the same diversity as the overall set. Examining Figure 5.5, which shows how many unique categories were identified at each point in time, it can be observed that worker pMB9 is the last worker to introduce a completely new approach to solve the ‘creating maps’ design problem. Repeating this analysis across all decision points, to get the same number of categories, a minimum of 9, 19, 11, and 21 workers is needed, respectively. We can observe that, on one hand, there are two decision points that saturate with around 10 workers, and, on the other hand, two other decision points which saturate much later, requiring on the order of 20 workers. This represents a significant difference, and possibly explained by the nature of each of the problems stated in each of the decision points. Perhaps it is more easily to think of different solution alternatives for some kinds of decision points than others.

## 5.2.2 Quality

Table 5.6 presents the results of our first analysis examining the quality of the work. For each decision point, we counted the number of solution alternatives that score across the range of 0–1 (lowest quality) to 6–7 (highest quality), as well as the totals and averages. It is readily observed that work of the highest quality is relatively absent, though 39 solution alternatives were rated 4–5 and 18 were rated 5–6, indicating an overall good quality for those solution alternatives.

	<b>#solution alternatives with quality <math>x</math>-<math>y</math></b>							<b>average quality</b>
	<b>0-1</b>	<b>1-2</b>	<b>2-3</b>	<b>3-4</b>	<b>4-5</b>	<b>5-6</b>	<b>6-7</b>	
<b>dp 1.1</b>	12	2	8	16	15	8	1	<b>3.3</b>
<b>dp 1.2</b>	3	13	4	15	9	1	1	<b>3.0</b>
<b>dp 1.3</b>	22	10	7	9	3	3	1	<b>2.9</b>
<b>dp 1.4</b>	1	11	4	3	12	6	1	<b>3.4</b>
<b>total</b>	18	36	23	43	39	18	4	<b>3.2</b>

Table 5.6: Quality of solution alternatives (UIplain).

Table 5.7, however, tells a somewhat different story. It shows a count of the number of requirements met, as decided by the same assessors. The majority of solution alternatives (71) met three of the requirements and another 25 met all four of the requirements.

	#requirements met					average
	0	1	2	3	4	
<b>dp 1.1</b>	12	4	3	35	8	<b>2.4</b>
<b>dp 1.2</b>	0	19	12	13	2	<b>2.0</b>
<b>dp 1.3</b>	2	6	11	10	6	<b>2.3</b>
<b>dp 1.4</b>	4	10	2	13	9	<b>2.3</b>
<b>total</b>	18	39	28	71	25	<b>2.3</b>

Table 5.7: Number of requirements met (UIplain).

We crossed both results (quality score and requirements met) to analyze the proportion of results that combine good quality and completeness. A careful examination of Table 5.8 shows us that 55 solution alternatives are located in the bottom right of the table (shaded area, which represents 30.4% of the total number of solutions). Overall, we are encouraged by this result, even though it is clear (and not unexpected) that what might be considered ‘wasted effort’ exists. Having a great number of solution alternatives within the group of the ‘top designs’ suggests that the crowd did achieve the creation of a solid basis for next steps in the work with the morphological chart.

A second reason why we are encouraged harkens back to the issue of diversity: many of the ‘meeting two requirements’ solution alternatives fall in different categories and, as such,

Scores	#requirements met					total
	0	1	2	3	4	
<b>0-1</b>	13	5	0	0	0	<b>18</b>
<b>1-2</b>	5	23	5	3	0	<b>36</b>
<b>2-3</b>	0	9	4	9	1	<b>23</b>
<b>3-4</b>	0	2	13	25	3	<b>43</b>
<b>4-5</b>	0	0	4	25	10	<b>39</b>
<b>5-6</b>	0	0	1	9	8	<b>18</b>
<b>6-7</b>	0	0	1	0	3	<b>4</b>
<b>total</b>	18	39	28	71	25	<b>181</b>

Table 5.8: Crosstabulation of quality scores and requirements met.

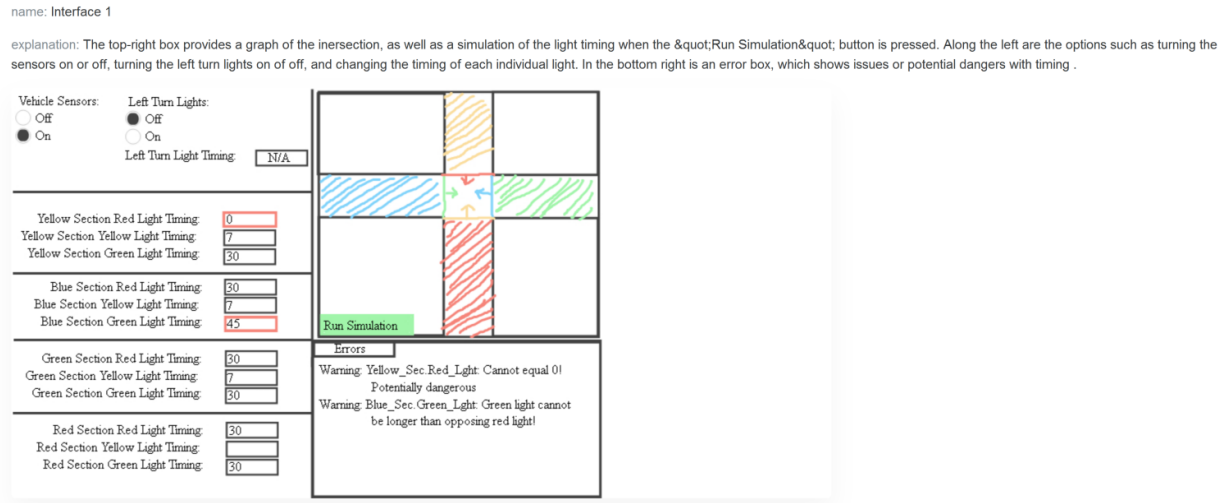


Figure 5.6: Example of a top solution provided for ‘setting timing of traffic lights’ decision point (Uplain).

represent out-of-the-box solutions that may well be useful to the designer of a system, or that could serve as a basis for improved solution alternatives, with an extra step of iteration.

A third reason is that some of the 5-6 and 6-7 solution alternatives are quite exceptional in describing complete and innovative solutions. For instance, consider the solution in Figure 5.6. It scored 6.75 as quality, and it meets all four of the requirements. It is especially useful, because it is a very clear and understandable solution alternative, provides a very simple mechanism for users to set traffic light timings, considers error information, and the approach the worker is using is very appropriate for the particular problem.

We looked at the correlation between score and time spent on a solution alternative. We found that there was a moderate positive correlation between these two variables ( $r=0.26$ ,  $p<0.001$ ). We also looked at sketch complexity (approximated by the number of strokes in the sketch area), relating this variable with the quality scores given. We found a stronger, but still moderate relationship between sketch complexity and quality scores ( $r=0.435$ ,  $p<0.0001$ ). Finally, we examined the description length (measured in amount of characters used in the description text field to explain the solution). Again, there is a moderate correlation between description length and solution quality ( $r=0.353$ ,  $p<0.0001$ ). These results suggest that there

is high variability between solutions, and not a ‘rule’ we can apply to automatically filter and detect good results. This is in line with what we already highlighted: some workers choose to be more visual, others more textual. Workers of each nature appear to be providing high quality solution alternatives.

### 5.2.3 Difficulty

Most of the 282 workers who passed the qualification test and entered the tool to begin their UI design task either did not finish the HIT (68.8%) or had all of their results rejected (3.5%). Of those who quit, many left the tool without leaving any feedback. Of those who did provide feedback, however, many expressed that the task was not clear or too hard. One of the workers commented: “I did not realize how much programming knowledge was actually required to complete this HIT. I don’t have any clue what to put in the sketch boxes.” Another asked for more information or examples: “I have no idea what this task is asking me to do. An example would have been helpful.”

Numerically, workers rated their ability to complete the entire task at an overall difficulty level of 5.17 (out of 7), difficulty level of the decision point at 4.72 (out of 7), and adequacy of support by the tool at 4.94 (out of 7). Overall, these numbers corroborate that the HIT was not easy, but also that our tool could use improvement (which the written feedback also highlighted). These improvements mostly refer to features of the sketching tool, such as straight lines, arrows, and availability to fill objects. Yet, quite a few workers also commented on having fun, and finding our HIT to be intriguing among the often monotone HITs of Amazon Mechanical Turk.

## 5.3 Internal Code Design Without Examples (ICplain)

### 5.3.1 Diversity

Table 5.9 presents the results of a first analysis with respect to diversity for the ICexamples tasks. Again, as in UIplain, each decision point had at least ten categories of conceptually different solution alternatives and a maximum of 14 different categories.

	<b>decision point description</b>	<b>#categories</b>
<b>dp 2.1</b>	<b>representing cars</b>	10
<b>dp 2.2</b>	<b>moving cars</b>	10
<b>dp 2.3</b>	<b>representing the road system</b>	10
<b>dp 2.4</b>	<b>changing the colors of traffic lights</b>	14

Table 5.9: Number of resulting categories per decision point for ICplain experiment.

These similarities of the results of UIplain and across ICplain, both in terms of decision points and in terms of experiments, hints that diversity saturates after a while. Despite having a different number of collected solution alternatives for each of the decision points, the number of categories remain relatively constant: between 10 and 14 categories. It seems to indicate that more solution alternatives does not guarantee an increment in diversity. This could be illustrated with the example of the ‘moving cars’ decision point. Despite being the decision point with the greatest number of solution alternatives (47) for the ICplain experiment, it is not the decision point with the best diversity (10 categories).

We examined the diversity in the categories of the solution alternatives each worker provided. Figure 5.7 shows the result for the ‘moving cars’ decision point for the ICplain experiment. Each of the rows represents a worker, with the name and color of each cell (up to five) illustrating the category of the corresponding solution alternative. Most but not all of the workers who submitted more than one solution ended up submitting variants of the same solution that were conceptually still very similar. Some exceptions exist. For instance, worker pDR2 submitted three conceptually different solutions. Similar to UIplain, this confirms that

Worker ID	Category				
pDR1	Traffic travel update loop	Simulator model	Car lane logic	Car lane logic	Car lane logic
pDR2	Intersection logic	Traffic lights rules	Car lane logic		
pDR3	Car lane logic				
pDR4	Simulator model	Graphic map			
pDR5	Intersection logic	Car travel	Car travel	Car logic	
pDR6	Multi thread	Car logic	Car logic	Traffic travel update loop	Simulator model
pDR7	Traffic lights rules				
pDR8	Car position centric				
pDR9	Car logic				
pDR10	Traffic travel update loop	Traffic travel update loop	Traffic travel update loop	Traffic travel update loop	Traffic travel update loop
pDR11	Intersection logic	Car logic	Car logic		
pDR12	Car position centric	Car logic	Car logic		
pDR13	Traffic travel update loop	Car logic	Car logic	Traffic travel update loop	Car logic
pDR14	Traffic travel update loop	Intersection logic			
pDR15	Traffic travel update loop				
pDR16	Car travel	Graphic map			
pDR17	Traffic travel update loop				
pDR18	Car travel				
pDR19	Car logic				

Figure 5.7: Diversity per worker for ‘moving cars’ design (ICplain).

is important to involve multiple workers in the creation of alternatives.

We also looked at whether fewer solutions per worker could be sufficient to generate the same level of diversity. In the case of the ‘moving cars’ decision point, asking for two solution alternatives per worker would have been sufficient to generate ten categories. Across all decision points, to get the same number of categories, we need a minimum of two, two, two and four, respectively, which suggests that fewer solution alternatives per worker appear to be sufficient for this kind of design as compared to UI design. Perhaps it is possible to more easily think of alternative data structures, algorithms, or internal representations.

Examining Figure 5.8, which shows how many unique categories were identified at each point in time, it can be observed that worker pDR8 is the last worker to introduce a completely new approach to address the ‘moving cars’ design problem.

Across all decision points, to get the same number of categories, a minimum of 13, 8, 18, and 16 workers is needed, respectively. Compared to UIplain, less workers are needed to achieve the same diversity. At the same time, while for UIplain we identified two groups of decision points (one group needing about 10 workers, and the second one around 20 workers), the values in ICplain are more spread out in range.

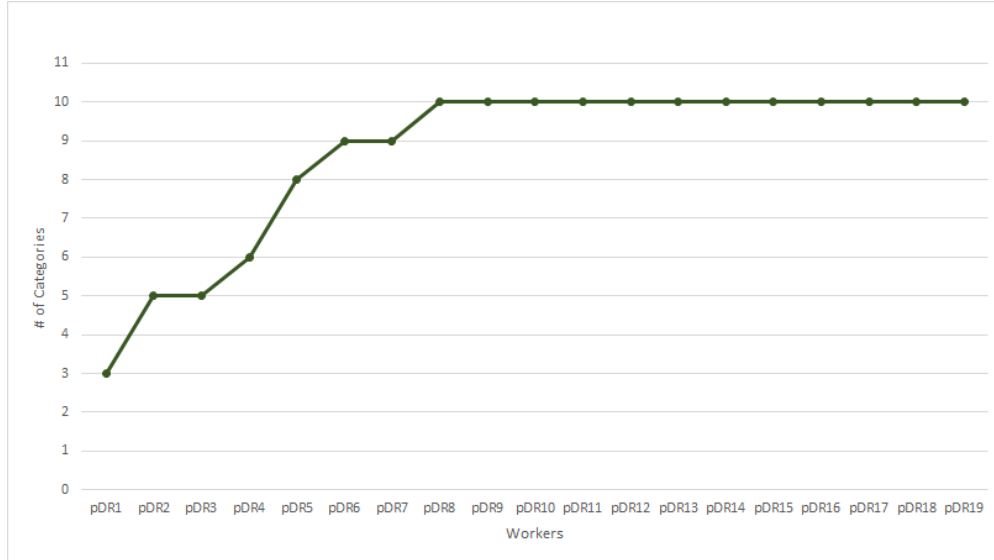


Figure 5.8: Cumulative number of unique categories provided by workers for ‘moving cars’ design (ICplain).

### 5.3.2 Quality

Table 5.10 presents the results of our first analysis examining the quality of the submitted work for the ICplain experiment. The absence of work of the highest quality is more pronounced compared to UIplain, though 41 alternative solutions were rated 4-5 and 11 were rated 5-6, indicating that an overall good level of quality is present. Using the independent samples t-test, we did not find any significant difference in overall quality between UIplain and ICplain ( $p < 0.58$ ).

Table 5.11 shows a count of the number of requirements met, as decided by the same assessors

	#solution alternatives with quality $x-y$							average quality
	0-1	1-2	2-3	3-4	4-5	5-6	6-7	
<b>dp 2.1</b>	3	2	9	9	10	3	0	3.3
<b>dp 2.2</b>	1	7	6	16	13	4	0	3.4
<b>dp 2.3</b>	13	5	3	3	7	2	0	2.4
<b>dp 2.4</b>	4	7	6	12	11	2	0	3.0
<b>total</b>	<b>21</b>	<b>21</b>	<b>24</b>	<b>40</b>	<b>41</b>	<b>11</b>	<b>0</b>	<b>3.1</b>

Table 5.10: Quality of solution alternatives (ICplain).

	#requirements met					average
	0	1	2	3	4	
<b>dp 2.1</b>	7	4	6	10	9	2.3
<b>dp 2.2</b>	9	10	6	8	14	2.2
<b>dp 2.3</b>	17	2	7	6	1	1.2
<b>dp 2.4</b>	13	5	9	11	4	1.7
<b>total</b>	<b>46</b>	<b>21</b>	<b>28</b>	<b>35</b>	<b>28</b>	<b>1.9</b>

Table 5.11: Number of requirements met (ICplain).

who rated the quality of the ICplain solution alternatives. Even though there is a high number of solutions that address either three (35) or four (28) of the requirements, the majority of solution alternatives (46) did not meet any of the four requirements. Compared to the distribution in UIplain solution alternatives, the difference in requirements met is significant ( $p < 0.008$ ): on average, the solution alternatives of the workers who participated in ICplain met fewer requirements. The main difference lies in the increment of workers not meeting any of the four requirements, meaning that workers did not address the design problem presented in the task. This result could suggest that the overall goal of each of the tasks in the ICplain experiment is harder or more complex, as compared to the tasks in UIplain. It seems that an understanding threshold exists for understanding IC design tasks, which is higher than the one for UI design tasks.

We plot the intersection of quality scores and requirements met in Table 5.12, with 25.3% of the solution alternatives scoring three or higher and meeting three or more requirements, at the same time. This is a lower percentage than UIplain. However, this difference is not statistically significant ( $p < 0.30$ ).

Different from UIplain, where there was a moderate positive correlation between time spent and quality score, there was no significant relationship between time spent and quality score for ICplain ( $r = -0.16$ ,  $p < 0.063$ ). Contrary to the results found for UIplain, we found low strength but a statistically significant negative correlation between the sketch complexity (approximated by the number of strokes in the sketch area) and quality score ( $r = -0.283$ ,



Scores	#requirements met					total
	0	1	2	3	4	
<b>0-1</b>	21	0	0	0	0	<b>21</b>
<b>1-2</b>	19	2	0	0	0	<b>21</b>
<b>2-3</b>	6	11	4	3	0	<b>24</b>
<b>3-4</b>	0	8	12	12	8	<b>40</b>
<b>4-5</b>	0	0	12	13	16	<b>41</b>
<b>5-6</b>	0	0	0	7	4	<b>11</b>
<b>6-7</b>	0	0	0	0	0	<b>0</b>
<b>total</b>	<b>46</b>	<b>21</b>	<b>28</b>	<b>35</b>	<b>28</b>	<b>158</b>

Table 5.12: Crosstabulation of quality scores and requirements met (ICplain).

$p < 0.001$ ), and low strength but a statistically significant positive correlation between description length (measured in amount of characters used in the description text field to explain the solution) and quality score. Results suggest, once again, that there is high variability between solutions, and that here is no ‘rule’ we could apply to automatically filter and detect good results. However, contrary to UIplain, there seems to exist a, still weak, indication that text is more important over graphics to produce a high quality solution alternative for IC design tasks.

### 5.3.3 Difficulty

Out of the 125 workers who did not finish the task, 35 left feedback expressing the reason. Once again, the most popular reason for quitting was the task’s lack of clearness. One of the workers said “[...] I find I’m at a loss with how to move forward at this time. I suppose that’s the exact problem you’re trying to tackle, just wish I had some inclination where to start”. We also observed in the written feedback that workers found the HIT was more time consuming than the average task they can find on Amazon Mechanical Turk: “I started this task too late at night and I don’t think I have enough time to do it right.”

Workers who finished the task and submitted accepted work rated their ability to complete the entire task at a difficulty level of 4.80 (out of 7), difficulty level of the decision point at

4.65 (out of 7), and adequacy of support by the tool at 4.32 (out of 7). Overall these numbers again indicate that the HIT was not easy, but that this kind of task was perceived as slightly less difficult than task of UIplain (ability to complete the entire task  $p < 0.04$ ; difficulty level of the decision point  $p < 0.11$ ; adequacy of support by the tool  $p < 0.07$ ).

Once more, we found a few workers commenting on having fun: “Overall it was a fun challenge for me and if the intent was to leave things open for interpretation for more brainstorming, then it did a good job”.

## 5.4 Influence of Examples (UIexamples and ICexamples)

### 5.4.1 Diversity

We repeated all of the previous analysis for the two experiments in which we revealed other workers’ work as examples. Tables 5.13 and 5.14 show the total number of categories and the number of unique categories identified in each decision point, for each of the four experiments. Looking first at the total number of categories, we observe a reduction in the number of categories, for all eight decision points, with the exception of the ICexamples for ‘representing cars’ decision point. This suggests that the exposition to examples negatively affects the diversity, as workers who are exposed to examples create less diverse sets of solution alternatives than workers working on their own.

Even though the crowd produced a less diverse set when exposed to examples, we noticed that most of the categories overlap with categories identified in the previous experiments, with some categories even being unique. There are two interesting observations out of these results. First we note that, even though in the second experiment we recruited new

		User Interface design #categories					
		UIplain		UIexamples		Delta	Total Unique
		Total	Unique	Total	Unique		
<b>dp 1.1</b>	<b>creating maps</b>	11	2	11	2	0	<b>13</b>
<b>dp 1.2</b>	<b>setting timing of traffic lights</b>	13	3	10	0	-3	<b>13</b>
<b>dp 1.3</b>	<b>determining the flow of traffic</b>	10	3	7	0	-3	<b>10</b>
<b>dp 1.4</b>	<b>visualizing the state of the simulation</b>	14	6	9	1	-5	<b>15</b>

Table 5.13: Number of resulting categories per decision point (UIexamples).

		Internal Code design #categories					
		ICplain		ICexamples		Delta	Total Unique
		Total	Unique	Total	Unique		
<b>dp 2.1</b>	<b>representing cars</b>	10	1	11	2	+1	<b>12</b>
<b>dp 2.2</b>	<b>moving cars</b>	10	3	7	0	-3	<b>10</b>
<b>dp 2.3</b>	<b>representing the road system</b>	10	3	8	1	-2	<b>11</b>
<b>dp 2.4</b>	<b>changing the colors of traffic lights</b>	14	2	13	1	-1	<b>15</b>

Table 5.14: Number of resulting categories per decision point (ICexamples).

workers and they did not have access to solution alternatives from UIplain and ICplain respectively, the same conceptual approaches emerged. Therefore, results are line with what we already highlighted: solution alternatives for decision points saturate. However, still, new conceptual approaches are provided by the second group of workers, but not as many as in the respectively experiment without examples.

Replicating the second analysis for diversity, we are not only able to see the diversity produced by each individual worker, but also how each worker influences the approach of the work of the workers that follow them. Figure 5.9 and Figure 5.10 show the diversity for the ‘creating maps’ (UI) and ‘moving cars’ (IC) decision points, respectively. It is interesting to see how in Figure 5.4 the ‘map only’ approach repeats worker after worker, until eMB7 introduces new approaches, after which these influence the approaches taken by the next workers.

Figures 5.11 and 5.12 reinforce the observation by showing how, in general, worker  $n$  much more frequently repeats a past approach to solve the task. In these figures, the number in the worker ID indicates the order in which workers submitted their solution alternatives (i.e., MB1 represents the first worker in submitting their worker for the ‘creating maps’ decision

Worker ID	Category				
eMB1	Map Only	Map Only	Map Only	Map Only	Map Only
eMB2	Map Only	Map Only	Map Only	Map Only	Map Only
eMB3	Map Only	Map Only			
eMB4	Map Only	Map Only	Map Only	Map Only	
eMB5	Map Only	Map Only			
eMB6	Map Only	Map Only	Map Only		
eMB7	Blocks	Blocks	Click and drag	Assisted Drawing	
eMB8	Click and drag	Click and drag	UI layout and extra features		
eMB9	Blocks	Grid Drawing	Assisted Drawing		
eMB10	Blocks	Isolated road properties			
eMB11	Map Only	Map Only	Map Only	Map Only	
eMB12	Traffic Simulation	Map Only	Isolated road properties	Isolated road properties	Traffic Simulation
eMB13	Nodes	Assisted Drawing	Assisted Drawing	Pencil-like (Draw)	Blocks
eMB14	Traffic Simulation	Isolated road properties			
eMB15	Click and drag	Click and drag	Click and drag	Grid Drawing	Blocks
eMB16	Nodes	Grid Drawing	Traffic Simulation	Isolated road properties	Automated using input
eMB17	Map Only	Map Only	Map Only	Map Only	Map Only
eMB18	Blocks				
eMB19	Nodes	Nodes			
eMB20	Nodes	Nodes	Grid Drawing	Nodes	
eMB21	Click and drag	Assisted Drawing	Isolated road properties	Blocks	
eMB22	Grid Drawing				
eMB23	Nodes	Blocks	Click and drag	Pencil-like (Draw)	

Figure 5.9: Diversity per worker for ‘creating maps’ decision point (UIexamples).

Worker ID	Category		
eDR1	Car logic		
eDR2	Intersection logic		
eDR3	Traffic lights rules		
eDR4	Traffic lights rules		
eDR5	Car logic		
eDR6	Traffic lights rules	Traffic lights rules	
eDR7	Car logic		
eDR8	Traffic lights rules		
eDR9	Simulator model		
eDR10	Intersection logic		
eDR11	Car logic	Simulator model	Intersection logic
eDR12	Car position centric	Car position centric	
eDR13	Car travel		
eDR14	Car lane logic		
eDR15	Car travel		

Figure 5.10: Diversity per worker for ‘moving cars’ decision point (ICexamples).

point, MB2 the second, and so on). In UI examples, worker MB7 is the one who introduces the new approaches, and that is when the set started to become more diverse. In the case of Figure 5.12, the first three workers provided solution alternatives representing unique categories, but not until worker DR9 is another new category introduced. In both charts we

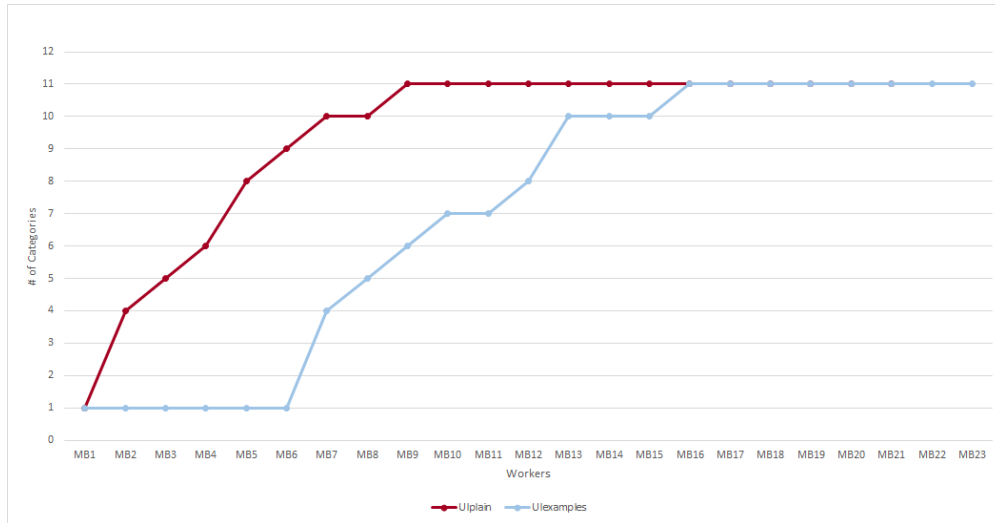


Figure 5.11: Cumulative number of unique categories created by workers for ‘creating maps’ design (UIexamples).

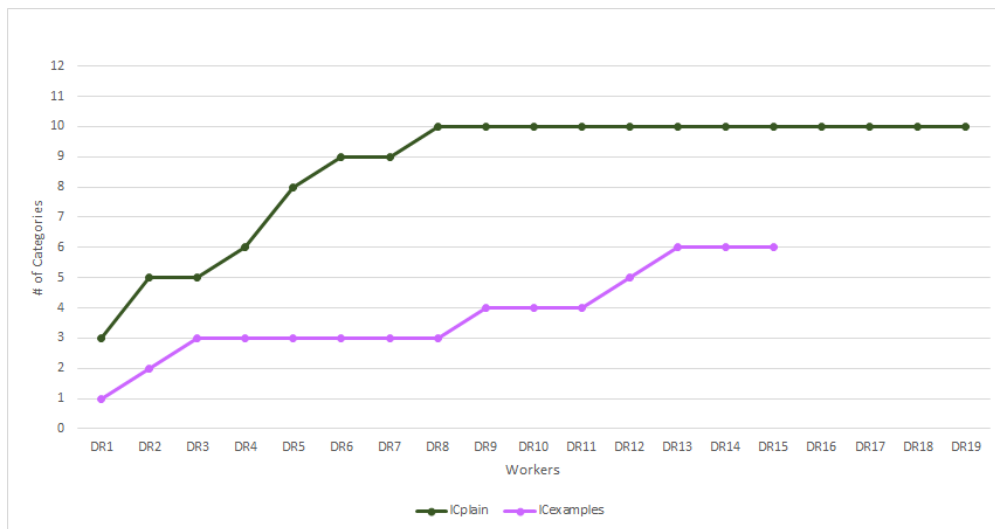


Figure 5.12: Cumulative number of unique categories created by workers for ‘moving cars’ design (ICexamples).

observe long flat lines, which indicates workers do not innovate in how to solve the problem, but create variations of the same approach. Saturation, too, took place much later. The pattern exists for the other decision points as well (see Appendix E).

## 5.4.2 Quality

Tables 5.15 (top) and 5.15 (bottom) show the quality scores for the solution alternatives collected during the experiments in which we revealed examples. In the case of UI design, there is a prominent increase in the number of low quality solutions (scores 0-1 and 1-2), resulting in a reduction of the average quality by 0.9 points. This difference is statistically significant ( $p < 0.0001$ ).

Looking at the ICexamples results we observe the opposite effect. The average quality increased by 0.3 points and the number of low quality solutions (those solutions which scored either 0-1 or 1-2) reduced from 27% of the total of solutions to 20% of the solutions. However, these results are not statistically significant ( $p < 0.07$ ).

We looked at the data from various angles trying to answer the question as to why results from the two experiments were so dissimilar. One particularly interesting answer emerged when looking at just the first ten solution alternatives produced for each decision point. In the case of UIexamples, the average quality of the first 10 solutions for each decision point is

	#solution alternatives with quality $x-y$							average quality
	0-1	1-2	2-3	3-4	4-5	5-6	6-7	
<b>dp 1.1</b>	30	7	11	9	18	4	1	2.6
<b>dp 1.2</b>	6	13	4	3	1	2	0	2.1
<b>dp 1.3</b>	7	11	8	1	5	2	0	2.3
<b>dp 1.4</b>	13	16	7	4	2	2	0	1.9
<b>total</b>	<b>56</b>	<b>47</b>	<b>30</b>	<b>17</b>	<b>26</b>	<b>10</b>	<b>1</b>	2.3

	#solution alternatives with quality $x-y$							average quality
	0-1	1-2	2-3	3-4	4-5	5-6	6-7	
<b>dp 2.1</b>	2	8	5	6	17	3	0	3.4
<b>dp 2.2</b>	1	3	3	2	8	2	0	3.4
<b>dp 2.3</b>	0	7	4	6	5	1	0	3.0
<b>dp 2.4</b>	1	1	5	10	12	3	0	3.6
<b>total</b>	<b>4</b>	<b>19</b>	<b>17</b>	<b>24</b>	<b>42</b>	<b>9</b>	<b>0</b>	3.4

Table 5.15: Quality of solution alternatives for UIexamples (top) and ICexamples (bottom).

1.0, 2.5, 2.9, and 1.8, respectively. In the case of ICexamples, the average quality per decision point was 4.3, 3.0, 2.2, and 4.1 respectively. We suspect that the first solutions in each set might therefore drive the quality of subsequent solutions, setting implicit expectations of the quality that is needed.

Figure 5.13 shows the cumulative average quality score with each new solution, for the ‘creating maps’ decision point (UIexamples) and ‘representing cars’ decision point (ICexamples). In the case of the ‘creating maps’ decision point, we can see that solution number 22 (which scored 2.75) is the one that starts raising the cumulative average, which until that moment was no greater than 1.01. Subsequent solutions started to score much higher (between 2.9 and 5.5), until solution alternative 34 scored 1.0 and after that there are 9 other solutions following the low quality. Different from the previous decision point, in the case of the ‘representing cars’ decision point, we do not see the flat tails of low quality. Until solution alternative number 15, which scored 1.25, the cumulative average quality score maintains between 4.0 and 4.5. After solution number 15, we start to observe more variability in the

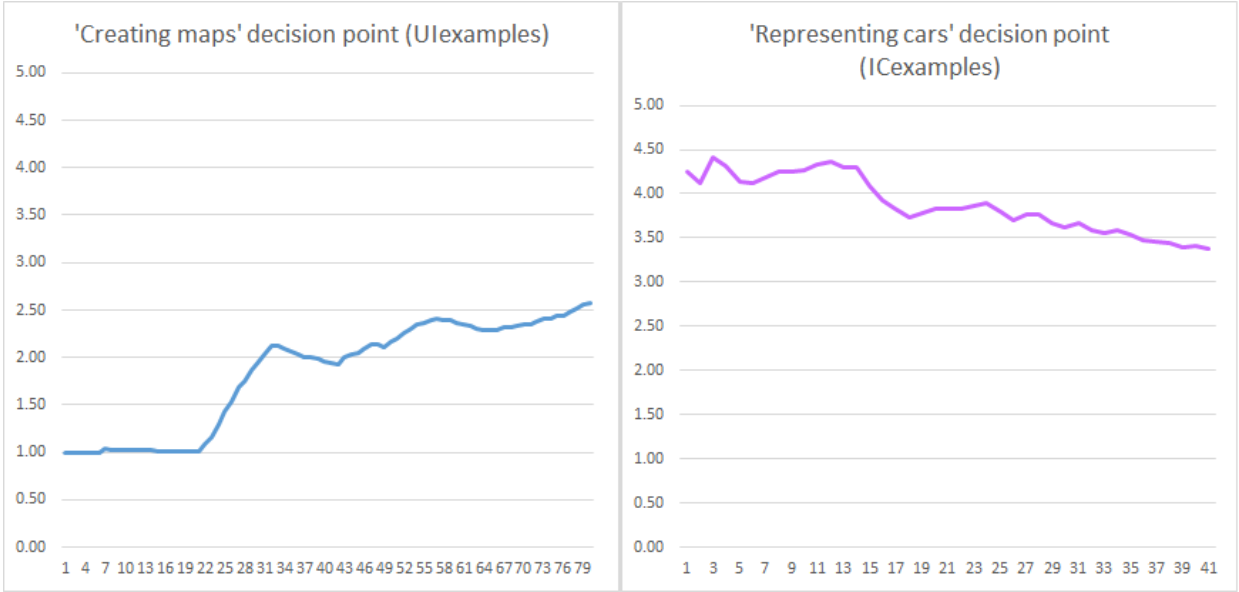


Figure 5.13: Cumulative average quality score for ‘creating maps’ decision point in the UIexamples experiment (left), and ‘moving cars’ decision point in the ICexamples experiment (right).

	#requirements met					average
	0	1	2	3	4	
<b>dp 1.1</b>	33	5	5	34	3	1.6
<b>dp 1.2</b>	0	18	7	3	1	1.6
<b>dp 1.3</b>	10	3	11	5	5	1.8
<b>dp 1.4</b>	7	24	5	8	0	1.3
<b>total</b>	<b>50</b>	<b>50</b>	<b>28</b>	<b>50</b>	<b>9</b>	<b>1.6</b>

	#requirements met					average
	0	1	2	3	4	
<b>dp 2.1</b>	12	1	6	12	10	2.2
<b>dp 2.2</b>	4	4	6	1	4	1.8
<b>dp 2.3</b>	9	2	6	4	2	1.5
<b>dp 2.4</b>	4	3	10	8	7	2.3
<b>total</b>	<b>29</b>	<b>10</b>	<b>28</b>	<b>25</b>	<b>23</b>	<b>2.0</b>

Table 5.16: Number of requirements met for UIexamples (top) and ICexamples (bottom).

quality scores. We presume that workers' expectations about the needed quality is highly influenced by the solution alternatives they have access to.

Tables 5.16 (top) and 5.16 (bottom) tell a similar story. The average number of requirements that were met decreased from 2.3 to 1.6 in the case of UI design ( $p < 0.0001$ ), and increased from 1.9 to 2.0 in the case of internal code design (however, with no statistical significance ( $p < 0.362$ )). These results are in line with the quality scores results: while for the UI design experiments, examples had a significant negative impact, we did not find significant differences between ICplain and ICexamples.

As with the first two experiments, we analyzed the combination of the quality scores and the requirements met. The shaded area in Table 5.17 presents the results for UIexamples, with 18.7% of the solution alternatives falling within the highlighted area. Compared to the UIplain experiment, where 30.4% of the solution alternatives were in this area, there is a loss of 11.7%. As we highlighted before, we see a decrease in the number of top designs for the UIexamples set of solutions.



Scores	#requirements met					total
	0	1	2	3	4	
<b>0-1</b>	41	13	2	0	0	<b>56</b>
<b>1-2</b>	8	32	6	1	0	<b>47</b>
<b>2-3</b>	1	5	15	7	2	<b>30</b>
<b>3-4</b>	0	0	3	14	0	<b>17</b>
<b>4-5</b>	0	0	1	22	3	<b>26</b>
<b>5-6</b>	0	0	1	5	4	<b>10</b>
<b>6-7</b>	0	0	0	1	0	<b>1</b>
<b>total</b>	<b>50</b>	<b>50</b>	<b>28</b>	<b>50</b>	<b>9</b>	<b>187</b>

Table 5.17: Crosstabulation of quality scores and requirements met (UIexamples).

In the case of the ICexamples experiment, 33.0% of the solution alternatives are within the highlighted area (see Table 5.18). Compared to ICplain, this represents an increase of 7.7%. However, the major increase occurs in the intersection of 4-5 (quality score) and 3 (requirements met). There is still an absence of the highest quality solution alternatives.

Scores	#requirements met					total
	0	1	2	3	4	
<b>0-1</b>	4	0	0	0	0	<b>4</b>
<b>1-2</b>	19	0	0	0	0	<b>19</b>
<b>2-3</b>	6	6	5	0	0	<b>17</b>
<b>3-4</b>	0	3	11	5	5	<b>24</b>
<b>4-5</b>	0	1	11	20	10	<b>42</b>
<b>5-6</b>	0	1	0	0	8	<b>9</b>
<b>6-7</b>	0	0	0	0	0	<b>0</b>
<b>total</b>	<b>29</b>	<b>10</b>	<b>28</b>	<b>25</b>	<b>23</b>	<b>115</b>

Table 5.18: Crosstabulation of quality scores and requirements met (ICexamples).

### 5.4.3 Difficulty

Few workers left feedback when quitting their task. While the most popular complaint among the people who quit in the previous two experiments was the lack of clearness, for both experiments in which examples were shown, the most popular reasons for abandoning their task were different. The difficulty of the task and the tool were the primary reasons provided. This would suggest that workers did not have as much difficulty in understanding the task,

helped by their coworkers' work.

Averages calculated for perceived difficulty are slightly lower than the corresponding experiments not revealing examples. Table 5.19 shows the comparison between the results of the questionnaires of the four experiments. A statistically significant difference between UIplain and UIexamples exists, with workers in the UIexamples experiment perceiving the task as less difficult than workers in the UIplain experiment. We did not find a significant difference between ICplain and ICexamples.

We also examined whether one or a few of the decision points stood out as particularly difficult as compared to the other ones. It turns out that, across the different experiments, different decision points were deemed most difficult by the workers. When we examined this with a one-way analysis of variance, we found no statistical significance between the average score of difficulty level of the different decision points.

	<b>UIplain</b>	<b>UIexamples</b>	<b>p-value</b>	<b>ICplain</b>	<b>ICexamples</b>	<b>p-value</b>
<b>ability to complete the entire task</b>	5.17	4.47	0.002	4.80	4.49	0.23
<b>difficulty level of the decision point</b>	4.72	4.47	0.04	4.65	4.35	0.17
<b>adequacy of support by the tool</b>	4.94	4.19	0.05	4.32	4.50	0.54

Table 5.19: Exit questionnaire results across the four experiments and comparisons.

## 5.5 Borrowing

### 5.5.1 Use of ‘Duplicate’ and ‘Copy’ Features

During the UIexamples and ICexamples experiments, participants had the opportunity to copy any part of any solution and use what was copied in their work, either as a basis to start from or integrated into what they already were working on. Even though we are aware that it is a conservative analysis (as workers can also copy or be inspired by existing work just by looking at it, and then incorporating ideas in on their solution alternative), we were

Source				Destination				Delta score	Delta req met
Worker ID	Solution #	Score	Req met	Worker ID	Solution #	Score	Req met		
eCA1	2/2	4.0	3	eCA3	1/1	4.5	4	+0.5	+1
eCA2	1/4	5.0	4	eCA16	1/2	5.3	4	+0.3	0
eCA9	3/5	4.0	4	eCA10	1/1	4.3	3	+0.3	-1
eCA9	5/5	4.6	4					-0.3	-1
eDR3	1/1	2.6	1	eDR9	1/1	4.0	3	+2.7	+2
eDR7	1/1	4.8	4				3	-0.8	-1
eDR7	1/1	4.8	4	eDR8	1/1	1.3	0	-3.5	-4
eDR11	2/3	4.2	2	eDR12	1/2	5.0	4	+0.8	+2
eRS1	1/1	3.3	2	eRS2	1/2	1.6	0	-1.7	-2
eRS5	2/2	2.1	0	eRS11	1/1	1.5	0	-0.6	0
eTL2	1/1	4.0	2	eTL4	1/5	4.3	4	+0.3	+2
eTL2	1/1	4.0	2	eTL4	2/5	3.8	4	-0.2	+2
eTL7	1/1	4.7	3	eTL9	1/1	3.1	2	-1.6	-1
eTL10	2/2	3.4	2	eTL11	1/3	4.1	3	+0.7	+1
<b>Average</b>		<b>3.9</b>	<b>2.6</b>	<b>Average</b>		<b>3.6</b>	<b>2.6</b>	<b>-0.3</b>	<b>0</b>

Table 5.20: Use of copy and duplicate feature (ICexamples).

interested in how many workers used the copy (or duplicate) feature and, when they did, if those workers improved over the solution alternative that was copied.

Table 5.20 shows the results for the ICexamples experiment. Out of 62 workers who had access to previous workers' work (as the first four workers are assigned as the first worker in each decision point and, thus, did not have any previous work to look at), only 12 workers (19%) used the *duplicate* feature to copy an entire example, and none used the *copy* feature. Some workers copied multiple solution alternatives (eCA10 and eDR9), and some solution alternatives were copied multiple times (eDR7 and eTL2).

With a few exceptions (eDR3, eRS1, eRS5 and eTL10), most of the sources scored 4.0 or higher (average 3.9) and met three or more requirements. This suggests that workers were able to identify good solutions as the basis from which to work. Only 5 workers, however, then went on to improve the source solution (see Table 5.20). eCA10, for instance, copied from two solution alternatives, the average of which equals the score of the new solution. However, the new solution met one requirement fewer than the source solution alternatives. eDR9 also copied from two solutions, but in this case from two different workers. With

respect to the first example, the quality drastically improved. With respect to the second one, however, the quality was reduced.

With the exception of workers eCA16 and eDR9, workers only borrowed from one of the two immediately previous workers and not further from any before those. These results suggest that workers did not browse across the solution alternatives. We can see in the table, where the number in each of the workers ID represents the order in which workers submit their work, how workers only looked at the work of at best two workers before them (e.g., eCA3 copied from eCA1, only two workers away).

Another interesting observation is that all workers who actively copied used the ‘duplicate’ function for the first solution alternative they created (with the exception of eTL4, who, in addition to the first alternative, duplicated a sketch for their second solution alternative). This could mean that the goal behind using the ‘duplicate’ function is to start the exploration of the design space, having as a baseline another solution alternative.

Performing the same analysis for the UI design experiment (see Table 5.21), the results tell a slightly different story. Only 10 workers out of 76 (13%) used the ‘copy’ or ‘duplicate’ functions, with one worker (eVT21) using ‘copy’ and nine using ‘duplicate’. Workers copied from

Source				Destination				Delta score	Delta req met
Worker ID	Solution #	Score	Req met	Worker ID	Solution #	Score	Req met		
eMB1	2/5	1.0	0	eMB2	5/5	1.0	0	0	0
eMB3	2/2	1.0	0	eMB4	1/4	1.0	0	0	0
eMB6	1/3	1.0	0	eMB12	3/5	1.2	0	+0.2	0
eMB9	2/3	4.6	3	eMB10	1/2	4.8	3	+0.2	0
eMB18	1/1	2.5	3	eMB21	1/4	4.1	3	+1.6	0
eSL13	1/5	1.2	1	eSL15	2/3	1.9	1	+0.7	0
eSL13	4/5	4.7	2	eSL15	3/3	3.4	3	-2.8	-1
eSL13	4/5	4.7	2	eSL15	3/3	3.4	3	-1.3	+1
eVT5	1/1	1.7	2	eVT7	1/1	1.4	1	-0.3	-1
eVT6	1/2	3.7	3	eVT8	1/1	1.9	1	-0.4	-2
eVT6	2/2	2.3	1	eVT10	1/1	2.0	1	-1.7	0
eVT13	1/3	1.0	2	eVT21 (c)	1/1	1.8	1	+0.8	-1
Average		2.5	1.5	Average		2.2	1.3	-0.3	-0.2

Table 5.21: Use of copy and duplicate feature (UIexamples).

solution alternatives very diverse in quality, ranging from very bad to very good solutions. The average quality of the sources is on the low side, and the same is true for the average number of requirements met. A possible explanation for this could be that, as we showed in the previous section, all of the initial workers scored between 0 and 2, therefore negatively influencing future workers. Only four workers improved the source solution, scoring better than the original, and just one new solution alternative met one extra requirement than the original. As in ICexamples, workers did not borrow from workers further than two previous workers, which suggest a lack of exploration of the available solution alternatives (MB12, VT10, and VT21 are exceptions, who copied from 6, 4, and 8 workers before them). The other workers copied from the previous two workers they had access to.

Results may be suggesting that we should probably experiment with different approaches. Clearly, when borrowing from previous workers' work, the overall quality tends to decrease, especially when copying from low quality solution alternatives. There are many questions that arise from this analysis. Would quality increase by giving workers the best examples? Would workers be discourage by seeing top designs? In order to start to answer those and many other questions, more experimentation needs to be done.

### **5.5.2 Solution Categories Distances**

As not every worker copied previous solution alternatives, our second analysis looked at the distance between solutions of the same category. Previously, in our diversity analysis, we plotted the categories identified for the solution alternatives created by each worker (see Figure 5.4 as an example of this analysis for UIplain experiment). Here, we take this analysis further, focusing on the distance between repeated categories.

First, we did a visual examination to observe the spread of the categories along the experiment. Figure 5.14 shows this analysis applied to the 'creating maps' decision point, for the

Worker ID	Categories for map creation decision point										
	Assisted Drawing	Automated using input	Blocks	Grid Drawing	Click and drag	Map Only	Isolated road properties	Nodes	Pencil-like (Draw)	Traffic Simulation	UI layout and extra features
eMB1						●					
eMB2						●					
eMB3						●					
eMB4						●					
eMB5						●					
eMB6						●					
eMB7	●		●		●						
eMB8					●						●
eMB9	●		●	●							
eMB10			●				●				
eMB11						●					
eMB12						●	●			●	
eMB13	●		●					●	●		
eMB14							●			●	
eMB15			●	●	●						
eMB16		●		●			●	●		●	
eMB17						●					
eMB18			●								
eMB19								●			
eMB20				●				●			
eMB21	●		●		●		●				
eMB22				●							
eMB23			●		●			●	●		

Figure 5.14: Occurrences of each category in a chronological order (‘creating maps’ decision point, UIexamples experiment).

UIexamples experiment. Each row represents a worker (in chronological order) and each column a category identified. Each of the green dots symbolizes a solution alternative under the category at the top of the column. We can see how this representation helps to identify some trends. First, as already observed, we can identify some categories being repeated worker after worker. A notorious example is the ‘map only’ category: from worker eMB1 to worker eMB6, the same conceptual approach was repeated over and over.

Second, we calculated the distance between categories for each of the decision points, for each of the four experiments. Specifically, we counted how many workers separate one occurrence of a category from a next instance. For example, if the category ‘assisted drawing’ was identified in a solution submitted by worker eMB7 (7<sup>th</sup> worker) and next in a solution submitted by worker eMB9 (9<sup>th</sup> worker) of the map building decision point, the distance between these solutions is two. In other words, it took two submissions for this category to be identified again in another solution alternative. We group under ‘new’ all the sketches

that represent the first occurrence of a category for that particular group.

Table 5.22 shows for each decision point, how many solutions with  $x$  distance were identified, across the four experiments. In both kinds of tasks, experiments revealing examples (UIexamples and ICexamples) present an increase in the solutions with distance one and two, and a decrease in solutions with distance higher than six as well as in novel approaches. That is to say, solution alternatives with the same category in the experiments revealing examples are closer to each other, while in the experiments without there is a higher distance between solution alternatives in the same category. This could serve as an indicator of borrowing: workers get influenced by solution alternatives created by their coworkers. At the same time, we observe again that workers do not browse in the set looking for different alternatives, but appear to just look at the first designs available in the set.

	#solutions with $x$ distance per experiment															
	UI No examples								UI With Examples							
	1	2	3	4	5	6+	new	total #solutions	1	2	3	4	5	6+	new	total #solutions
<b>dp 1.1</b>	12	5	3	1	4	19	18	<b>62</b>	24	12	7	3	10	7	17	<b>80</b>
<b>dp 1.2</b>	3	5	0	5	1	13	19	<b>46</b>	6	6	2	1	1	3	10	<b>29</b>
<b>dp 1.3</b>	3	3	0	7	2	4	16	<b>35</b>	5	10	6	4	0	2	7	<b>34</b>
<b>dp 1.4</b>	2	0	6	0	2	12	16	<b>38</b>	12	9	2	2	2	7	10	<b>44</b>
<b>Total (%)</b>	20 (11%)	12 (7%)	10 (5%)	13 (7%)	7 (5%)	50 (27%)	69 (38%)	<b>181</b>	47 (25%)	37 (20%)	17 (9%)	10 (5%)	13 (7%)	19 (10%)	44 (24%)	<b>187</b>

	#solutions with $x$ distance per experiment															
	IC No examples								IC With Examples							
	1	2	3	4	5	6+	new	total #solutions	1	2	3	4	5	6+	new	total #solutions
<b>dp 2.1</b>	4	4	5	2	1	7	13	<b>36</b>	2	4	6	2	1	8	18	<b>41</b>
<b>dp 2.2</b>	4	2	1	0	2	6	18	<b>33</b>	9	2	0	1	0	2	9	<b>23</b>
<b>dp 2.3</b>	1	6	2	2	0	13	18	<b>42</b>	9	3	3	1	2	0	14	<b>32</b>
<b>dp 2.4</b>	11	5	6	6	2	4	13	<b>47</b>	2	6	0	2	0	1	8	<b>19</b>
<b>Total (%)</b>	20 (13%)	17 (11%)	14 (9%)	10 (6%)	5 (3%)	30 (19%)	62 (39%)	<b>158</b>	22 (19%)	15 (13%)	9 (8%)	6 (5%)	3 (3%)	11 (10%)	49 (43%)	<b>115</b>

Table 5.22: Number of sketches with  $x$  distance for each of the four experiments.

## 5.6 Additional Analyses

### 5.6.1 Finding Subgroups within the Solution Alternatives

Earlier in this chapter we presented the analysis of the highest quality solution alternatives for each of the four experiments. An important question concerns the diversity of the “best” solution alternatives (those with high overall quality and high number of requirements met). This set, after all, will form the basis for future design work (including, for morphological chart, choosing overall designs by combining choices from the individual decision points). If all of the best solutions are not diverse, this would represent a problem. In Tables 5.8, 5.12, 5.17, and 5.18 we highlighted what we consider the best subsets for each experiment. Here, we analyze the results for diversity trying to find out if which is the ideal subset of solution alternatives to work with.

Table 5.23 shows the resulting average quality (Q), the resulting average number of requirements met (Req), and number of categories (Cat) of the ‘top designs’ subset. An expected result is a highly increase in both average quality and average number of requirements met. Diversity, however, is drastically negatively affected. By keeping only ‘top designs’, we loose about 50% of the categories collected across the four experiments, which is a percentage that substantially hurts the diversity of the set.

However, a careful analysis of the actual categories dropped tells us something more nuanced.

	All Solutions			Top Solutions		
	Q	Req	Cat	Q	Req	Cat
<b>UIplain</b>	3.2	2.3	45	4.8 (+1.6)	3.4 (+1.1)	23 (-48.9%)
<b>UIexamples</b>	2.3	1.6	35	4.7 (+2.4)	3.2 (+1.6)	13 (-62.9%)
<b>ICplain</b>	3.1	1.9	42	4.6 (+1.5)	3.5 (+1.6)	17 (-59.5%)
<b>ICexamples</b>	3.4	2.0	38	4.6 (+1.2)	3.5 (+1.5)	23 (-39.5%)
<b>Average delta</b>	-	-	-	<b>+1.7</b>	<b>+1.5</b>	<b>-21 (-52.7%)</b>

Table 5.23: Resulting average quality, average number of requirements met, and total diversity of the subset of ‘top designs’ (all experiments).



Categories	UIplain		UIexamples	
	Average Quality	Average Requirements met	Average Quality	Average Requirements met
Assisted Drawing	4.4	3.7	4.0	3.0
Automated using input	3.6	4.0	2.5	2.0
Blocks	4.9	3.3	4.0	2.9
Build as you go	3.8	3.0	-	-
Click and drag	4.1	3.0	4.7	3.0
GPS	2.0	1.0	-	-
Grid Drawing	-	-	3.7	2.8
Isolated road properties	-	-	1.8	1.2
Map only	1.0	0.1	1.0	0.1
Nodes	4.7	3.3	3.7	3.1
Pencil-like (Draw)	4.0	3.0	4.0	3.0
Traffic Simulation	1.0	0.0	2.0	0.8
UI layout and extra features	3.0	2.4	3.6	3.0

Table 5.24: Average quality and average number of requirements met for categories identified for ‘creating maps’ decision point.

When examining the average quality and the average number of requirements met, we can observe that some of the categories do not represent proper solutions for the design problem. Table 5.24 shows an example for the ‘creating maps’ decision point, for the UI experiments. We can see in the table that, for some of the categories, average quality and average number of requirements met are not greater than 1.0. Categories such as “map only” or “traffic light simulation” are examples of categories that cluster solution alternatives the approach of which is not a viable solution for the problem. Figure 5.15 illustrates an example of a such solution alternative within the “map only” group. The proposed solution alternative does not represent a viable solution for the problem of ‘designing an interface mechanism through which users build maps with roads and intersections’.

A somewhat similar story emerges when focusing on the number of requirements met. Table 5.25 shows the average quality (Q), the average number of requirements met (Req), and the number of categories representing different kinds of overall solution approaches (Cat), per experiment – when filtering by number of requirements met.

The first filter leaves out solution alternatives not meeting any requirement. If we look at the reduction in categories resulting from taking out solution alternatives not meeting any

name: avenues and crosses

explanation: Simple arrangement of exactly perpendicular roads (rectangles), having 8 intersections (circles).

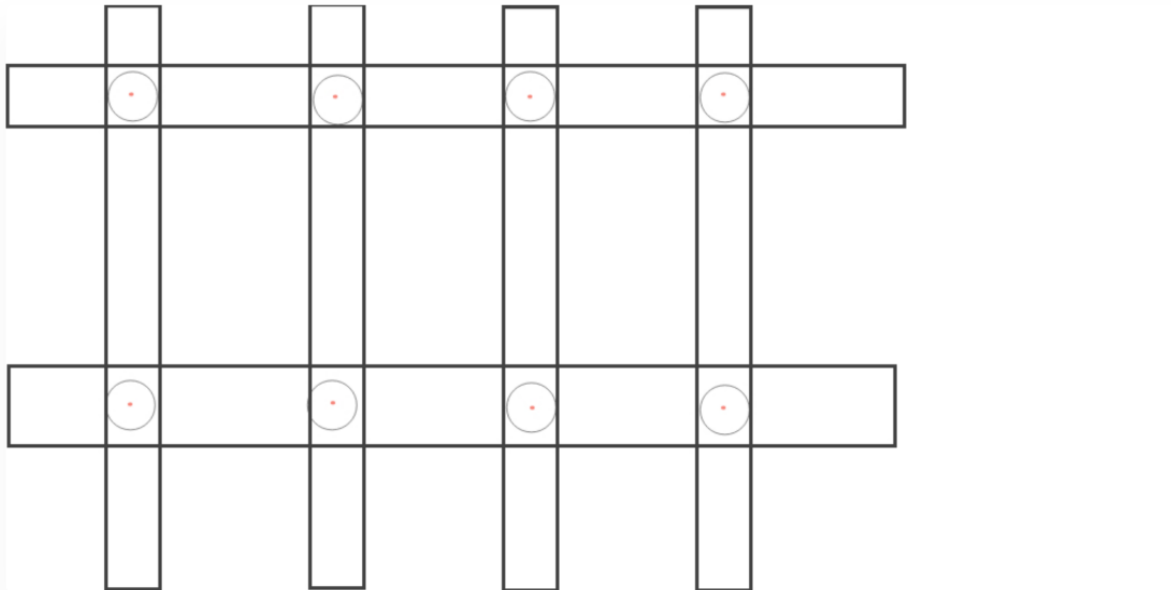


Figure 5.15: Example of a solution within the “map only” category (‘creating maps’ decision point, UIplain).

	All Solutions			1 to 4 req met			2 to 4 req met			3 to 4 req met		
	Q	Req	Cat	Q	Req	Cat	Q	Req	Cat	Q	Req	Cat
<b>UIplain</b>	3.2	2.3	45	3.4	2.5	43	3.9	3.0	38	4.1	3.3	30
<b>UIexamples</b>	2.3	1.6	35	2.7	2.1	35	3.5	2.8	29	4.0	3.2	20
<b>ICplain</b>	3.1	1.9	42	3.8	2.6	35	4.0	3.0	34	4.2	3.4	26
<b>ICexamples</b>	3.4	2.0	38	4.0	2.7	34	4.1	2.9	30	4.3	3.5	24
<b>Average delta</b>	-	-	-	<b>+0.5</b>	<b>+0.5</b>	<b>-3.3</b>	<b>+0.4</b>	<b>+0.5</b>	<b>-4.0</b>	<b>+0.3</b>	<b>+0.4</b>	<b>-7.8</b>

Table 5.25: Resulting average quality, average number of requirements met, and total diversity when filtering out solutions with  $x$  requirements met (all experiments).

requirement in the internal code experiments, we see that categories such as “Graphical maps” for the ‘representing cars’ decision point (for which workers drew roads and intersections without any detail about the actual implementation of cars), “traffic light rules” for the ‘moving cars’ decision point (for which workers described the traffic light cycle, without explaining how it affects the ‘moving cars’), and “traffic rules” for the ‘changing the colors of traffic lights’ decision point (for which workers described general rules about traffic, and not any detail about the implementation of an algorithm for a traffic simulator) are no longer included. A similar phenomenon occurred in the case of the two user interface experiments:

“Map only” (for which workers drew final maps, not the mechanism to create them) for the ‘creating maps’ decision point and “traffic lights status” (for which workers indicated the current status of certain traffic lights in the map, instead of the traffic status) for the traffic visualization decision point are dropped. All of these categories represent groups of solution alternatives which did not address the design problem. While diversity is reduced, only truly non-viable solutions are eliminated.

Now, consider the right side of Table 5.25. By keeping all of the solution alternatives meeting 3 and 4 requirements (unlike the ‘top designs’ analysis, conserving as well solutions which a score lower than four), quality generously improves. More importantly, this filter preserves valuable solution alternatives, which are novel and unique conceptual approaches that contribute to a great diversity of the set. An example of a category that was kept by including medium quality solution alternatives is ‘automated using input’ for the ‘creating maps’ decision point (UI design), where the mechanism to create the maps is through textual specifications, instead of drawing-like interactions (see Figure 5.16). An average of at

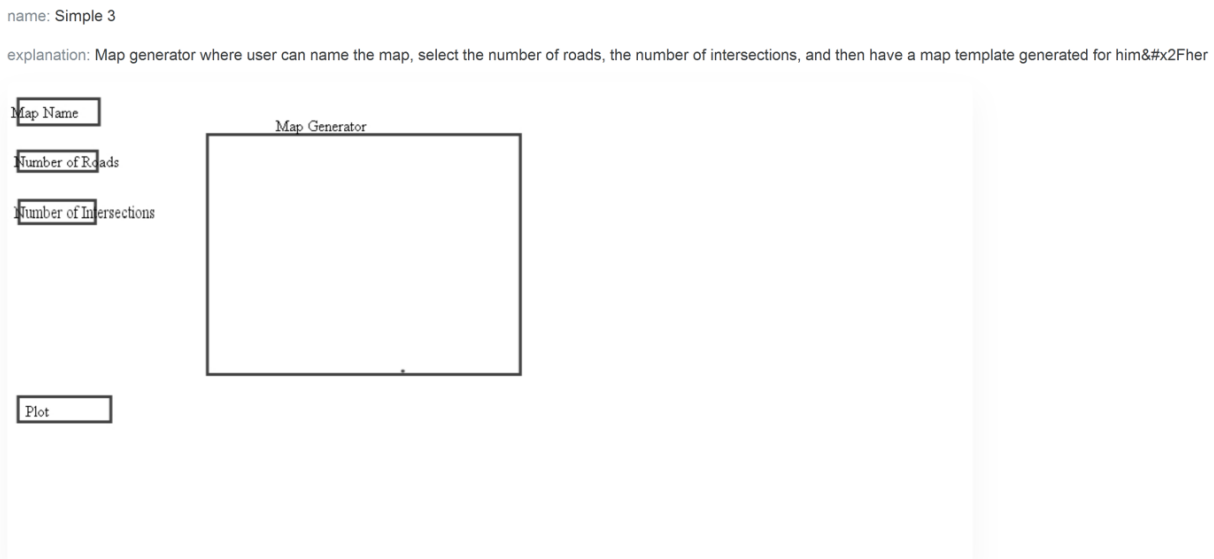


Figure 5.16: Example of a solution within the “automated using input” category (‘creating maps’ decision point, UIplain).

least five categories remain for each decision point in each experiment. This is a non trivial amount of diversity, and, still, is more than individual designers or a team could generate easily.

## 5.6.2 Finding Subgroups within the Crowd

### Worker Experience

We first examined the differences between demographic groups. Table 5.26 shows the average quality (Q), the average number of requirements met (Req), and the total number of categories (Cat) provided by each of the demographic groups in each of the experiments. Additionally, the last column for each of the four experiments (Cat\*) indicates the number of meaningful categories identified by each of the demographic groups.

Across the four experiments, there is one demographic group that stands out. In general, professional UI/UX designers performed better than the other demographic groups. At the same time, it is unfortunately the least represented group: only five professional designers participated across the first three experiments and none in the last experiment (see Table 5.2). Despite their excellent performance, given their low representation we cannot draw strong conclusions about this group and their work, though it certainly appears that if more participated, results would improve. Because few participated, however, we exclude them from what follows below.

	Uplain				Uexamples				ICplain				ICexamples			
	Q	Req	Cat	Cat*	Q	Req	Cat	Cat*	Q	Req	Cat	Cat*	Q	Req	Cat	Cat*
<b>Undergraduate student</b>	3.0	2.3	14	7	2.2	1.3	17	6	2.7	1.5	16	8	3.2	2.0	21	7
<b>Graduate Student</b>	2.5	1.7	11	7	2.5	1.6	9	4	2.5	1.4	7	2	3.3	2.0	3	1
<b>Professional Developer</b>	3.5	2.5	23	16	2.8	2.0	17	11	2.9	1.5	23	9	3.5	2.1	24	14
<b>Professional UI/UX Designer</b>	3.9	2.5	6	4	3.2	2.6	7	4	5.0	4.0	1	1	-	-	-	-
<b>Hobbyist</b>	3.1	2.2	25	18	2.0	1.3	28	10	3.4	2.3	26	15	3.3	2.0	21	10
<b>Other</b>	3.2	2.4	18	9	3.0	2.0	11	5	3.0	1.8	14	8	3.3	1.8	7	2

Table 5.26: Average quality, average number of requirements met, and total diversity per demographic group.

We did not find a significant difference between groups in the UIplain experiment, both regarding quality scores ( $p < 0.054$ ) or requirements met ( $p < 0.131$ ). The distribution of quality and requirements met is the same across demographic groups. In the case of ICexamples, there is some evidence of significant differences between groups regarding quality ( $p < 0.04$ ), but no differences as to requirements met ( $p < 0.082$ ). The groups which perform the worst in this experiment are undergraduate students and hobbyists.

In terms of diversity, hobbyists had an important role, being the group which identified the most categories. This may well be because it is the most represented group. In terms of meaningful categories, professional developers (even though being least represented) are very close to hobbyists in terms of contributions. We can see this, in particular, for the UIexamples experiment. Even though hobbyists contributed 28 categories, only 10 (35.7%) were meaningful categories, while in the case of software developers 11 of the 17 provided categories (64.7%) represented meaningful categories.

Demographic groups had a more even performance in the internal code experiments. In the case of ICplain, there is some evidence of differences between groups, both for quality scores ( $p < 0.053$ ) and requirements met ( $p < 0.051$ ). The different demographic groups performed practically even in the ICexamples experiment. We did not find any differences in quality score ( $p < 0.768$ ) or requirements met ( $p < 0.983$ ).

In terms of diversity, hobbyists are again the group contributing the most diversity for UIplain. Interestingly, software developers were not as effective this time. Only 39.1% of the categories provided by this group were truly valid categories. ICexamples had greater participation of software developers over other demographic groups, which is also reflected in their contribution to diversity.

Overall, is surprising the fact that professionals do not stand out as much as are might be expected as compared to other groups. This may be because they did not perform at the top

of their ability, perhaps being merely curious about the experiment. At the same time, the results show that not only professional designers and developers can produce good designs.

A different analysis confirms this statement. Earlier in this chapter we showed the percentage of high quality solution alternatives for each of the four experiments. We now examine the composition of the group which generated those designs. Table 5.27 shows the distribution across demographics within this group of workers. At least one worker in each of the demographic groups generated a top design. In fact, software developers were not the group that contributed the most such designs (except for ICexamples). Hobbyists had a more important role in generating top rated solution alternatives.

	<b>UIplain</b>	<b>UIexamples</b>	<b>ICplain</b>	<b>ICexamples</b>
<b>Undergraduate student</b>	3 (8.6%)	2 (11.1%)	5 (18.5%)	5 (19.2%)
<b>Graduate Student</b>	4 (11.4%)	2 (11.1%)	1 (3.7%)	1 (3.8%)
<b>Professional Developer</b>	11 (31.4%)	3 (16.7%)	7 (25.9%)	11 (42.3%)
<b>Professional UI/UX Designer</b>	2 (5.7%)	1 (5.6%)	1 (3.7 %)	-
<b>Hobbyist</b>	9 (25.7%)	5 (27.8%)	10 (37.0%)	7 (26.9%)
<b>Other</b>	6 (17.1%)	5 (27.8%)	3 (11.1%)	2 (7.7%)
<b>Total</b>	<b>35</b>	<b>18</b>	<b>27</b>	<b>26</b>

Table 5.27: Demographics of the group of workers who generated the best designs.

## Qualification Test Score

Our second analysis examines the impact of the score that each of the workers obtained in the qualification test. Workers needed at least three out of five correct answers to participate in the experiment. As only four workers in the UIplain experiment and two workers in the UIexamples experiment obtained the maximum score, for our analysis we divided workers into two groups: (1) those workers who scored three out of five, and (2) workers with four out of five correct answers together with workers with a perfect score. Table 5.28 shows, per experiment, workers in each group (#W indicates the number of workers who obtained a score of  $x$  on the test), the solution alternatives they produced (Q is the average quality for workers on that group, and Req the average number of requirements met for the same

	UIplain					UIexamples					ICplain					ICexamples				
	#W	Q	Req	Cat	Cat*	#W	Q	Req	Cat	Cat*	#W	Q	Req	Cat	Cat*	#W	Q	Req	Cat	Cat*
<b>3/5 correct</b>	48	2.9	2.2	38	24	65	2.2	1.5	32	19	27	2.8	1.5	29	9	17	3.2	1.8	17	9
<b>4/5 and 5/5 correct</b>	30	3.5	2.4	30	21	17	2.6	1.9	21	10	49	3.2	2.0	38	25	49	3.4	2.1	34	20
<b>Total</b>	<b>78</b>	<b>3.2</b>	<b>2.3</b>	<b>45</b>	<b>30</b>	<b>80</b>	<b>2.3</b>	<b>1.6</b>	<b>35</b>	<b>20</b>	<b>76</b>	<b>3.1</b>	<b>1.9</b>	<b>42</b>	<b>26</b>	<b>66</b>	<b>3.4</b>	<b>2.0</b>	<b>38</b>	<b>24</b>

Table 5.28: Performance of workers with  $x$  correct answers out of five during the qualification test per experiment.

group), and how each group contributed to diversity (Cat being the total number of categories provided by the group, and Cat\* the number of meaningful categories).

Overall, results suggest that workers who scored higher in the qualification test performed better. However, the significance of the results vary from experiment to experiment. In the case of UIplain, we found strong evidence of quality being significantly better on solutions generated by people in the second group ( $p < 0.0035$ ) and weaker evidence of requirements met also being higher in the case of workers who scored higher in the qualification test ( $p < 0.079$ ). In the case of UIexamples, the strongest evidence is regarding requirements met ( $p < 0.023$ ), and weaker evidence of quality being better in the second group ( $p < 0.062$ ). In the case of ICplain, both average quality and average number of requirements met was significant better in the set of solution alternatives created by the second group (quality  $p < 0.0285$ ; requirements met  $p < 0.018$ ). We did not find any significant difference between the two groups in the ICexamples experiment ( $p < 0.258$  for quality, and  $p < 0.237$  for requirements met).

The great number of workers who scored three out five over workers who scored four or five correct answers could possibly explain the reasons of why results on UIexamples were so much worse. However, we do not have enough evidence to support such a claim. Nevertheless, it might be worthwhile exploring balancing strong workers (i.e., high score on qualification test) across different decision points.

Even though it seems advantageous in terms of quality to raise the qualification threshold for workers who can participate in further experiments, we need to analyze how diversity would

then be affected. We can see in Table 5.28 how workers with a lower qualification test score across the four experiments, and in particular in the UIexamples experiment, contributed with meaningful unique categories. For instance, if we did not have people who scored three out of five correct questions, we would have lost several novel categories, which were not provided by the second group of workers. Some examples are “Automated using input” for the ‘creating maps’ decision point (in which are indicated the map configurations and the application draws the desired map; an example can be seen in Figure 5.16), or “Bars indicating queues” for ‘visualizing the state of the simulation’ decision point (which indicates traffic density through the length of bars in intersections) for the UI design, and “Traffic Lights change based on Traffic density” for the ‘changing the colors of traffic lights’ decision point.

Results suggest that there is a more significant difference between groups based on the qualification test score than based on demographics. Our qualification test works better than self declared work experience as an indicator for valuable contributors. However, at the same time and as we highlighted before, it is important for diversity to involve, to a greater or lesser extent, multiple workers with different skills.



# 6

## Discussion

Our study demonstrates the potential of microtask crowdsourcing as applied to software design. Across the four experiments, we show that it is feasible for a crowd of workers from Amazon Mechanical Turk to generate a broad range of solution alternatives for small, partial design problems. Our results represent merely a first step toward the broader research agenda we are pursuing. Ultimately, we wish to understand whether microtask crowdsourcing, through the use of morphological charts, may be a viable approach for software design.

**It is important for diversity to involve multiple workers; individual workers did not create diverse solution alternatives.** Besides some rare exceptions, individual workers did not create diverse solution alternatives. Instead, they worked on small variations of the same conceptual approach. In contrast, when collecting solution alternatives from multiple workers, many different conceptual approaches were produced. We conclude that it is important for diversity, and therefore the exploration of the design space, to involve multiple workers in the creation of solution alternatives. Corroborating this further is the fact that workers created solution alternatives that were not previously considered in previous designs resulting from using the same design prompt in different settings (see [48] and [66]).

**The majority of the workers provided only one or two solutions; just a few workers provided three or more solutions.** Across the four experiments, not many workers submitted more than two solution alternatives. In fact, the majority of the workers chose to provide only one. In the experiments where workers had to provide solution alternatives for an internal code design task, particularly, only 23% of the workers provided three or more solution alternatives. Some workers mentioned in their feedback that they did not understand why we are interested in alternatives, since one good design was more than sufficient. This phenomena is known as design fixation. Design fixation refers to a blind, and sometimes counterproductive, adherence to a limited set of ideas in the design process [38]. The characterization of conceptual design as a discovery process means that the designer should ‘visit’ many points in both the concept space and the configuration space in order to reveal more about the problem and potential solutions, thus discovering new aspects of the problem. Jansson and Smith [38] in 1991 report a series of experiments where they demonstrate the existence of design fixation in engineering design. We can see that software design is not exempt of design fixation. Even when workers had monetary incentives to provide more than one solution, the majority of the workers decided to provide no more than one solution alternative. Therefore, we reassert the idea that is necessary to involve multiple workers to have a wide exploration of the design space.

**When workers are exposed to previous workers’ work, the diversity of the overall set decreases.** In both kinds of tasks, UI and IC design, the exposure to previous workers’ work decreased the diversity of the set. When examples are available, we observed that workers tend to repeat a limited number of conceptual approaches and, in most of the cases, do not explore new ways to solve the design problem. In fact, most of the workers who participated in UIexamples and ICexamples did not dive into the set of available solution alternatives, but just took the solution alternative created by the last or second to last worker and updated some. This is probably related to microtasking as a work model. Workers do not want to spend too much time [25]. Even searching for the best previous examples appear

cut short by just picking the first one that looks “good enough” to use. Perhaps participants thought other participants’ work represented the quality expected, so they might have felt social pressure to conform to these designs or have felt relief in the task being something they can do. Another cause could again be design fixation: once an idea (from an example in this case) is in the worker’s head, it is difficult to step outside of it and create something entirely new. This last phenomenon has been extensively studied in other design fields. As an example, Kohn and Smith [43] performed several studies comparing the diversity generated by groups whether exposed or not to other participants’ ideas. Similar to what we observed in our experiments, they conclude that people conformed to ideas to which they were exposed, the rate of conformity increased as the number of ideas exposed increased, and participants exposed to other participants’ ideas were more likely to conform than participants who did not see others’ ideas. To test these explanations, more experimentation is necessary, with perhaps a particular focus on understanding workers’ motivations behind selecting certain approaches to solve the design task.

**The average quality in experiments revealing examples was equal to or worse than the equivalent experiment not revealing examples.** Contrary to what we expected, quality did not improve when workers were exposed to previous workers’ work. This is probably related, again, to the microtasking model. Usually, workers in Amazon Mechanical Turk tend to complete as many tasks as they can in the shortest possible time. When workers are exposed to examples, it appears they adjust their work quality to what they see, assuming that what is expected from them is what we are showing. In contrast, when working on their own, they seem to push themselves more to deliver the work quality they think might be necessary to get paid.

**Solution alternatives range all over the quality spectrum.** In our four experiments, the quality of the solution alternatives varies considerably. Despite this variability, we are encouraged by the results. A relatively high percentage of the solution alternatives fall within

the group that has both a high quality score and meets three or four requirements. There was certainly still a lot of ‘wasted effort’, which means that we have to still explore different mechanisms to improve the overall quality of the set and reduce the amount of solutions with the lowest score. Some studies show the relationship between disclosing examples and overall quality in the competition model. Boudreau and Lakhani [8] found, in a series of experiment with Topcoder workers, that intermediate disclosure has the advantage of efficiently steering development towards improving existing solution approaches. LaToza et al. [48] showed that participants improved their designs after being exposed to other participants’ complete designs. Despite these findings being the result of a different crowdsourcing model (competition), we think that lessons learned might transfer.

**Not only professional designers and developers can produce good designs.** The crowd that participated in our experiments was very diverse, ranging from undergraduate students, to hobbyists, to experienced professional developers. Our results show that all of them contributed with high quality designs and novel solution alternatives. In most of the cases, there was not a significant difference between professionals and non-professionals regarding the quality of the solution alternatives they produced. Contrary to what prior research highlights about software development and expertise (e.g., [59, 50, 68]), we did not find that experienced professional software developers stood out among their coworkers. Perhaps, high expertise does not correlate with work quality in a microtask setting.

**The task is seen as difficult by many workers.** Feedback from workers reveals that the task is perceived as difficult. Even though workers who had examples stated the task was less difficult than workers who did not have examples, they still stated the task to be on the difficult side. Our task was unusual for the type of tasks workers usually find on Amazon Mechanical Turk. HITs are typically simple, asking for mechanic and repetitive tasks. Our task is definitely more difficult than that given its nature. It would be interesting to test our approach with a different crowd, probably more specialized in software engineering work

(e.g., Topcoder [81] or Upwork [82]), and compare the results.

**It takes time to collect a diverse set of solutions.** It took us one week per experiment to collect solution alternatives for the UI experiments, and twice the time to collect a similar amount of solution alternatives for the IC experiments. Compared to the competition model, in which tasks are usually on the order of a few weeks, these results do not represent the competitive advantage of the microtask model as being fast, which was somewhat disappointing. Individual workers did complete the task in a matter of minutes, as pointed out in many other studies about Amazon Mechanical Turk (e.g., [25, 41]). Different from those studies where the whole set of tasks is completed in a short time, in our experiments it became harder to attract workers once the HIT started to be several days old, and the difference in time between one worker finishing and the next worker joining started to increase over time. Parallelism, thus, only resulted in diversity and quality being much broader, but not a speed up. Perhaps this could be because the Amazon Mechanical Turk crowd has just not that many software oriented workers. It would be interesting, again, to test how the approach on different crowds and analyze the impact on speed.

# 7

## Threats to Validity

As all studies, our study has several limitations. Several threats to validity exist that must be kept in mind when considering our results. We first describe our internal threats to validity, i.e., the factors that affect our ability to claim cause–effect relationships from the results. Following, we describe the conditions of the experiment that may pose a risk to its external validity, i.e., its ability to generalize to what workers do.

### 7.1 Internal Threats to Validity

Internal validity refers to whether or not an experimental treatment or experimental condition makes a difference in the results. First, while the CrowdDesign platform was exclusively designed for the presented set of experiments, it could have limited the performance of the workers. Many workers may have their favorite design tool, with features that differ from the features of our platform. Moreover, because the platform has its own features, a learning curve exists that may have negatively influenced the design performance.

Second, the breakdown of the traffic simulator problem into decision points was made based

on an extensive analysis of multiple designs created to solve it [66]. However, there could exist multiple other ways to subdivide the problem into decision points, which in turn may have lead to different results. Indeed, our results are based on just eight decision points thus far. Their wording, specific instructions, and domain could have influence the performance of workers.

Third, the design task may not be representative of design tasks in the real world. Even though the structure of the task resembles the structure of a User Story [18] (with a general goal and several acceptance criteria) and, even though the task has been used in other experiments [48, 66], it may not be representative of design tasks in the real world.

Fourth, our results are based on a mix of quantitative and qualitative analysis. While data analysis involving coding may introduce bias, we used several mechanisms to reduce and mitigate potential sources of bias. In scoring solution alternatives, panelists independently scored each of the designs. Members of the panel were blind to the worker and the experiment, and the order in which solution alternatives were presented to the panelists was randomized. In clustering solution alternatives and identifying categories, panelists also were blind, both to the worker and the experiment. Finally, in analyzing the exit survey and quit survey data, two members of the research team independently coded each response to identify insights before the coded insights were organized into themes. Overall, then, bias may exist, but we have taken measures to mitigate it.

## 7.2 External Threats to Validity

Threats to external validity compromise our confidence in stating whether the study results are applicable to other groups. For our particular study, the crowd that we attracted may not be a crowd that is representative of what other design tasks may attract. Despite being

an open call, some factors influenced the recruitment of workers. For instance, all HITs were posted between 7am and 9am EST. With this decision we sought to attract as many workers as we could within the United States. Posts at other times may attract different crowds. As another example, our HIT was refreshing and of interest to workers. If there are more HITs like ours, self-selection of workers and tasks may differ.

Second, despite the qualification test being carefully designed, its structure could have affected the participation. It may have attracted workers who are less qualified or failed to attract workers of better competence.

Third, the characteristics of the Amazon Mechanical Turk crowd differ from other crowds (e.g., Topcoder or Upwork). Therefore, our results can not be generalized to these other crowds.



# 8

## Conclusion and Future Work

Crowdsourcing is gaining a place in different fields as a viable approach for solving a variety of problems. At this moment in time, the challenge is to use crowdsourcing for more complex problems. So it is in software engineering where some “easy” problems can be and are crowdsourced, but the applicability of crowdsourcing to complex activities is unclear.

In this thesis, we presented our first step in the exploration of microtask crowdsourcing for one such more complex task: software design. We reported on the results of four experiments on Amazon Mechanical Turk, where workers provided between one and five solution alternatives for small, partial design problems for an educational traffic light simulator. We studied with two experimental conditions: (1) the kind of design (UI design versus internal code design), and (2) the exposure of workers to examples (workers being exposed to previous workers’ work versus workers working independently). We found that it is feasible for a crowd to generate a broad range of solution alternatives, although these solutions are of a variety of quality levels. Regarding the exposure to examples, we observed a drop in both terms of quality and diversity when workers had access to previous workers’ work. What is important, those solution alternatives that are quite exceptional in describing complete and innovative

solutions exists. We found in the set of solution alternatives a great number of ‘top designs’, characterized for their high quality score and high number of requirements met. The crowd, indeed, succeed in creating a solid basis for next steps designing through the morphological chart. Further, the fact that we could successfully perform large-scale experiments related to software design and the fact that we could involve non-professionals in the process, are perhaps two of the most important general conclusions of this work.

Substantial work remains to be done. Future research might extend the reported experiment in three ways. First, all our participant workers were recruited through Amazon Mechanical Turk. Other populations, and in particular expert populations (such as Topcoder or Upwork), might have different ways of solving and approaching software design microtasks.

Second, we want to explore whether filtering out “bad” examples could positively influence the quality and diversity results. That is, we want to experiment with different quality levels of the set of examples that are revealed to workers, and observe the effects on the resulting designs both in quality and diversity.

Third, the approach might be extended towards a more iterative process. Perhaps solution alternatives can be further improved with a second phase of revision, improvement, and, possibly, recombination. Previous research in crowdsourcing design out of the software field [88] has shown early success with such a two phase process in a competition model, and we wonder if it is possible to obtain similar results in a microtasking model.

This thesis presented the results of the first step of a broader research agenda, in which we addressed just one of three overarching research questions about microtasking software design with a morphological chart. Our ultimate goal is to involve the crowd in carrying out the whole creation of the morphological chart (decision points and solution alternatives), and the identification of a final solution out of the resulting morphological chart. Future steps include the exploration of mechanisms to involve workers in the break down of the design

problem into decision points, and mechanisms for selecting a single complete design from a morphological chart.

# Bibliography

- [1] 99designs. <http://99designs.com/>.
- [2] 99tests. <https://99tests.com/>.
- [3] Amara. <http://www.amara.org/>.
- [4] Amazon Mechanical Turk. <https://www.mturk.com/>.
- [5] Araujo, R. M. 99designs: An analysis of creative competition in crowdsourced design. *First AAAI conference on Human computation and crowdsourcing* (2013).
- [6] Arcbazar. <http://www.arcbazar.com/>.
- [7] Bernstein, M. S., Little, G., Miller, R. C., Hartmann, B., Ackerman, M. S., Karger, D. R., Crowell, D., and Panovich, K. Soylent: a word processor with a crowd inside. *Communications of the ACM* 58, 8 (2015), 85–94.
- [8] Boudreau, K. J., and Lakhani, K. R. open disclosure of innovations, incentives and follow-on reuse: Theory on processes of cumulative innovation and a field experiment in computational biology. *Research Policy* 44, 1 (2015), 4–19.
- [9] Bountify. <https://bountify.co/>.
- [10] Brabham, D. C. Crowdsourcing as a model for problem solving an introduction and cases. *Convergence: the international journal of research into new media technologies* 14, 1 (2008), 75–90.
- [11] Breaux, T. D., and Schaub, F. Scaling requirements extraction to the crowd: Experiments with privacy policies. *IEEE 22nd International Requirements Engineering Conference (RE)* (2014), IEEE, pp. 163–172.
- [12] Bugcrowd. <https://bugcrowd.com/>.
- [13] Campbell, D. T. Blind variation and selective retentions in creative thought as in other knowledge processes. *Psychological review* 67, 6 (1960), 380.
- [14] CastingWords. <https://castingwords.com/>.
- [15] Chawla, S., Hartline, J. D., and Sivan, B. Optimal crowdsourcing contests. *Games and Economic Behavior* (2015).

- [16] Chilton, L. B., Little, G., Edge, D., Weld, D. S., and Landay, J. A. Cascade: Crowdsourcing taxonomy creation. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2013), pp. 1999–2008.
- [17] Code Hunt. <http://research.microsoft.com/en-us/projects/codehunt/>.
- [18] Cohn, M. *User stories applied: For agile software development*. Addison-Wesley Professional, 2004.
- [19] CrowdSource. <http://www.crowdsourcing.com/>.
- [20] CrowdStudio. <https://www.crowdstudio.com/>.
- [21] Dawson, R., and Bynghall, S. *Getting results from crowds*. Advanced Human Technologies San Francisco, 2012.
- [22] Doan, A., Ramakrishnan, R., and Halevy, A. Y. Crowdsourcing systems on the worldwide web. *Communications of the ACM* 54, 4 (2011), 86–96.
- [23] Dym, C. L., and Little, P. *Engineering design: A project based approach*, 2000.
- [24] Eppinger, S. D., and Ulrich, K. T. *Product design and development*. 1995 (1995).
- [25] Gao, Y., Chen, Y., and Liu, K. On cost-effective incentive mechanisms in microtask crowdsourcing. *Computational Intelligence and AI in Games, IEEE Transactions on* 7, 1 (2015), 3–15.
- [26] Gengo. <https://gengo.com/>.
- [27] Giroto, V. Collective creativity through a micro-tasks crowdsourcing approach. *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion* (2016), pp. 143–146.
- [28] Goldman, M., Little, G., and Miller, R. C. Collabode: collaborative coding in the browser. *Proceedings of the 4th international workshop on Cooperative and human aspects of software engineering* (2011), pp. 65–68.
- [29] Goldman, M., Little, G., and Miller, R. C. Real-time collaborative coding in a web ide. *Proceedings of the 24th annual ACM symposium on User interface software and technology* (2011), pp. 155–164.
- [30] Haik, Y., Shahin, T., and Sivaloganathan, S. *Engineering design process*. Cengage Learning, 2010.
- [31] Hanington, B., and Martin, B. *Universal methods of design: 100 ways to research complex problems, develop innovative ideas, and design effective solutions*. Rockport Publishers, 2012.
- [32] Hartmann, B., MacDougall, D., Brandt, J., and Klemmer, S. R. What would other programmers do: suggesting solutions to error messages. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2010), pp. 1019–1028.

- [33] Heer, J., and Bostock, M. Crowdsourcing graphical perception: using mechanical turk to assess visualization design. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2010), pp. 203–212.
- [34] Howe, J. The rise of crowdsourcing. *Wired magazine* 14, 6 (2006), 1–4.
- [35] Howe, J. Crowdsourcing: Why the power of the crowd is driving the future of business. *Crown Business, New York* (2008).
- [36] Huang, Y.-C., Wang, C.-I., and Hsu, J. Leveraging the crowd for creating wireframe-based exploration of mobile design pattern gallery. *Proceedings of the companion publication of the 2013 international conference on Intelligent user interfaces companion* (2013), pp. 17–20.
- [37] HYVE Crowd. <https://www.hyvecrowd.net/>.
- [38] Jansson, D. G., and Smith, S. M. Design fixation. *Design studies* 12, 1 (1991), 3–11.
- [39] Kaufmann, N., Schulze, T., and Veit, D. More than fun and money. worker motivation in crowdsourcing-a study on mechanical turk. *AMCIS* (2011), vol. 11, pp. 1–11.
- [40] Kittur, A. Crowdsourcing, collaboration and creativity. *ACM Crossroads* 17, 2 (2010), 22–26.
- [41] Kittur, A., Chi, E. H., and Suh, B. Crowdsourcing user studies with mechanical turk. *Proceedings of the SIGCHI conference on human factors in computing systems* (2008), pp. 453–456.
- [42] Kittur, A., Smus, B., Khamkar, S., and Kraut, R. E. Crowdforge: Crowdsourcing complex work. *Proceedings of the 24th annual ACM symposium on User interface software and technology* (2011), pp. 43–52.
- [43] Kohn, N. W., and Smith, S. M. Collaborative fixation: Effects of others’ ideas on brainstorming. *Applied Cognitive Psychology* 25, 3 (2011), 359–371.
- [44] Kulkarni, A., Can, M., and Hartmann, B. Collaboratively crowdsourcing workflows with turkomatic. *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work* (2012), pp. 1003–1012.
- [45] Lakhani, K., Garvin, D. A., and Lonstein, E. Topcoder (a): Developing software through crowdsourcing. *Harvard Business School General Management Unit Case*, 610-032 (2010).
- [46] Lampel, J., Jha, P. P., and Bhalla, A. Test-driving the future: How design competitions are changing innovation. *The Academy of Management Perspectives* 26, 2 (2012), 71–85.
- [47] Lasecki, W. S., Kim, J., Rafter, N., Sen, O., Bigham, J. P., and Bernstein, M. S. Apparition: Crowdsourced user interfaces that come to life as you sketch them. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (2015), pp. 1925–1934.

- [48] LaToza, T. D., Chen, M., Jiang, L., Zhao, M., and Van Der Hoek, A. Borrowing from the crowd: A study of recombination in software design competitions. *Proceedings of the 37th International Conference on Software Engineering-Volume 1* (2015), IEEE Press, pp. 551–562.
- [49] LaToza, T. D., Di Lecce, A., Ricci, F., Towne, W. B., and van der Hoek, A. Ask the crowd: Scaffolding coordination and knowledge sharing in microtask programming. *Visual Languages and Human-Centric Computing (VL/HCC), 2015 IEEE Symposium on* (2015), IEEE, pp. 23–27.
- [50] LaToza, T. D., Garlan, D., Herbsleb, J. D., and Myers, B. A. Program comprehension as fact finding. *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering* (2007), ACM, pp. 361–370.
- [51] LaToza, T. D., Towne, W. B., Adriano, C. M., and Van Der Hoek, A. Microtask programming: Building software with a crowd. *Proceedings of the 27th annual ACM symposium on User interface software and technology* (2014), pp. 43–54.
- [52] LaToza, T. D., Towne, W. B., Van Der Hoek, A., and Herbsleb, J. D. Crowd development. *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)* (2013), IEEE, pp. 85–88.
- [53] LaToza, T. D., and van der Hoek, A. Crowdsourcing in software engineering: Models, motivations, and challenges. *IEEE Software* 33, 1 (2016), 74–80.
- [54] Li, W., Seshia, S. A., and Jha, S. Crowdmine: towards crowdsourced human-assisted verification. *Proceedings of the 49th Annual Design Automation Conference* (2012), pp. 1254–1255.
- [55] Lim, S. L., Quercia, D., and Finkelstein, A. Stakesource: harnessing the power of crowdsourcing and social networks in stakeholder analysis. *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2* (2010), pp. 239–242.
- [56] Lorge, I., Fox, D., Davitz, J., and Brenner, M. A survey of studies contrasting the quality of group performance and individual performance, 1920-1957. *Psychological bulletin* 55, 6 (1958), 337.
- [57] Luther, K., Tolentino, J.-L., Wu, W., Pavel, A., Bailey, B. P., Agrawala, M., Hartmann, B., and Dow, S. P. Structuring, aggregating, and evaluating crowdsourced design critique. *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing* (2015), pp. 473–485.
- [58] Mao, K., Capra, L., Harman, M., and Jia, Y. A survey of the use of crowdsourcing in software engineering. *RN* 15 (2015), 01.
- [59] McConnel, S. What does 10x mean? measuring variations in programmer productivity. *Making Software, OReilly* 16 (2011).

- [60] Mullen, B., Johnson, C., and Salas, E. Productivity loss in brainstorming groups: A meta-analytic integration. *Basic and applied social psychology* 12, 1 (1991), 3–23.
- [61] Nebeling, M., Leone, S., and Norrie, M. C. Crowdsourced web engineering and design. In *Web Engineering*. Springer, 2012, pp. 31–45.
- [62] Noy, N. F., Mortensen, J., Alexander, P. R., and Musen, M. A. Mechanical turk as an ontology engineer. *Using microtasks as a component of an ontology engineering workflow*. *Web Sci* (2013).
- [63] Park, C. H., Son, K., Lee, J. H., and Bae, S.-H. Crowd vs. crowd: large-scale cooperative design through open team competition. *Proceedings of the 2013 conference on Computer supported cooperative work* (2013), pp. 1275–1284.
- [64] passbrains. <https://www.passbrains.com/>.
- [65] Pastore, F., Mariani, L., and Fraser, G. Crowdoracles: Can the crowd solve the oracle problem? *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on* (2013), IEEE, pp. 342–351.
- [66] Petre, M., and Van Der Hoek, A. *Software Designers in Action: A Human-Centric Look at Design Work*. CRC Press, 2013.
- [67] Retelny, D., Robaszekiewicz, S., To, A., Lasecki, W. S., Patel, J., Rahmati, N., Doshi, T., Valentine, M., and Bernstein, M. S. Expert crowdsourcing with flash teams. *Proceedings of the 27th annual ACM symposium on User interface software and technology* (2014), pp. 75–85.
- [68] Robillard, M. P., Coelho, W., and Murphy, G. C. How effective developers investigate source code: An exploratory study. *Software Engineering, IEEE Transactions on* 30, 12 (2004), 889–903.
- [69] Saxton, G. D., Oh, O., and Kishore, R. Rules of crowdsourcing: Models, issues, and systems of control. *Information Systems Management* 30, 1 (2013), 2–20.
- [70] Schenk, E., and Guittard, C. Towards a characterization of crowdsourcing practices. *Journal of Innovation Economics & Management*, 1 (2011), 93–107.
- [71] Slogan Slings. <http://www.sloganslingers.com/>.
- [72] Smith, G., Richardson, J., Summers, J. D., and Mocko, G. M. Concept exploration through morphological charts: an experimental study. *Journal of mechanical design* 134, 5 (2012), 051004.
- [73] Stack Overflow. <http://stackoverflow.com/>.
- [74] StakeSource. <http://www0.cs.ucl.ac.uk/research/StakeSource/>.



- [75] Stol, K.-J., and Fitzgerald, B. Two’s company, three’s a crowd: a case study of crowd-sourcing software development. *Proceedings of the 36th International Conference on Software Engineering* (2014), pp. 187–198.
- [76] Surowiecki, J. *The wisdom of crowds*. Anchor, 2005.
- [77] Testbirds. <https://www.testbirds.com/>.
- [78] Threadless. <https://www.threadless.com/>.
- [79] Tillmann, N., Bishop, J., Horspool, N., Perelman, D., and Xie, T. Code hunt: Searching for secret code for fun. *Proceedings of the 7th International Workshop on Search-Based Software Testing* (2014), pp. 23–26.
- [80] Tonga. <https://tonga1.com/>.
- [81] Topcoder. <https://www.topcoder.com/>.
- [82] Upwork. <https://www.upwork.com/>.
- [83] uTest. <https://www.utest.com/>.
- [84] Von Ahn, L., and Dabbish, L. Labeling images with a computer game. *Proceedings of the SIGCHI conference on Human factors in computing systems* (2004), pp. 319–326.
- [85] Wordy. <http://wordy.com/>.
- [86] Xu, A., Huang, S.-W., and Bailey, B. Voyant: generating structured feedback on visual designs using a crowd of non-experts. *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing* (2014), pp. 1433–1444.
- [87] Xue, H. *Using redundancy to improve security and testing*. PhD thesis, University of Illinois at Urbana-Champaign, 2013.
- [88] Yu, L., and Nickerson, J. V. Cooks or cobblers?: crowd creativity through combination. *Proceedings of the SIGCHI conference on human factors in computing systems* (2011), pp. 1393–1402.
- [89] Zaidan, O. F., and Callison-Burch, C. Crowdsourcing translation: Professional quality from non-professionals. *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1* (2011), Association for Computational Linguistics, pp. 1220–1229.
- [90] Zhang, H., Law, E., Miller, R., Gajos, K., Parkes, D., and Horvitz, E. Human computation tasks with global constraints. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2012), pp. 217–226.

# Appendices

## A Consent Form and Demographics Questionnaire

**University of California, Irvine  
Study Information Sheet**

**Programming Online Study**

**Faculty Sponsor and Lead Researcher**

Professor Adriaan W. van der Hoek

Department of Informatics

Donald Bren School of Information and Computer Sciences

andre@ics.uci.edu

949-824-6326

- You are being asked to participate in a research study to perform some programming tasks related to software design, coding, debugging, and testing.
- Programming tasks will be performed in an online tool that consists of an external website accessible via a link in a Mechanical Turk task (HIT - Human Intelligent Task).
- The purpose of the study is to better understand the challenges developers face in using tools to answer their questions about code and to help inform the design of new tools that help developers to work more effectively.
- You are eligible to participate in this study if you are at least 18 years of age or older; are fluent in English; and have at least minimal programming skills.
- The research procedures involve using an online software development tool and will last approximately from 5 to 45 minutes.
- There are no risks/discomforts associated with the study. No personal information will be collected.
- There are no direct benefits from participation in the study. However, this study may help us to better understand how programmers work with tools.

- You will be paid the equivalent of 9 dollars per hour, which is California minimal wage, prorated by the expected length of the task to be completed. You will be paid through Amazon Mechanical Turk. At the end of the study, you will be given a code to enter in your HIT (Human Intelligent Task) that confirms that you participated.
- All research data collected will be stored securely and confidentially in encrypted files. At the end of the study, the original answers to demographics questions will be deleted from our files.
- The research team and authorized UCI personnel may have access to your study records to protect your safety and welfare. Any information derived from this research project that personally identifies you will not be voluntarily released or disclosed by these entities without your separate consent, except as specifically required by law.
- If you have any comments, concerns, or questions regarding the conduct of this research please contact the researchers listed at the top of this form.
- Please contact UCI's Office of Research by phone, (949) 824-6662, by e-mail at IRB@research.uci.edu or at 5171 California Avenue, Suite 150, Irvine, CA 92617 if you are unable to reach the researchers listed at the top of the form and have general questions; have concerns or complaints about the research; have questions about your rights as a research subject; or have general comments or suggestions.
- Participation in this study is voluntary. There is no cost to you for participating. You may refuse to participate or discontinue your involvement at any time without penalty. You are free to withdraw from this study at any time. If you decide to withdraw from this study you should notify the research team immediately by clicking on the "No, thanks" button below.

*By checking this box I hereby state that "I have read the study information sheet and want to proceed with this study".*

No, thanks

Yes, I want to participate

**Thank you for your interest in CrowdDesign and for helping us finding out better ways to design software by using the power of the crowd.**

**Please let us know a little bit more about you. This will help us to design future tasks.**

I am currently a:

- Professional software developer
- Professional UI/UX designer
- Graduate student
- Undergraduate student
- Hobbyist
- Other

How many years of experience do you have?

Where did you learn your skills (mark all that apply)?

- High school
- College/University
- On the web
- Other

What is your gender?

- Female
- Male
- Other
- Prefer not to tell

What is your age?

What is your country of residence?

## B Qualification tests

### B.1 User Interface Design Qualification Tests

**Before we allow you to continue, we need to evaluate your skills. Please answer the following questions.**

1 - A good User Interface design can improve the user experience of an application. Which of the following statements about user interfaces are correct?

I. Controls and other objects necessary for the successful use of software have to be visibly accessible at all times

II. Users are capable of learning quickly, therefore after giving instructions once they will not need them again

- I is correct and II is false
- I is false and II is correct
- Both I and II are correct
- Both I and II are false

2 - The aesthetics of an application can influence the user experience a lot. Which of the following principles about aesthetics in design is not true?

- Aesthetic design should be left to those schooled and skilled in its application(e.g. graphic and visual designers)
- Fashion should never be put before usability
- Aesthetics should lead the design of software
- Test the visual design as thoroughly as the behavioral design

3 - When considering design principles which of the following is true?

- Efficiency comes before learnability
- Discoverability comes before consistency
- None of these are true
- Simplicity comes before usability

4 - What is a problem, also known as the "Illusion of Simplicity", that can happen when focusing too much on simplicity?

- Creating optical illusions on your website will mislead the user
- Simplicity improves usage patterns
- Creating simplicity will harm the usability
- Hiding complexity, in favour of simplicity, will actually increase it

5 - Giving feedback about the system status to the user is an important part of its design. The system should always keep users informed about what is going on. Which of the following statements is an example of such feedback?

- The system shows a progress indicator when it is loading
- The system prompts the user for his password when logging in
- The system gives the user a tutorial about how to use a new feature
- The system sends an email to the user informing about the latest developments

**Before we allow you to continue, we need to evaluate your skills. Please answer the following questions.**

1 - Even better than good error messages is a careful design, which prevents a problem from occurring in the first place. Which of the following statements is not an example of preventing errors

- Google auto-complete
- Amazon's "you might also like" product recommendations
- Autocorrect of grammar and spelling
- Conflicting buttons, like "cancel" and "submit", are clearly kept separated

2 - Placeholder text is used in text fields as a temporary solution until a proper value or variable can be assigned. Which of the following statements about placeholder text fields are correct?

I. Placeholder text within a field should be easy to replace

II. The word "default" is a meaningful and responsive term to put in a default text field

- I is correct and II is false
- I is false and II is correct
- Both I and II are correct
- Both I and II are false

3 - When users need to read text in your application it is important that it presented in a way that does not harm its readability. Which of the following could harm the readability of text in your application

- Text with a high contrast
- Text that is colored to stand out
- Unique labels for menu's and buttons
- Font sizes that are large enough to be readable on standard displays

4 - When considering design principles which of the following is true

- Simplicity comes before usability
- Discoverability comes before consistency
- Efficiency comes before learnability
- None of these are true

5 - While using your website an user clicks a button but unfortunately an error occurs. What should your website show to the user

- The user gets a detailed message containing all the information about the error including the stack trace and an option to retry the action
- The user gets shown a blank page with an error code
- The user gets a message that an error occurred and is asked to try again
- The user gets navigated back to the homepage

Quit

Submit answers

**Before we allow you to continue, we need to evaluate your skills. Please answer the following questions.**

1 - Giving feedback about the system status to the user is an important part of its design. The system should always keep users informed about what is going on. Which of the following statements is an example of such feedback

- The system sends an email to the user informing about the latest developments
- The system prompts the user for his password when logging in
- The system gives the user a tutorial about how to use a new feature
- The system shows a progress indicator when it is loading

2 - When users need to read text in your application it is important that it presented in a way that does not harm its readability. Which of the following could harm the readability of text in your application

- Text with a high contrast
- Font sizes that are large enough to be readable on standard displays
- Text that is colored to stand out
- Unique labels for menu's and buttons

3 - When considering design principles which of the following is true

- Simplicity comes before usability
- Discoverability comes before consistency
- Efficiency comes before learnability
- None of these are true

4 - When designing an user interface it is important to make your designs consistent. Which of the following statements about consistency in design is correct?

I. Offer users consistent visual cues for a sense of "home"

II. Users do not have to be informed when they face dela

- I is correct and II is false
- I is false and II is correct
- Both I and II are correct
- Both I and II are false

5 - What is a problem, also known as the "Illusion of Simplicity", that can happen when focusing to much on simplicity?

- Creating simplicity will harm the usability
- Hiding complexity, in favour of simplicity, will actually increase it
- Creating optical illusions on your website will mislead the user
- Simplicity improves usage patterns

Quit

Submit answers

**Before we allow you to continue, we need to evaluate your skills. Please answer the following questions.**

1 - There are many known common design mistakes. Which of the following is not a design mistake?

- Prompting alert boxes to a user
- Using very small fonts
- Having a confusing navigation
- Not informing the user about a successful submission

2 - What is a problem, also known as the "Illusion of Simplicity", that can happen when focusing too much on simplicity?

- Creating optical illusions on your website will mislead the user
- Simplicity improves usage patterns
- Creating simplicity will harm the usability
- Hiding complexity, in favour of simplicity, will actually increase it

3 - When designing an user interface it is important to make your designs consistent. Which of the following statements about consistency in design is correct?

I. Users should not have to wonder whether different words, situations, or actions mean the same thing.  
II. It is just as important to be visually inconsistent when things act differently as it is to be visually consistent when things act the same

- I is correct and II is false
- I is false and II is correct
- Both I and II are correct
- Both I and II are false

4 - Placeholder text is used in text fields as a temporary solution until a proper value or variable can be assigned. Which of the following statements about placeholder text fields are correct?

I. The word "default" is a meaningful and responsive term to put in a default text field  
II. Placeholder text within a field should be easy to replace

- I is correct and II is false
- I is false and II is correct
- Both I and II are correct
- Both I and II are false

5 - A good User Interface design can improve the user experience of an application. Which of the following statements about user interfaces are correct?

I. Controls and other objects necessary for the successful use of software do not have to be visibly accessible at all times  
II. Users are capable of learning quickly, therefore after giving instructions once they will not need them again

- I is correct and II is false
- I is false and II is correct
- Both I and II are correct
- Both I and II are false



## B.2 Internal Code Design qualification Tests

Before we allow you to continue, we need to evaluate your skills. Please answer the following questions.

The source code below is used for all the questions below.

```
1 public class WordFinder {
2
3     private String sentence;
4
5     public WordFinder(String text){
6         this.sentence = text;
7     }
8
9     public int wordPosition(String wordToBeFound){
10        String[] wordlist = this.sentence.split(" ");
11        int found=-1;
12        for (int i=0; i<wordlist.length;i++) {
13            if (wordlist[i].compareTo(wordToBeFound)==0)
14                found= i;
15        }
16        return found;
17    }
18
19    public static void main(String[] args) {
20        String mySentence = "cogito ergo sum";
21        WordFinder finder = new WordFinder(mySentence);
22        System.out.println(finder.wordPosition("cogito"));
23    }
24 }
```

1 - What is the output of the code above

- 1
- 0
- 1
- 2
- 3

2 - What would have been the output if the variable mySentence was set to "cogito ergo cogito" at line 20

- 1
- 0
- 1
- 2
- 3

3 - What would have been the output if instead of "int i = 0;" we had "int i = 1;" at line 12

- 1
- 0
- 1
- 2
- 3

4 - What would have been the output if instead of "found = i;" we had "return i;" at line 14

- 1
- 0
- 1
- 2
- 3

5 - What line in the program would have caused a Null Pointer exception if we set the variable mySentence to null at line 20?

- 21
- 22
- 6
- 10
- 13

Before we allow you to continue, we need to evaluate your skills. Please answer the following questions.

The source code below is used for all the questions below.

```
1 public class FindTop {
2
3     private int[] numbers;
4
5     public FindTop(int[] numbersArg){
6         this.numbers = numbersArg;
7     }
8
9     public int findHighest(int lowIndex, int highIndex){
10        int top = this.numbers[lowIndex];
11        for(int i = lowIndex; i <= highIndex; i++){
12            if (top < this.numbers[i])
13                top = this.numbers[i];
14        }
15        return top;
16    }
17
18    public static void main(String[] args) {
19        int myNumbers[] = {10, 5, 2, 4, 8};
20        FindTop numbers = new FindTop(myNumbers);
21        System.out.println( numbers.findHighest(1, 4));
22    }
23 }
```

1 - What is the output of the code above

- 1
- 0
- 2
- 4
- 8

2 - What would have been the output if we had "numbers.findHighest(1;1)" at line 21

- 10
- 5
- 2
- 4
- 8

3 - What would have been the output if instead of "int i = lowIndex;" we had "int i = 0;" at line 11

- 10
- 5
- 2
- 4
- 8

4 - What would have been the output if we had "return i;" at line 13

- 10
- 5
- 2
- 4
- 8

5 - What line in the program would have caused an `ArrayIndexOutOfBoundsException` exception if we had "numbers.findHighest(0;5)" at line 21

- 6
- 10
- 11
- 12
- 13

Quit Submit answers

Before we allow you to continue, we need to evaluate your skills. Please answer the following questions.

The source code below is used for all the questions below.

```
1 public class PositionFinder {
2
3     public static void findPosition(String[] list, String term){
4         if (list==null || list.length==0)
5             System.out.print("Empty array");
6         else{
7             int position=-1;
8             for (String name: list){
9                 if(name.compareTo(term)==0)
10                    break;
11                position++;
12            }
13            System.out.print(position > 0 ? position: 0);
14        }
15    }
16
17    public static void main(String[] args){
18        String[] myArray = {"Hola", "Kunusta", "Hello", "Ciao"};
19        findPosition(myArray, "Ciao");
20    }
21 }
```

1 - What is the output of the code?

- 1
- 0
- 1
- 2
- 3

2 - What would have been the output if instead of "Ciao" we had "Hola" at line 19

- 1
- 0
- 1
- 2
- 3

3 - What would have been the output if instead of "Ciao" we had an empty String (e.g. "") at line 19

- 1
- 0
- 1
- 2
- 3

4 - What would have been the output if we had "String[] myArray = {};" at line 18

- Empty array
- NullPointerException
- ArrayOutOfBoundsException
- 0

5 - What line of the program would have caused a NullPointerException if we set myArray to {"Hola"; null} at line 18

- 19
- 20
- 6
- 9
- 10

[Out](#) [Submit answers](#)

## C Tasks description

### C.1 User Interface Design Tasks

#### Creating Maps

##### Task:

Design an interface mechanism through which users build maps with roads and intersections.

Sketch solutions that cover the following requirements:

- The user can create a simple visual map of roads on an empty, rectangular canvas.
- The user can create a map that supports at least 6 intersections.
- Roads may only lead to 4-way intersections (3-way intersections are not allowed).
- The user can create a map that allows roads of varying lengths, with different arrangements of intersections.

##### Tips:

- You don't need to support very complex maps. Try to focus on the different user interactions your solutions need to have to satisfy the requirements.

##### Reminder:

We are not looking for one perfect design but are interested in a variety of designs that each can have their own pro's and con's.

## Setting Timing of Traffic Lights

### Task:

Design an interface mechanism through which users set the timing of green, yellow, and red for the traffic lights on an intersection.

### Sketch solutions that cover the following requirements:

- Your intersection should support left hand turns.
- Your solution must avoid letting the user set timings that allow car crashes.
- The intersection should support (optional) use of sensors to detect cars waiting.
- You only have to design a solution for a 4-way intersection.

### Tips:

- Try to focus on what settings the user needs to configure to set traffic lights timings that meet the requirements.
- Think about the different ways the user can manipulate these settings.
- Optionally, you can also think about how your system would work with multiple intersections.

### Reminder:

We are not looking for one perfect design but are interested in a variety of designs that each can have their own pro's and con's.

## Visualizing the State of The Simulation

### Task:

Design an interface mechanism through which users are informed of the state of traffic and traffic light timings.

Sketch solutions that cover the following requirements:

- The users must be able to see how their traffic light timings influence the traffic.
- The feedback should inform the user about traffic jams on his/her road system and provide information that helps him/her to avoid these traffic jams.
- The feedback must support a road system with at least 6 intersections and provide both information on the intersection level as on the total road system level.
- Only 4-way intersections are allowed.

### Tips:

- You don't need to support very complex maps. Try to focus on which information the user needs to satisfy all the requirements.
- Think about the different ways you can provide this information to the user.

### Reminder:

We are not looking for one perfect design but are interested in a variety of designs that each can have their own pro's and con's.

## Determining the Flow of Traffic

### Task:

Design an interface mechanism through which users control where traffic goes on a map.

### Sketch solutions that cover the following requirements:

- Your solutions should allow users to control where cars enter and exit on the map, as well as how much traffic flows into the map from each entrance.
- Your solution should support a map of at least six 4-way intersections.
- Your solution should allow user to specify the behavior of the traffic; that is, decide what it does when it encounters a traffic light.
- Your solution should support a large amount of traffic.

### Tips:

- You don't have to support a complex system to drive cars. Try to focus on the settings the user needs to configure the direction of traffic flows.
- Think about the different ways the user can manipulate these settings.

### Reminder:

We are not looking for one perfect design but are interested in a variety of designs that each can have their own pro's and con's.

## C.2 Internal Code Design Tasks

### Representing the Road System

#### Task:

Sketch and describe the internal design through which the road system is represented by the code, including maps, roads, intersections, and traffic lights. We are not looking for the user interface design, but the design of the code that captures the state of the simulation.

Your design should cover the following:

- A map that consists of roads and intersections.
- Roads that may vary in length.
- All intersections are 4-way and have traffic lights on all sides.
- Left turn signals must be accommodated.

#### Tips:

- Try to focus on the different parts of the internal design and the information each of the parts needs to capture to satisfy the requirements.
- To illustrate your solution you may use any notation, from a simple drawing or sketch of the code, to pseudocode, to one or more UML diagrams, or any other representation you find convenient. This is up to you; you are free to explain your solution in any way that you believe would help a programmer to understand and implement your design.

#### Reminder:

We are not looking for one perfect design but are interested in a variety of designs that each can have their own pro's and con's.



## Representing Cars

### Task:

Sketch and describe the internal design through which cars are represented by the code, including their location and direction of driving on a given map. We are not looking for the user interface design, but the design of the code that captures the state of the simulation.

Your design should cover the following:

- Cars drive on roads.
- Cars travel through intersections, possibly changing direction.
- A map has multiple entry and exit points for cars; cars enter at one, and exist at another.
- Only one type of car exists. No other vehicles types are present.

### Tips:

- Try to focus on the different parts of the internal design and the information each of the parts needs to capture to satisfy the requirements.
- To illustrate your solution, you may use any notation, from a simple drawing or sketch of the code, to pseudocode, to one or more UML diagrams, or any other representation you find convenient. This is up to you; you are free to explain your solution in any way that you believe would help a programmer to understand and implement your design.

### Reminder:

We are not looking for one perfect design but are interested in a variety of designs that each can have their own pro's and con's.

## Moving Cars

### Task:

Sketch and describe the design for the algorithm that moves cars. We are not looking for the user interface design, but the design of the code that advances the state of the simulation internally.

Your design should cover the following:

- The algorithm should advance all cars from where they are now to their next spot.
- Cars can change direction at an intersection (all intersections are 4-way).
- Intersections may have left turn signals.
- Cars respect the rules of traffic lights.

### Tips:

- Do not design all of the entities of the simulator, but focus on the algorithm by which cars advance. You can simply specify what information you expect the various entities to be able to provide your algorithm.
- To illustrate your solution, you may use any notation, from a simple drawing or sketch of the code, to pseudocode, to one or more UML diagrams, or any other representation you find convenient. This is up to you; you are free to explain your solution in any way that you believe would help a programmer to understand and implement your design.

Reminder:

We are not looking for one perfect design but are interested in a variety of designs that each can have their own pro's and con's.

## Changing the Colors of Traffic Lights

### Task:

Sketch and describe the internal design for the algorithm that changes the color of the traffic lights on the map. We are not looking for the user interface design, but the design of the code that advances the state of the simulation internally.

Your design should cover the following:

- The algorithm should advance the state of all traffic lights.
- All traffic lights govern 4-way intersections, with traffic lights on every side, and must support left turns.
- Each traffic light has its own timing, configured by the user of the simulator.
- Each traffic may or may not use sensors (as decided by the user of the simulator), that determine whether traffic is present and influence the traffic light timings as a result.

### Tips:

- Do not design all of the entities of the simulator but focus on the algorithm by which traffic lights change their colors. You can simply specify what information you expect the various entities to be able to provide your algorithm.
- To illustrate your solution, you may use any notation, from a simple drawing or sketch of the code, to pseudocode, to one or more UML diagrams, or any other representation you find convenient. This is up to you; you are free to explain your solution in any way that you believe would help a programmer to understand and implement your design.

### Reminder:

We are not looking for one perfect design but are interested in a variety of designs that each can have their own pro's and con's.

# D Diversity per Worker

## D.1 User Interface Decision Points

### Creating Maps

#### Uplain

Worker ID	Category				
Worker MB1	Blocks				
Worker MB2	Click and drag	Build as you go	Blocks	GPS	
Worker MB3	Traffic Simulation				
Worker MB4	Blocks	Assisted Drawing			
Worker MB5	Click and drag	Pencil-like (Draw)	Pencil-like (Draw)	Pencil-like (Draw)	UI layout and extra features
Worker MB6	UI layout and extra features	UI layout and extra features	Automated using input	UI layout and extra features	UI layout and extra features
Worker MB7	Map Only	Map Only	Map Only	Map Only	Map Only
Worker MB8	Build as you go				
Worker MB9	Nodes	Nodes			
Worker MB10	Click and drag	Click and drag			
Worker MB11	Pencil-like (Draw)		Blocks	Build as you go	
Worker MB12	Click and drag	Click and drag			
Worker MB13	Map Only	Map Only	Map Only	Map Only	Map Only
Worker MB14	Nodes	Nodes	Click and drag		
Worker MB15	UI layout and extra features	UI layout and extra features	UI layout and extra features	UI layout and extra features	Click and drag
Worker MB16	Click and drag				
Worker MB17	Assisted Drawing	Assisted Drawing	Pencil-like (Draw)		
Worker MB18	Blocks	UI layout and extra features			
Worker MB19	Blocks	UI layout and extra features	UI layout and extra features	UI layout and extra features	UI layout and extra features
Worker MB20	Map Only				
Worker MB21	Build as you go	Blocks	Pencil-like (Draw)		

#### Uexamples

Worker ID	Category				
Worker MB1	Map Only	Map Only	Map Only	Map Only	Map Only
Worker MB2	Map Only	Map Only	Map Only	Map Only	Map Only
Worker MB3	Map Only	Map Only			
Worker MB4	Map Only	Map Only	Map Only	Map Only	
Worker MB5	Map Only	Map Only			
Worker MB6	Map Only	Map Only	Map Only		
Worker MB7	Blocks	Blocks	Click and drag	Assisted Drawing	
Worker MB8	Click and drag	Click and drag	UI layout and extra features		
Worker MB9	Blocks	Grid Drawing	Assisted Drawing		
Worker MB10	Blocks	Isolated road properties			
Worker MB11	Map Only	Map Only	Map Only	Map Only	
Worker MB12	Traffic Simulation	Map Only	Isolated road properties	Isolated road properties	Traffic Simulation
Worker MB13	Nodes	Assisted Drawing	Assisted Drawing	Pencil-like (Draw)	Blocks
Worker MB14	Traffic Simulation	Isolated road properties			
Worker MB15	Click and drag	Click and drag	Click and drag	Grid Drawing	Blocks
Worker MB16	Nodes	Grid Drawing	Traffic Simulation	Isolated road properties	Automated using input
Worker MB17	Map Only	Map Only	Map Only	Map Only	Map Only
Worker MB18	Blocks				
Worker MB19	Nodes	Nodes			
Worker MB20	Nodes	Nodes	Grid Drawing	Nodes	
Worker MB21	Click and drag	Assisted Drawing	Isolated road properties	Blocks	
Worker MB22	Grid Drawing				
Worker MB23	Nodes	Blocks	Click and drag	Pencil-like (Draw)	

# Setting Timing of Traffic Lights

## UPlain

Worker ID	Category				
Worker pSL1	Sensors description				
Worker pSL2	Sensor description + timing description	Sensor description + timing description	Traffic Light timing description	Sensor description + timing description	Sensor description + timing description
Worker pSL3	Unknown traffic lights settings				
Worker pSL4	Sensor description + timing description				
Worker pSL5	Menu + sensors				
Worker pSL6	Menu + sensors				
Worker pSL7	Menu + sensors + error prevention	Input boxes for times			
Worker pSL8	Input box + error prevention	Input box + error prevention		Menu + sensors + error prevention	
Worker pSL9	Slider	Sensors description		Slider	
Worker pSL10	Menu + sensors	Traffic Light timing description			
Worker pSL11	Slider	Slider			
Worker pSL12	Input box + error prevention				
Worker pSL13	Input boxes for times	Sensor description + timing description			
Worker pSL14	Menu + sensors + error prevention	Unknown traffic lights settings	Traffic Light Builder (no time)	Click to add or decrease time	Menu + sensors + error prevention
Worker pSL15	Menu + sensors	Slider			
Worker pSL16	Menu + sensors				
Worker pSL17	Sensors description	Sensors description	Sensors description	Sensors description	
Worker pSL18	Input box + error prevention	Unknown traffic lights settings	Input box + error prevention	Click to change light state	Click to change light state
Worker pSL19	Map only				
Worker pSL20	Input box + error prevention	Input box + error prevention			
Worker pSL21	Sensors description				

## UExamples

Worker ID	Category				
Worker eSL1	Input box + error prevention				
Worker eSL2	Input boxes for times				
Worker eSL3	Input boxes for times				
Worker eSL4	Sensors description				
Worker eSL5	Sensors description				
Worker eSL6	Menu + sensors				
Worker eSL7	Map only				
Worker eSL8	Sensor description + timing description	Sensors description	Map only		
Worker eSL9	Sensors description				
Worker eSL10	Traffic Light timing description	Click to add or decrease time	Sensor description + timing description		
Worker eSL11	Map only	Unknown traffic lights settings	Sensors description		
Worker eSL12	Sensors description				
Worker eSL13	Map only	Map only	Map only		
Worker eSL14	Menu + sensors + error prevention	Sensors description			
Worker eSL15	Map only				
Worker eSL16	Sensors description	Traffic Light timing description	Menu + sensors		
Worker eSL17	Sensor description + timing description				
Worker eSL18	Click to add or decrease time				

# Visualizing the State of The Simulation

## Uplain

Worker ID	Category				
Worker pVT1	Route suggestion				
Worker pVT2	Intersection table				
Worker pVT3	Traffic Flow				
Worker pVT4	Traffic light status				
Worker pVT5	Visualizing cars on a map				
Worker pVT6	Tips + colors	Tips + colors			
Worker pVT7	Map table				
Worker pVT8	Bars indicating queues	Route suggestion			
Worker pVT9	Color encoded and traffic light timing				
Worker pVT10	Intersection table	Intersection table	UI Layout and extra features	UI Layout and extra features	Flow rate graph
Worker pVT11	Route suggestion				
Worker pVT12	Tips + colors	Traffic Flow	Tips + colors		
Worker pVT13	Traffic Flow				
Worker pVT14	Route suggestion				
Worker pVT15	Intersection info + color density	Visualizing cars on a map	Color encoded and traffic light timing		
Worker pVT16	Traffic Flow	Visualizing cars on a map			
Worker pVT17	Tips + colors	Tips + colors			
Worker pVT18	Color encoded and traffic light timing				
Worker pVT19	Visualizing cars on a map	Visualizing cars on a map	Map table	Map table	
Worker pVT20	Bars indicating queues				
Worker pVT21	First person design (route suggestion)	Color encoded only			
Worker pVT22	Intersection table				

## Uexamples

Worker ID	Category				
Worker eVT1	Traffic light status	First person design (route suggestion)			
Worker eVT2	Color encoded and traffic light timing	Color encoded only			
Worker eVT3	Traffic light status	Color encoded only	Route suggestion		
Worker eVT4	Traffic light status	Traffic Flow	Route suggestion	Alternative to traffic light Alternative to traffic light	
Worker eVT5	Color encoded and traffic light timing	Color encoded only			
Worker eVT6	Route suggestion				
Worker eVT7	Color encoded only				
Worker eVT8	Route suggestion				
Worker eVT9	Route suggestion				
Worker eVT10	Color encoded and traffic light timing				
Worker eVT11	Route suggestion	Route suggestion			
Worker eVT12	Color encoded and traffic light timing	First person design (route suggestion)	Route suggestion		
Worker eVT13	Intersection info + color density	First person design (route suggestion)	Route suggestion		
Worker eVT14	First person design (route suggestion)	First person design (route suggestion)	First person design (route suggestion)	Traffic light status	First person design (route suggestion)
Worker eVT15	Color encoded only	First person design (route suggestion)	Route suggestion		
Worker eVT16	Intersection table				
Worker eVT17	Traffic Flow				
Worker eVT18	Intersection info + color density				
Worker eVT19	Color encoded only	First person design (route suggestion)			
Worker eVT20	Traffic Flow				
Worker eVT21	Traffic light status	Traffic light status			
Worker eVT22	Route suggestion				

# Determining the Flow of Traffic

## Uplain

Worker ID	Category				
Worker pTF1	Entry/exit points				
Worker pTF2	Traffic Direction	Flow only	Entry/exit points	Flow only	Unknown map settings
Worker pTF3	Direction based on fixed rules	Direction based on fixed rules			
Worker pTF4	Traffic Light				
Worker pTF5	Advance settings menu for all map elements	Advance settings menu for all map elements			
Worker pTF6	Flow only	Flow only			
Worker pTF7	Keystrokes	Keystrokes	Keystrokes	Keystrokes	Traffic Direction
Worker pTF8	Direction based on fixed rules	Traffic Light			
Worker pTF9	Entry/exit points	Map Only	Traffic Direction		
Worker pTF10	Traffic Light				
Worker pTF11	Traffic Direction	Basic unkown lane setting	Unknown map settings	Unknown map settings	Keystrokes
Worker pTF12	Unknown map settings				
Worker pTF13	Entry/exit points	Entry/exit points	Entry/exit points		
Worker pTF14	Advance settings menu for all map elements				
Worker pTF15	Advance settings menu for all map elements				

## Uexamples

Worker ID	Worker ID				
Worker eTF1	Traffic Direction	Flow only	Basic unkown lane setting		
Worker eTF2	Advance settings menu for all map elements				
Worker eTF3	Traffic Direction				
Worker eTF4	Direction based on fixed rules				
Worker eTF5	Flow only				
Worker eTF6	Advance settings menu for all map elements				
Worker eTF7	Traffic Light				
Worker eTF8	Traffic Light	Flow only	Flow only		
Worker eTF9	Advance settings menu for all map elements	Advance settings menu for all map elements			
Worker eTF10	Traffic Direction				
Worker eTF11	Map Only				
Worker eTF12	Flow only	Flow only	Map Only	Map Only	Map Only
Worker eTF13	Traffic Direction	Basic unkown lane setting			
Worker eTF14	Map Only	Map Only	Traffic Direction		
Worker eTF15	Basic unkown lane setting	Basic unkown lane setting	Basic unkown lane setting	Basic unkown lane setting	Basic unkown lane setting
Worker eTF16	Traffic Direction	Traffic Direction			
Worker eTF17	Map Only				

## D.2 Internal Code Decision Points

### Representing the Road System

#### ICplain

Worker ID	Category			
Worker pRS1	Simulation run	Simulation run	Simulation run	
Worker pRS2	Traffic Lights			
Worker pRS3	Map centric	Map centric	Map centric	
Worker pRS4	Tiles			
Worker pRS5	Map centric			
Worker pRS6	High level components	High level components		
Worker pRS7	Traffic Lights	Graphic map	Graphic map	Graphic map
Worker pRS8	Graphic map	Graphic map		
Worker pRS9	Simulation run	Graphic map	Simulation run	
Worker pRS10	Nodes			
Worker pRS11	Traffic model description	Traffic model description		
Worker pRS12	Traffic model description	Traffic Lights		
Worker pRS13	Nodes			
Worker pRS14	Tiles			
Worker pRS15	Graphic map			
Worker pRS16	Map centric			
Worker pRS17	Cars description			
Worker pRS18	Roads and intersections	Map centric		
Worker pRS19	Traffic Lights			

#### ICexamples

Worker ID	Category			
Worker eRS1	Roads and intersections			
Worker eRS2	Cars description	Traffic Lights		
Worker eRS3	Coordinates system	Traffic Lights	Simulation run	Roads and intersections Cars description
Worker eRS4	Roads and intersections	Simulation run		
Worker eRS5	Roads and intersections	Simulation run		
Worker eRS6	Roads and intersections			
Worker eRS7	Roads and intersections	Cars description		
Worker eRS8	Roads and intersections			
Worker eRS9	Traffic model description	Roads and intersections		
Worker eRS10	High level components	Map centric		
Worker eRS11	Simulation run			
Worker eRS12	High level components			
Worker eRS13	Coordinates system			



# Representing Cars

## ICplain

Worker ID	Category				
Worker pCA1	Grip map	Grip map			
Worker pCA2	Coordinates	Coordinates			
Worker pCA3	Moving cars				
Worker pCA4	Moving cars				
Worker pCA5	Coordinates	OO cars and maps elements			
Worker pCA6	Graphical Maps	High level components			
Worker pCA7	OO cars and maps elements				
Worker pCA8	Cars functions	Simulation run (based on lights)	Simulation run (based on lights)		
Worker pCA9	Coordinates	Grip map	Graphical Maps		
Worker pCA10	OO cars and maps elements	Moving cars	Moving cars		
Worker pCA11	Moving cars	Moving cars			
Worker pCA12	OO cars and maps elements	High level components	Cars functions	High level components	Coordinates
Worker pCA13	GPS				
Worker pCA14	Map as graph				
Worker pCA15	Grip map				
Worker pCA16	Map as graph				
Worker pCA17	OO cars and maps elements	Moving cars	Grip map		
Worker pCA18	OO cars and maps elements				
Worker pCA19	Map as graph				

## ICexamples

Worker ID	Category				
Worker eCA1	Moving cars	Simulation run (based on lights)			
Worker eCA2	OO cars and maps elements	Valid state check	Car data structure and map elements	Cars functions	
Worker eCA3	Simulation run (based on lights)				
Worker eCA4	Moving cars	Coordinates	Map as graph	Map as graph	Map as graph
Worker eCA5	Cars functions				
Worker eCA6	OO cars and maps elements	High level components	High level components		
Worker eCA7	OO cars and maps elements				
Worker eCA8	Graphical Maps				
Worker eCA9	Grip map	Grip map	Grip map	Grip map	Grip map
Worker eCA10	OO cars and maps elements				
Worker eCA11	Graphical Maps	High level components			
Worker eCA12	Car data structure and map elements				
Worker eCA13	Coordinates				
Worker eCA14	Graphical Maps				
Worker eCA15	Grip map				
Worker eCA16	OO cars and maps elements	Graphical Maps			
Worker eCA17	Coordinates	Grip map			
Worker eCA18	High level components	High level components	OO cars and maps elements	Valid state check	Valid state check
Worker eCA19	OO cars and maps elements				
Worker eCA20	Grip map				

## Moving Cars

### ICplain

Worker ID	Category				
Worker pDR1	Traffic travel update loop	Simulator model	Car lane logic	Car lane logic	Car lane logic
Worker pDR2	Intersection logic	Traffic lights rules	Car lane logic		
Worker pDR3	Car lane logic				
Worker pDR4	Simulator model	Graphic map			
Worker pDR5	Intersection logic	Car travel	Car travel	Car logic	
Worker pDR6	Multi thread	Car logic	Car logic	Traffic travel update loop	Simulator model
Worker pDR7	Traffic lights rules				
Worker pDR8	Car position centric				
Worker pDR9	Car logic				
Worker pDR10	Traffic travel update loop	Traffic travel update loop	Traffic travel update loop	Traffic travel update loop	Traffic travel update loop
Worker pDR11	Intersection logic	Car logic	Car logic		
Worker pDR12	Car position centric	Car logic	Car logic		
Worker pDR13	Traffic travel update loop	Car logic	Car logic	Traffic travel update loop	Car logic
Worker pDR14	Traffic travel update loop	Intersection logic			
Worker pDR15	Traffic travel update loop				
Worker pDR16	Car travel	Graphic map			
Worker pDR17	Traffic travel update loop				
Worker pDR18	Car travel				
Worker pDR19	Car logic				

### ICexamples

Worker ID	Category		
Worker eDR1	Car logic		
Worker eDR2	Intersection logic		
Worker eDR3	Traffic lights rules		
Worker eDR4	Traffic lights rules		
Worker eDR5	Car logic		
Worker eDR6	Traffic lights rules	Traffic lights rules	
Worker eDR7	Car logic		
Worker eDR8	Traffic lights rules		
Worker eDR9	Simulator model		
Worker eDR10	Intersection logic		
Worker eDR11	Car logic	Simulator model	Intersection logic
Worker eDR12	Car position centric	Car position centric	
Worker eDR13	Car travel		
Worker eDR14	Car lane logic		
Worker eDR15	Car travel		

# Changing the Colors of Traffic Lights

## ICplain

Worker ID	Category				
Worker pTL1	Fixed timing no sensors	Traffic Rules	Loop (traffic light) no timing	Loop (traffic light) no timing	Loop (traffic light) no timing
Worker pTL2	Timing sequence no sensors	Fixed timing no sensors	Traffic Lights change based on Traffic	Traffic lights change based on time of the day	Traffic lights change based on time of the day
Worker pTL3	Timing with sensors				
Worker pTL4	Loop (traffic light)	Car lane priority			
Worker pTL5	Unison change	Timing with sensors	Loop (traffic light) no timing	Loop (traffic light) no timing	
Worker pTL6	Traffic Elements	Traffic Elements			
Worker pTL7	State check				
Worker pTL8	Fixed timing with sensors				
Worker pTL9	State check				
Worker pTL10	Traffic Lights change based on Traffic	Traffic Lights change based on Traffic			
Worker pTL11	Fixed timing no sensors	Fixed timing no sensors	Fixed timing with sensors		
Worker pTL12	Loop (traffic light)				
Worker pTL13	Fixed timing no sensors				
Worker pTL14	Car lane priority	Car lane priority			
Worker pTL15	Loop (traffic light)				
Worker pTL16	Unison change	Loop (traffic light) with sensors	Car lane priority	Car lane priority	Car lane priority
Worker pTL17	Traffic Elements	State check	Traffic Rules		
Worker pTL18	Timing with sensors	Timing sequence no sensors			

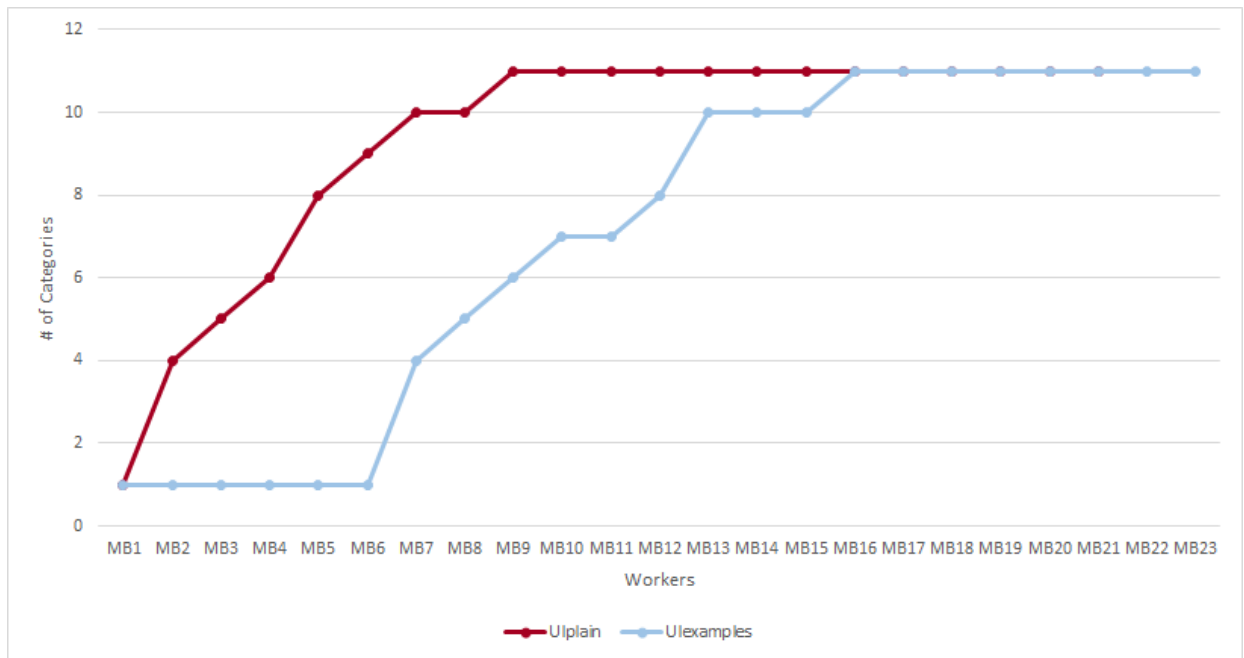
## ICexamples

Worker ID	Category				
Worker eTL1	Loop (traffic light)				
Worker eTL2	Timing with sensors				
Worker eTL3	Loop (traffic light) with sensors				
Worker eTL4	Loop (traffic light) with sensors	Loop (traffic light)	Timing sequence no sensors	Loop (traffic light) with sensors	Timing with sensors
Worker eTL5	Timing with sensors				
Worker eTL6	Loop (traffic light)	Loop (traffic light) with sensors	Traffic Lights change based on Traffic		
Worker eTL7	Loop (traffic light)				
Worker eTL8	Traffic Rules				
Worker eTL9	Timing sequence no sensors				
Worker eTL10	Timing with sensors	State check			
Worker eTL11	Timing with sensors	Class definition	Class definition		
Worker eTL12	Fixed timing with sensors	Fixed timing no sensors			
Worker eTL13	Fixed timing no sensors				
Worker eTL14	Class definition	Unison change			
Worker eTL15	Unison change				
Worker eTL16	Fixed timing no sensors	Fixed timing with sensors			
Worker eTL17	Fixed timing no sensors	Traffic lights change based on time of the day			
Worker eTL18	Car lane priority	Traffic lights change based on time of the day			

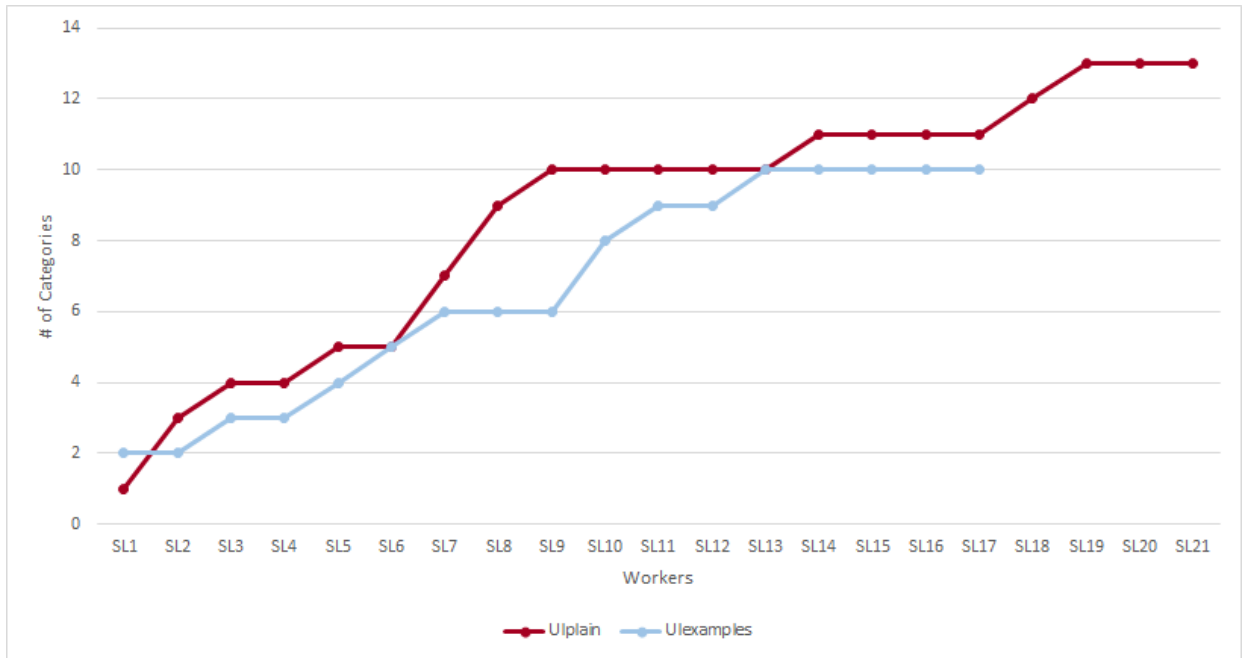
# E Cumulative Number of Unique Categories Created by Workers

## E.1 User Interface Decision Points

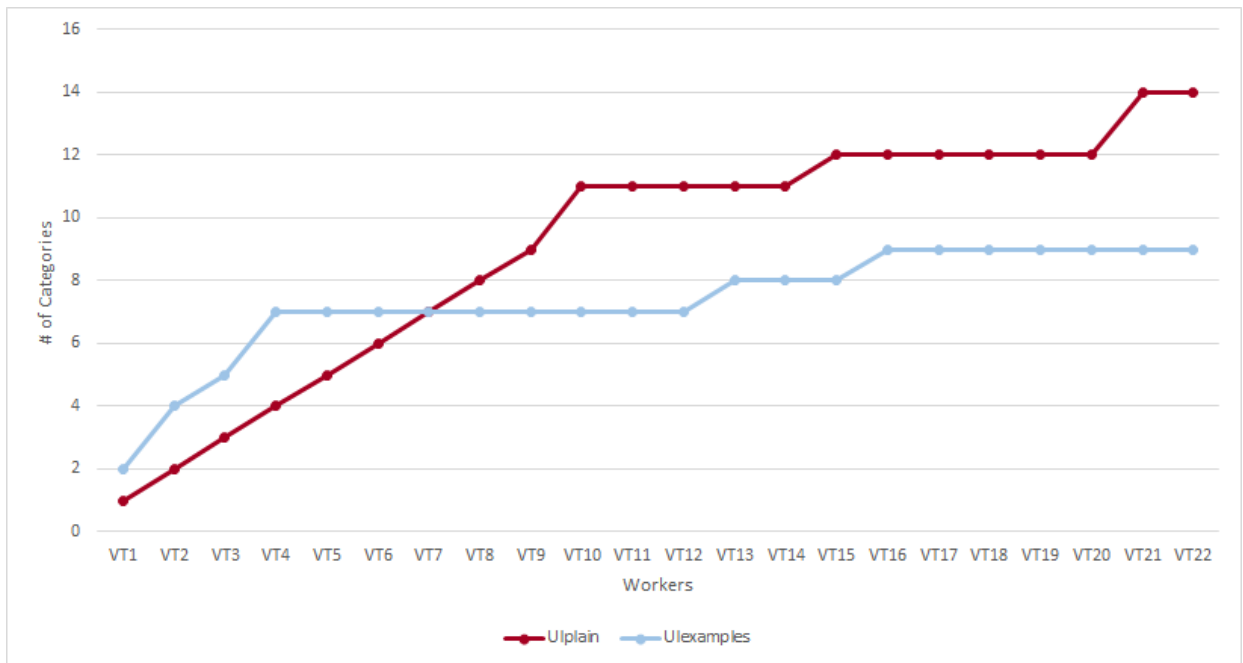
### Creating Maps



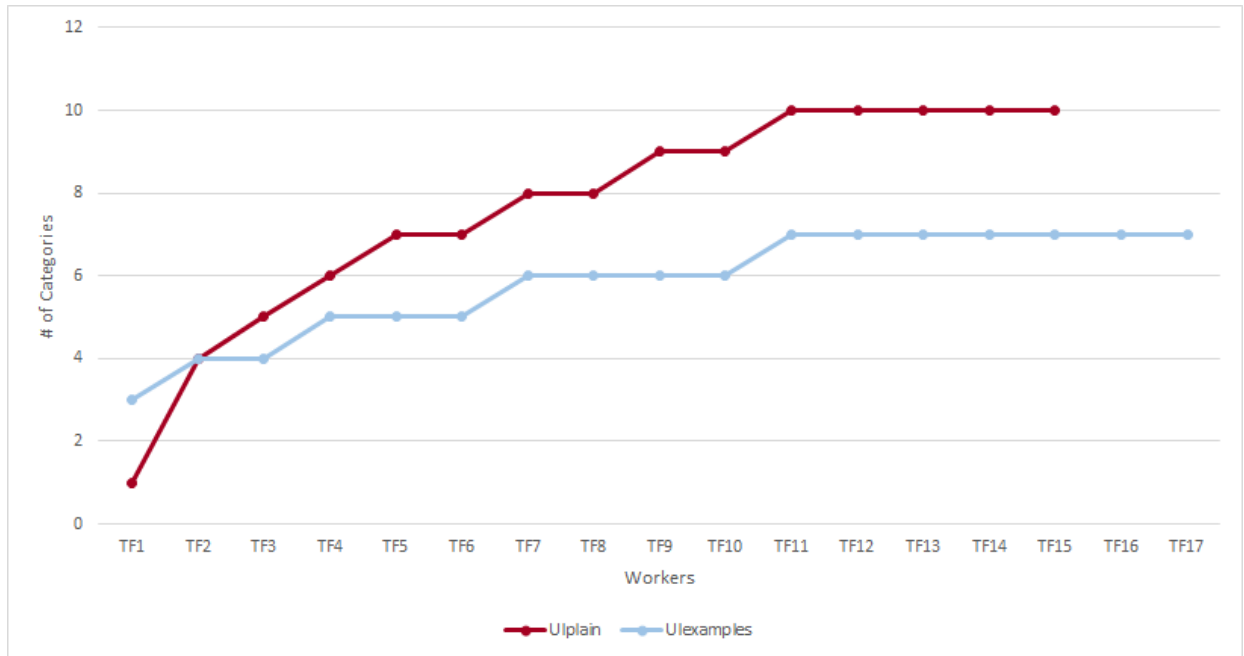
## Setting Timing of Traffic Lights



## Visualizing the State of The Simulation

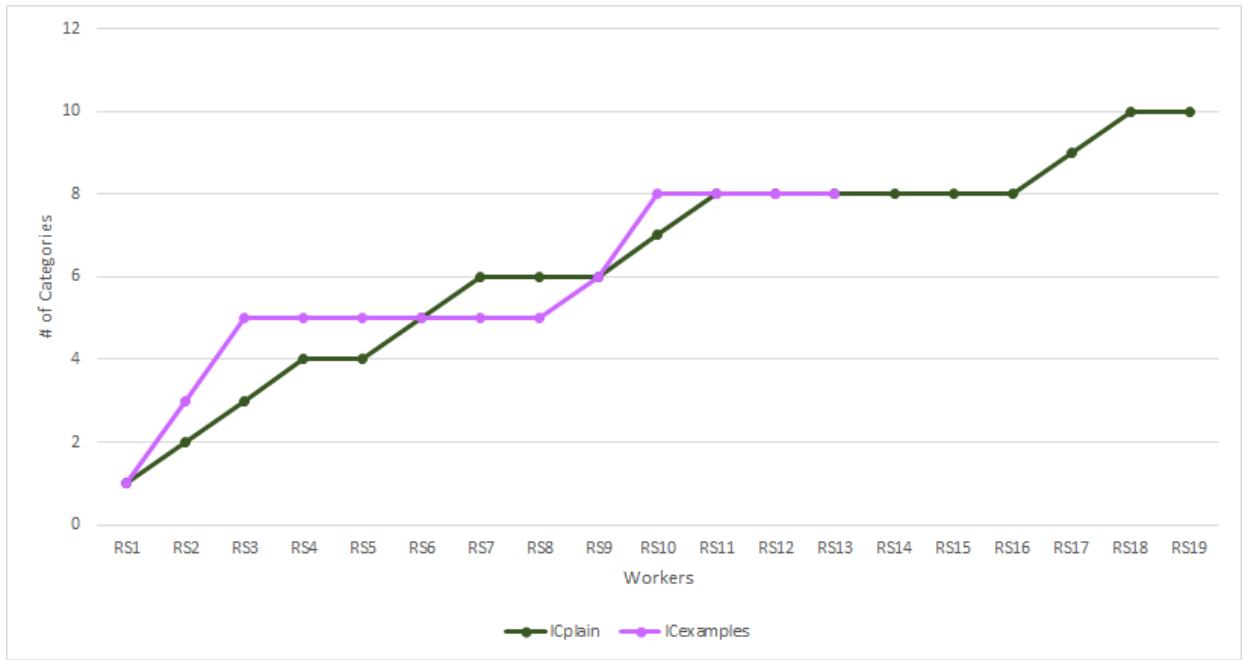


## Determining the Flow of Traffic

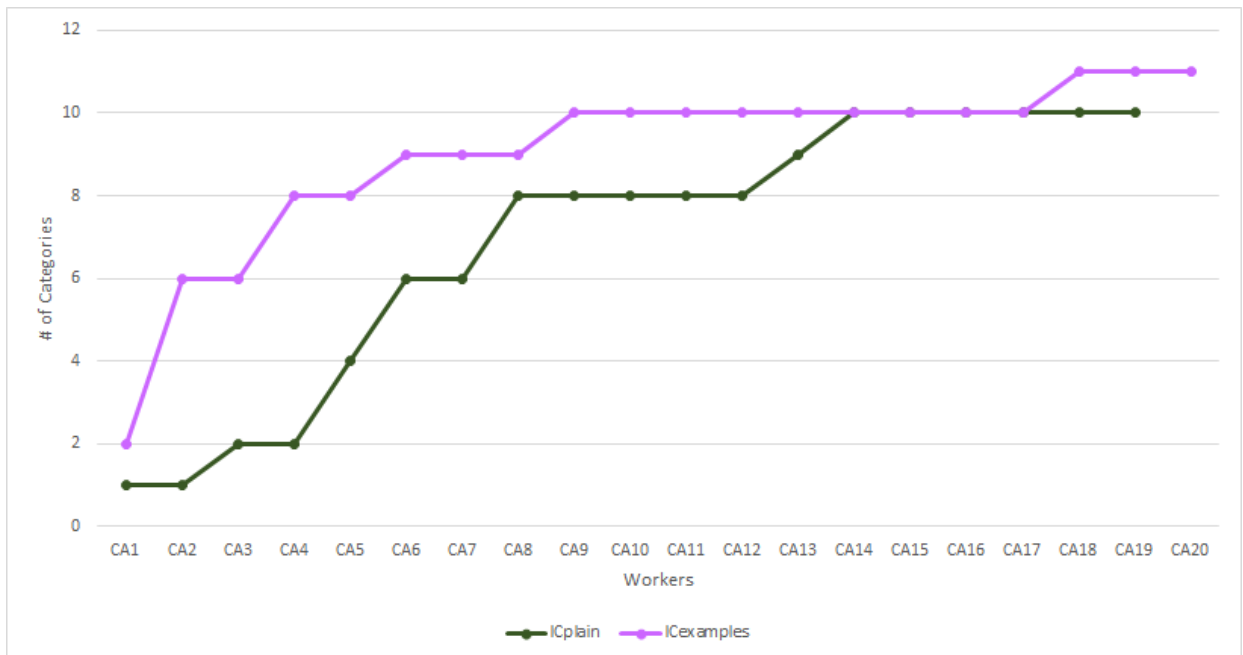


## E.2 Internal Code Decision Points

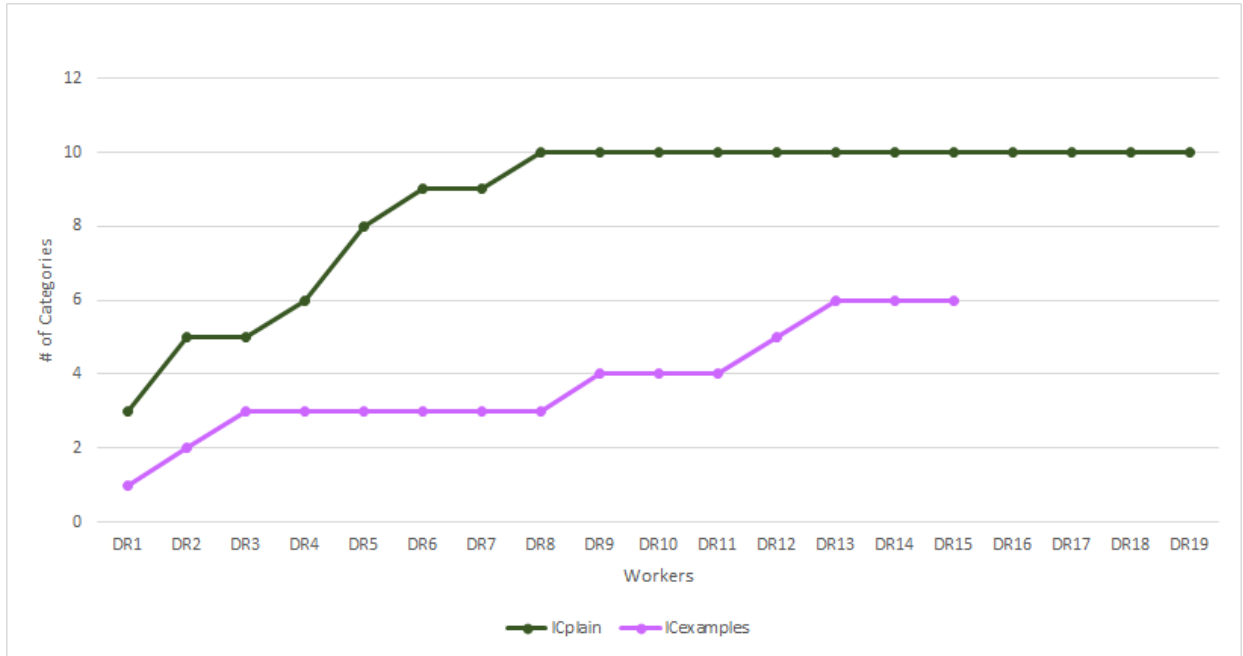
### Representing the Road System



### Representing Cars



## Moving Cars



## Changing the Colors of Traffic Lights

