Web Tasking:
An Investigation of End User Interaction for the Ideal Control Metaphor

by

Elizabeth Kittel

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2016

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Users are finding multiple ways to utilize web applications (apps) outside of their typical self-contained purposes, resulting in an increasing need to connect apps together. This connectivity can be achieved through web tasking: the integration of web services/apps to achieve a personal goal. This research investigates the end user perspective, focused on comparing user interaction with web tasking interfaces through various analytical and empirical studies. These studies were divided into three distinct parts: i) Hierarchical Task Analysis (HTA) performed on existing web tasking interfaces as a usability benchmark, ii) creation and evaluation of a new interactive prototype, WebTasker, and iii) a full scale usability study with 16 participants evaluated four web tasking interfaces by performing 4 high complexity tasks and 4 low complexity tasks on 4 different interfaces (32 distinct tasks). A significant correlation was found between the number of keystrokes and mouse clicks/scrolls and task completion time; suggesting that simple task input counts could be used as an early usability predictor in web tasking interfaces. In addition, the HTA revealed several HF issues such as freedom of user actions by examining task structures (e.g. linear path versus wide HTA structure). The usability study showed that participants had poorer performance and found it more difficult to create web tasks with higher complexity. A mental model examination of composing web tasks found that participants preferred to enter task conditions first then actions.

Web tasking is a new area in Human Computer Interaction (HCI) research and this research aimed to further develop web tasking interfaces to ultimately lead to an increase in user adoption of web tasking. The design of a new web tasking interface, WebTasker, utilized a journey line metaphor and proved to be successful in the usability study. It was recommended that it be further developed as a viable web tasking interface. Further lines or research are recommended including refining study tasks, dashboard development, and improvements to the WebTasker interface.

Acknowledgements

This work was supported by IBM's Centre for Advanced Studies (CAS). I am grateful to Joanna Ng (IBM) and Tinny Ng (IBM) for this unique opportunity to be a part of the web tasking project and for their continuous support and *inspiration*. Thank you to my readers, Professor Carolyn MacGregor and Professor Stacey Scott for their valuable comments. I would like to thank Kavita Chepovetsky for her contribution to the creation of the prototype and help with the study. Thank you to Murat Dikmen and Damla Kerestecioglu for assisting in administering the study. My thanks also goes out to Anson Ho for his input and feedback to this thesis. Special thanks to all the members of the Advanced Interface Design Lab (AIDL) for their friendship and our heart-to-hearts.

Thank you to my family for their love and encouragement throughout my program.

Finally, thank you to my supervisor, Professor Catherine Burns, for her advice, support, and guidance over the entire duration of my stay at the University of Waterloo.

Elizabeth Kittel

March 2016

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Apps	Web applications
CAD	Computer aided design
EID	Ecological Interface Design
EUP	End user programming
IFTTT	IF This Than That (Note: this is a web service)
IOT	Internet of Things
IP	Internet protocol
IT	Information Technology
HF	Human Factors
HTA	Hierarchical Task Analysis
PWT	Personalized web tasking
SUS	System Usability Scale
TaskaaS	Task as a Service

Chapter 1
Introduction

Users increasingly rely on web applications (apps) to complete a variety of everyday tasks that used to be performed offline (Castañeda et al., 2013). Moreover, users are finding multiple ways to use apps outside of their typical self-contained purposes, resulting in a need to connect apps together. As an example imagine Sally, an avid app user, is trying to keep in shape by meeting her daily fitness step goal being tracked by her new Fitbit device. She wants to immediately let her best friend know when she achieves her daily step goal by emailing her. There is a missing link here, because there is no automatic communication between her Fitbit and email app. Sally must complete some of these steps manually to achieve this task. A solution to provide the link between apps is web tasking. A web task is defined as the as the integration of web services, interactions, and sessions, from which the user benefits to achieve a personal goal (Castañeda et al., 2013). An integration across web services through apps that can be achieved with a web tasking platform.

Web tasking platforms/programs are in its infancy, being utilized by 'early adopters'. Early adopters are people who are quick to make connections between clever innovations and their personal needs, love getting an advantage over their peers or to be seen as leaders, and have a natural desire to be trend setters (Rogers, 1962). Because web tasking is in its early stages, it is important to focus on the human aspect, what the user will see and experience when using the technology, to help increase user adoption. What makes a new technology spread is whether the product or service is being reinvented to become easier, simpler, quicker, cheaper, and more advantageous (Moore, 1991). This thesis investigates what makes up the ideal control metaphor in web tasking from the end users' perspective. More specifically, this thesis looks at current web tasking interfaces through a controlled experimental design and proposes a new web tasking interface design based on human factors (HF) analyses.

Today the technology exists to provide users with constant connectivity to their apps. In 2012, 83% of Canadian households had access to the internet at home compared with 80% in 2010 (Statistics Canada, 2012). Access to the internet indicates that these people have access to web apps; approximating the consumer population that have access to apps. The statistics imply that apps are reaching a broader population. Consumers using apps may find a need for additional functionality with their use. Augmenting app functionality used to be in the control of software developers/programmers. With the rise of web tasking, consumers now have the power to achieve programmable control over their apps. Cloud provides a rich and efficient environment for software developers to develop, deploy, and run apps

for consumers; web tasking platforms contain intermediary parts to allow control to users enabling them to construct their own tasks by using resources of their own choice from across the cloud (Ng, 2015).

As the user adoption of portable and wearable technologies such as smartphones, personal health trackers, and smart watches increases – having an automated task service will become increasingly attractive (Hoy, 2015). A recent study of the Internet of Things (IOT) defines it as, "a network of networks of uniquely identifiable end points (or things) that communicate without human interaction using IP connectivity – be it locally or globally," (IDC, 2014, p.6). The main strength of the IOT idea is the high impact it will have on several aspects of everyday-life and behavior of potential users, for example, assisted living, e-health, enhanced learning are only a few instances of possible application scenarios in which the new paradigm (Atzori, 2010). The phenomena of the IOT has caused everyday objects to be continuously connected and integrated into the users' life; making way for these web tasking services to control or regulate end user tasks.

1.1 Motivation

The main driver for this research stemmed from user interaction with apps through web tasking. Existing web tasking platforms have been developed with the goal to eliminate complexity of software engineering such that average users can use their interface to create and control tasks for themselves. Task as a Service (TaskaaS) (Ng, 2015) was introduced as a new paradigm that breaks entirely away from the programming metaphor and does not require users to acquire any programming technical skill to 'program' their own tasks.

Very few studies exist on web tasking; none of which focus on ideal user interaction design. This thesis aimed to contribute to ongoing research (Ng, 2015 and Castañeda et. al, 2014) in web tasking focusing on the HF issues associated with web tasking interface use.

Several research questions have been asked by the researcher in an attempt to explore web tasking from the end user's perspective: How can users successfully engage in creating web tasks with existing web tasking interfaces? Does having multiple conditions and/or actions in a web task affect user performance in putting together a web task? What are some levels of task complexity? What can be used as a predictor of performance (e.g. task completion times)? Will users with computer programming experience perform better in some or all web tasking interfaces than those with none? This thesis aimed at answering these questions.

1.2 Thesis Overview

This thesis is organized into eight chapters:

- **Chapter 1, Introduction** – presents the motivation and research questions of the thesis.
- **Chapter 2, Background** – presents background information in the field of web tasking.
- **Chapter 3 to 5, Studies and Analyses** – This research focused on comparing existing web tasking platforms through an analytical study and an in-depth empirical study. It answered several research questions that were divided into three distinct parts (Chapter 3, 4 and 5), as seen in Figure 1.
    - **Chapter 3: What are the current human factors issues with existing web tasking platforms?** Analytical HF analysis through Hierarchical Task Analysis (HTA) (Annett and Stanton, 2000) was performed on three web tasking interfaces: IFTTT[1], Zapier[2], and Scribble. This analysis can reveal issues such as the steps that need to be done to complete a certain task and areas for potential human errors, and identify interface element or convention improvements. However, the HTA cannot speak to issues such as identifying the relationship between user actions and cognitive processes (e.g. interaction mental models of web tasking).
    - **Chapter 4: How do people perceive composing web tasks? What makes up the ideal user control metaphor for web tasking?** A literature review was conducted about web tasking. A pilot usability study was conducted on existing web tasking interfaces and the results were used as input into the researcher's design of a unique web tasking interface, called WebTasker. An interactive prototype of WebTasker was created using Axure[3] software. In addition to the pilot study, a simple heuristic review was implemented to generate recommendations of design features for a new web tasking interface.
    - **Chapter 5: How does the new design, WebTasker, compare to the existing web tasking interfaces?** A full scale usability study was conducted with 16 participants evaluating four web tasking interfaces for approximately 2 hours per session. Each participant performed 4 high complexity tasks and 4 low complexity tasks on 4 different interfaces (32 distinct tasks). Metrics to study end user interaction included: task timings, errors, and ratings from a System Usability Scale (SUS) questionnaire (Brooke, 1996).

[1] www.ifttt.com
[2] www.zapier.com
[3] www.axure.com

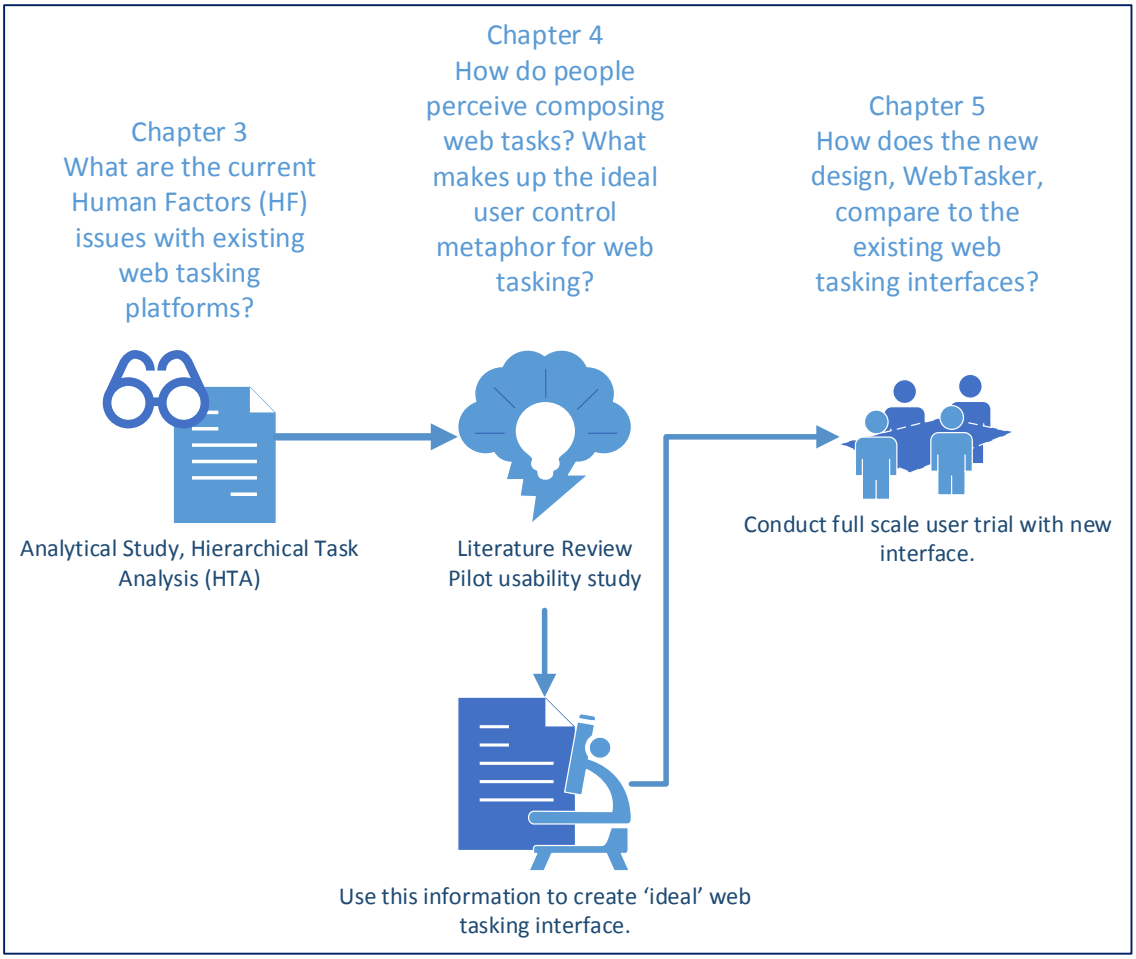


Figure 1: Thesis Outline

- **Chapter 6, Discussion** – highlights the prominent findings from the studies and analyses and presents them in terms of the research questions.
- **Chapter 7, Future Research** – presents recommendations for future work.
- **Chapter 8, Conclusions** – discusses and summarizes significant findings.

Chapter 2
Background

This chapter provides an overview of relevant areas or research to understand web taking. It describes the literature review method and results conducted as part of this thesis.

2.1 Related Research

There has been considerable work in empowering end users to be able to write their own programs, and as a result, users are indeed doing so (Burnett et. al, 2006). Recent technology can enable end users to be contributors rather than just consumers of information on the web. The trend is now moving from content and personalization to functionality, in the direction of user-generated web services (Costabile et. al, 2010). Two related areas of research to web tasking are end user programming (EUP) and mashups.

2.1.1 End user programming

An EUP task is typically a consequence of a user's perception of a lack of needed functionality and this can only happen if the user is convinced that s/he fully understands the application as it is presented by the user interface language (de Sousa et al., 2001). End user programming usually involves some type of user scripting and can actually look like a programming language. Real-world examples of end-user programming environments include (Prabhakararao et al., 2003):

- educational simulation builders,
- web authoring systems,
- multimedia authoring systems,
- e-mail filtering rule systems,
- CAD systems,
- and some spreadsheet functions.

Using such systems, end users create software that could be in forms of educational simulations or dynamic e-business web applications (Burnett et. al, 2006).

Another example of EUP can be seen in the gaming domain. Many computer games are now built with the intention that they will be modified by enthusiastic users (Robinson, 2009). Passionate users of these games are actually participating in the design of the game; perhaps feeling a sense of ownership in doing so.

Another form of EUP is visual programming. Scratch[4] was launched in 2007 by MIT as an approach to programming that would appeal to people who hadn't previously imagined themselves as programmers – people of all ages, backgrounds, and interests (Resnick et. al, 2009). Scratch is a visual programming tool to create interactive stories, games, animations, and simulations, that can be shared online (Figure 2). Although using Scratch may appear more like playing a game, the control metaphor here is programming.

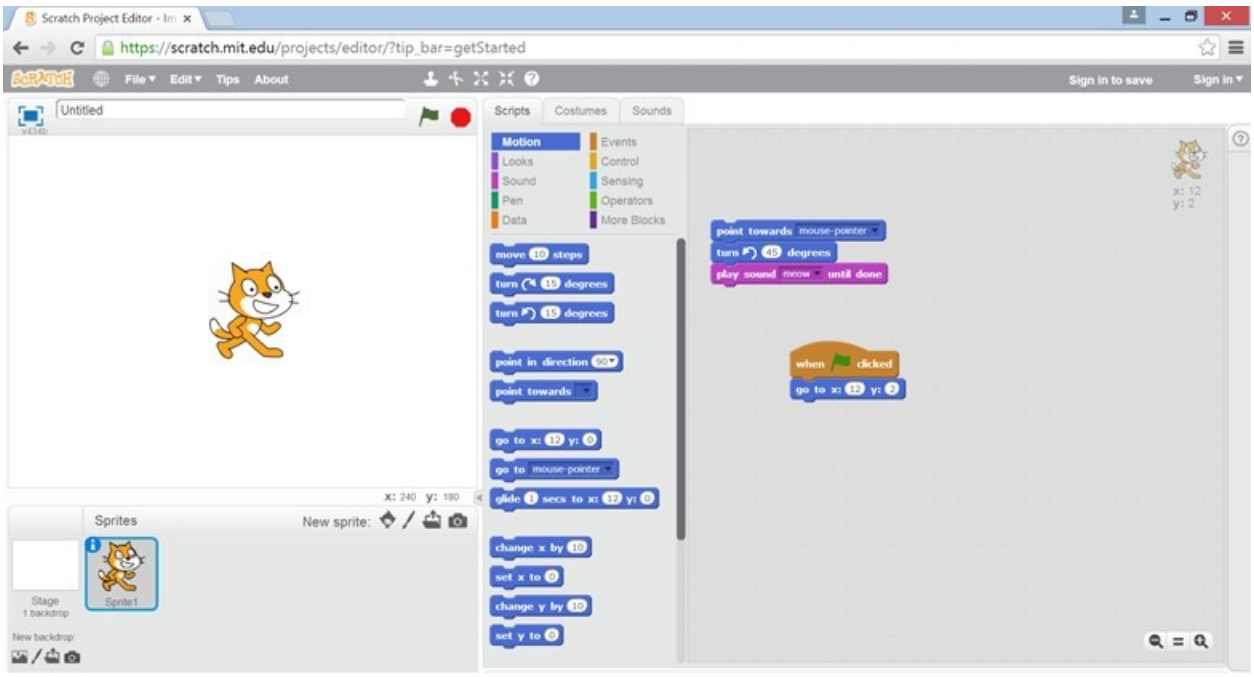


Figure 2: Scratch screenshot example

EUP requires some knowledge of programming. In tasking, it does not require the user to have any programming skills (Ng et. al, 2014).

2.1.2 Mashups

Mashup has been defined as a combination of pre-existing, integrated units of technology, glued together to achieve new functionality, as opposed to creating that functionality from the scratch (Harmann et. al, 2008). In theory, a mashup sounds like a solution that will meet user's goals (in terms of tasking). However, it was found that existing mashup tools are too technical for end-users and, as a consequence, end-users are not able to: (i) understand what exactly they can do with the tool and (ii) how to do it (Stefano et. al, 2014). Mashup tools with open APIs have been developed for users with some level of

[4] www.scratch.mit.edu

programming skills but a majority of Web service users are not skilled or interested in such efforts (Mattila et. al, 2011).

2.2 Web browsing versus web tasking

Whatever web or mobile app software developers do not provide, web users have to do for themselves manually, such as doing their own work around, writing information from one site and typing written down information into a hypermedia form of another site as a manual form of integration of the web silos (Ng et. al, 2014). One may ask the question, how can distributed resources from siloed, disjoint server systems be put in the hands of web users such that users can interact to perform tasks, to delegate and automate tasks, without any programming requirement? (Ng et al., 2014). This can be achieved through web tasking.

Ng and Lau (2013) outline the difference between web browsing and web tasking (see Table 1). They used the web browsing paradigm as a starting point to develop a web tasking platform. That platform is IBM's Scribble. Scribble has found a way to fill the need to automate tasks on the web. Scribble enables users to choose from resources from across the web for their tasks and then gives them control to specify task intents, such that they are executed automatically (Ng et. al, 2014).

Table 1: How web browsing is different from web tasking (Ng and Lau, 2013)

	Web Browsing	Web Tasking
Purpose	Information retrieval, search	Action, transaction, progress towards users goals
Resource characteristics	Resources are read-only	Resources have operations and behavior
Application state transition	Unpredictable prior execution	More predictable prior to execution
Representation	Resource state	Action state or actionable resource
Consequence	Read only, no side effect on server side components	Update and write, side effects on server side components
Usage characteristics	Ad hoc, transient, not intended for repetition	Intentional, transactional, possible for repetition

2.2.1 Task as a Service

Ng (2015) explored how the complexity of software engineering can be abstracted into simplified controls that the average users can use to control task for themselves, independent of software engineers and without any cognizance of software engineering. Task as a Service (TaskaaS) is presented in her research and a tasking conceptual model was created (Figure 3). As Ng explained, tasking is a new paradigm that breaks entirely away from the programming metaphor and does not require users to acquire any programming technical skill.

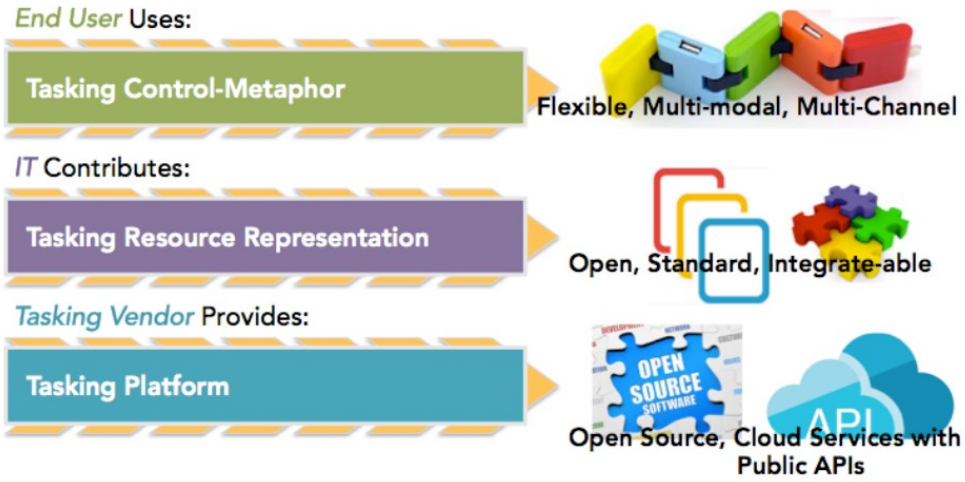


Figure 3: An Open Tasking Conceptual Model (Ng, 2015)

As seen in Figure 3, the bottom layer there is a 'Tasking Platform' provided by a tasking vendor or service provider. The middle layer is a 'Tasking Resource Representation' which is a resource model defined by the tasking vendor to prescribe how IT can take their entities from current Apps and transform them into tasking resources for the tasking platform. The top layer is the 'Tasking Control-Metaphor', which is the focus of this thesis. The control metaphor is the interface that users use compose web tasks. In essence, users will use this metaphor to maneuver tasking resources (middle layer) from the tasking platform to create their own personalized tasks without being aware of the software engineering behind it.

2.3 Literature Review

Keywords and terms related to web tasking were used to search three academic databases.

2.3.1 Keywords and Approach

The following databases were used for this literature review:

- Scopus
- ACM Digital Library

- IEEE Xplore Digital Library.

The keywords/terms in Table 2 were used in combination to search the databases identified above. If an unmanageable number of hits resulted from a primary keyword/term search, then a secondary keyword was added to focus the results. For example, 'End user' was entered in the search, then 'tasking'.

Table 2: Literature Review Search Terms

Primary Key Words/Terms
Web tasking
End user
Task
IFTTT
Zapier
Internet of Things
Secondary Key Words/Terms
Tasking
Technology
Web

2.3.2 Literature Review Results

The initial search conducted was to define web tasking. The approach taken to answer, "what is web tasking" has been broken down into a few components:

- who are the users,
- what is end user tasking,
- what are web tasks?

The subsequent sub-sections aimed to answer these questions to formulate a comprehensive answer to what is web tasking.

2.3.2.1 Who is the end user?

The end user simply stated is the person who will use the product, system, or website. When designing these products, systems, and websites, designers have to make an effort to get to know the user. In the context of design, designers should not consider themselves users of their own product. In a study from "The Management of End-User Computing: Status and Directions" end user computing was defined as, "the adoption and use of information technology by personnel outside the information systems

department to DEVELOP software applications in support of organizational tasks," (Branchau, 1993, 439). In other words, non-technical people completing technical tasks.

2.3.2.2 What is end user tasking?

The types of end user tasks range greatly in complexity. Traditional examples include programming by demonstration such as via macro-recording, visual programming language, and scripting. A common end user task example is transferring data across applications (i.e. copy and paste) (Stolee et al., 2009). Some other examples include installation, setup, and customization of software and applications.

End user tasking has evolved and changed in recent years due to phenomena of social networking and new technology available (e.g. wearables and IOT). Everyday objects now have the capability to be continuously connected and integrated into the end users' life. This creates a new need to manage web tasks and one way to control end user tasks is through web tasking.

2.3.2.3 What are web tasks?

The goal of any web tasking platform should be to automate eligible tasks without requiring any formal programming by the user. A web task is defined as the as the integration of web services, interactions and sessions, from which the user benefits to achieve a personal goal; if the user classifies the task as complex, s/he must decompose it manually into smaller and simpler tasks logically sequenced (Castañeda, et. al, 2013).

Castañeda, et al. (2014) clearly defines Personalized Web-Tasking (PWT) as the automation of repetitive and mundane web interactions that, together with the exploitation of personal context (e.g., information from personal profiles, social relationships, and historical web interactions), seeks to optimize user experiences by assisting people in the fulfillment of personal goals using internet technologies.

Web tasks can be used across web applications such as: social media (Facebook, Instagram, Twitter, etc.), email, weather, shopping, banking, stock market, and location based services. Web tasking involves distributed resources that have behaviours and actions. Some of these actions are transactional (i.e. information exchange) in their characteristics. It can be used in the healthcare domain, such as monitoring tasks (e.g. blood pressure, blood sugar levels, and fitness measures). It can also be used from businesses for ordering inventory, customer support, event management, marketing, sales, and project management. Lastly, it can be used in IOT, for example in smart home monitoring. Below are specific web tasking examples in a few domains:

- Tweet my Facebook status updates.

- When my mom's blood pressure is high, text me.
- When inventory is running low, order X amount of stock from X supplier via email.
- When no one is home, turn the thermostat down to 20°C.

Castaneda et al. (2013) examined PWT by automating personal web tasks, driven by user needs, matters of concerns, and personal context. As an important concern in PWT they examined task simplification and characterized several challenges for task simplification. Task complexity measures are required to determine whether a task is candidate to simplification. A set of attributes that are relevant to the complexity measure of a task (Castaneda et al., 2013):

(1) Number of web interactions,

(2) Available knowledge about the task,

(3) Information available about previous simplifications,

(4) Number of inputs that can be inferred and number of inputs that require user's intervention, availability of resources including web services and context sources, and level of dynamics of the relevant context information (i.e., whether the relevant context dimensions static (e.g., the user's birthday) or change frequently (e.g., the user's location, preferences).

2.3.2.4 Current Tasking Platforms

There are several companies that provide services close to web tasking. Two prominent commercially available ones are IFTTT and Zapier. These two platforms utilize a trigger-action programming (i.e. the user must select a trigger that will cause an action to occur). These programs are also referred to as "web automation" applications. They are triggered based on changes to other web services (such as bank apps, email apps, and social media apps).

2.3.2.4.1 Terminology

Different terminology was used in each interface to describe the components of a web task. Table 3 is a summary of the terms for condition (what needs to be satisfied or met of which action is dependent), action (what will be done when condition is met), and completed task (condition(s) and action(s) put together) for each web tasking platform/interface examined in this thesis.

Table 3: Web Tasking Interface Terminology

Interface	Term for Condition Component	Term for Action Component	Term for completed task
IFTTT	Trigger channel	Action channel	Recipe
Scribble	Condition BOTBit	Action BOTBit	Scribble
Zapier	Trigger app	Action app	Zap
WebTasker	Condition	Action	Task

2.3.2.4.2 IFTTT

If This Then That (IFTTT[5]) is the best known automated task service where users can combine more than 140 different ‘‘channels’’ and ‘‘triggers’’ to create ‘‘recipes’’ that accomplish a specific task (Hoy, 2015). Users can upload their recipes to share with the community, where the recipes are easily searchable and categorized by theme (e.g. recipes for music lovers, for your garden, for parenting, for the online shopper, for following the news, etc.). There is currently no fee for this service. IFTTT is also quickly moving into the IOT arena as there are channels for the Nest thermostat, Fitbit health trackers, and Hue lightbulbs have recently been added, allowing users to create rules that cross over into the physical world (Hoy, 2015). IFTTT allows only one trigger and one action per recipe. A few examples can be seen on their “dashboard” in Figure 4.

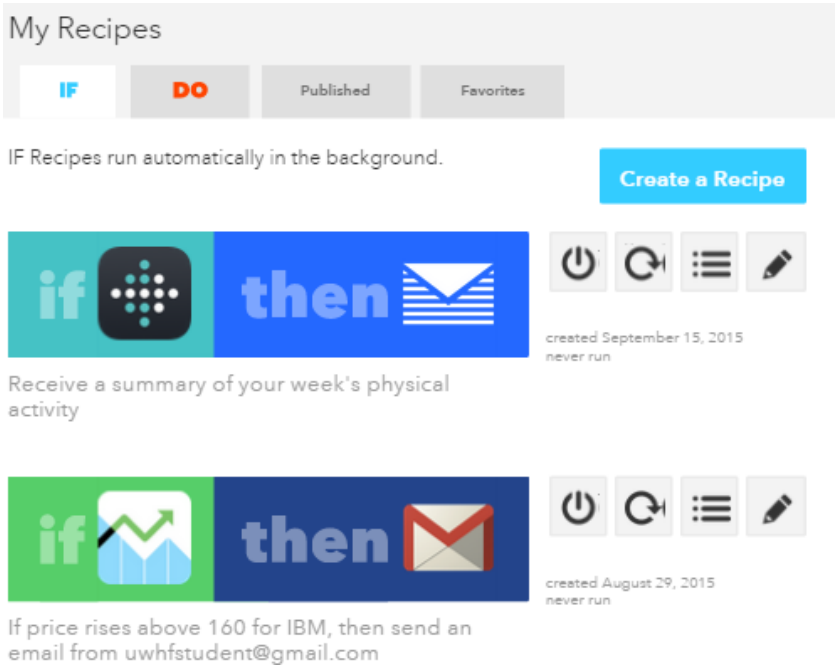


Figure 4: IFTTT screenshot examples

[5] www.ifttt.com

2.3.2.4.3 Zapier

Zapier[6] focuses mostly on enterprise solutions. It offers users more than 300 apps that can be tied together in many different ways; users can choose from categories of apps including project management, help desk, and sales (Hoy, 2015). Zapier also uses trigger-action programming and a "recipe" in Zapier is called a "Zap". Zapier is free for up to five zaps and then offers tiered packages. Zapier allows only one trigger and one action per zap. An example of what the Zapier interface is like is shown in Figure 5.

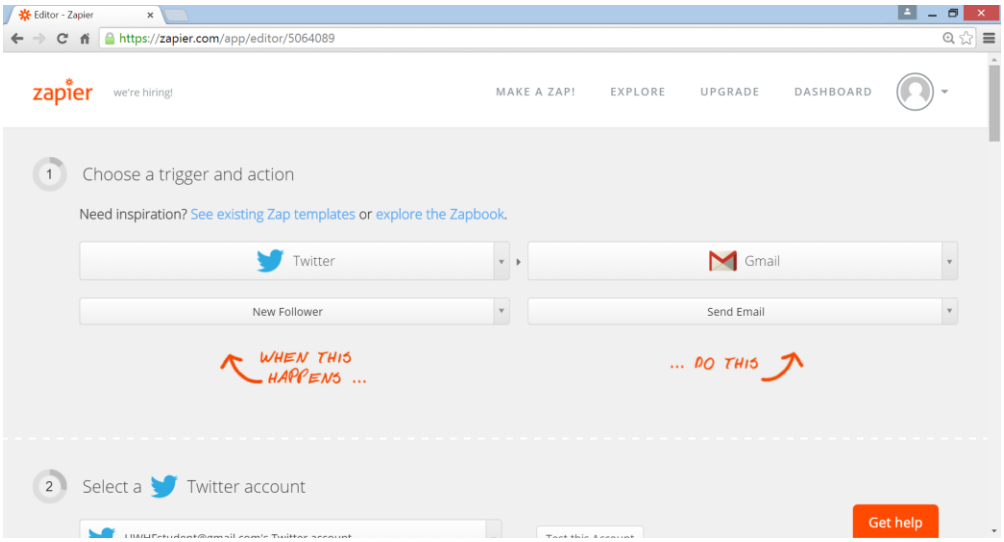


Figure 5: Zapier screenshot example

2.3.2.4.4 Scribble

Scribble by IBM is a web tasking platform that also uses trigger-action programming. Scribble added a third dimension, schedules. Scribble is currently in beta version and has not yet been released to the public. An example of the Scribble interface (as of Sept 2015) can be seen in Figure 6. The researcher was granted special access to Scribble, as it is not commercially available. Scribble uses a jigsaw puzzle control metaphor. Each piece of the jigsaw puzzle is colour coded to represent a component of the task. Red represents an action, yellow represents a condition, and blue represents the schedule. The action of the task is placed on an "action track" and conditions are stacked upon each action piece.

[6] www.zapier.com

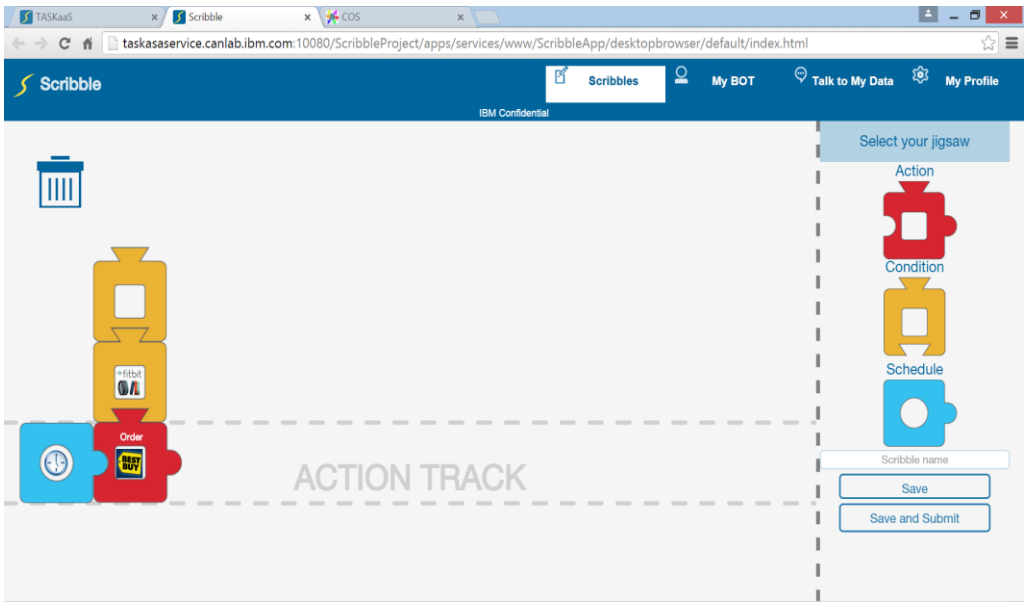


Figure 6: Scribble screenshot example

2.3.2.5 Technology Adoption

As Norman (1988) said, technology development happens when users want more functionality. This is analogous to what is happening with apps. The use of single apps are no longer meeting users' needs and a combination of use across apps (more functionality) is what would help achieve their goals. The use of web tasking or web automation is in the early adopter phase. In particular, there are two technology adoption factors (Robinson, 2009) that were noted by the researcher for the design of the new web tasking interface:

- **Compatibility with existing values and practices** – Incompatibility with existing values or practices will not be adopted.
- **Simplicity and ease of use** – The easier it is to use it more rapidly it will be adopted versus innovations that require the user to develop new skills and understandings.

2.3.2.6 Tasks for Study

There is a tradeoff between a simple interface and increases with functionality – the more functionality a product has, is usually proportional to an increase from a simple interface to a complex one. The same principle applies to task complexity and learnability. The more complex a task is, usually the more complicated the interface is. The same technology that simplifies life by providing more functions in each device also complicates life by making the device harder to learn, harder to use (Norman, 1988).

Chapter 3
Analytical Study of Web Tasking Interfaces

3.1 Background

In support of the experimental study and as input into the design of a new web tasking interface, it was decided that the steps involved in the entry of a task could be identified using a hierarchical task analysis (HTA) approach. HTA involves describing the activities under analysis in terms of a hierarchy of goals, sub-goals, operations, and plans; with an end result of an exhaustive description of task activity (Stanton, 2013).

Annett and Stanton (2000) describes HTA as a general program approach, helping analyst understand the problem and the domain. In the past, HTA has been used for interface evaluation (Kirwan and Ainsworth, 1992; Wilson and Corlett, 1995; Stanton, 1996; Shepherd, 2001). HTA has been put into many different uses including interface evaluation, training, interface design, and work organization.

3.1.1 Aim

The aim of the HTA completed in this thesis was to evaluate the discrete steps and screens required by each of the web tasking entry tasks required by each of the three web tasking interfaces (IFTTT, Zapier, and Scribble). The goals of this study included:

- development of a hierarchy of actions/functions
- generalized structure (e.g. information or link available on a given page)
- identification of trigger and action entry structure and sequence
- the number of steps in a given task
- item selections needed to complete a given task, and
- interface components (buttons, drop down menus, graphics, and icons).

3.2 Method

The task hierarchies and structures were determined by the researcher. These results were then converted into flow charts through Microsoft Visio Professional 2013 to show the generalized structures of each web tasking interface. A general HTA was created for Scribble, IFTTT, and Zapier for two level of complexity tasks.

3.3 Tasks

These tasks and complexity were defined as follows:

- High level complexity tasks
    - Creating a web task (i.e. Scribble, Recipe, or Zap) from scratch without a template or starting from a 'published' Scribble, Recipe, or Zap.
- Low level complexity tasks
    - Involved selecting a previous Scribble, Recipe, or Zap either from a search or selecting it from a category.

In this study, six task breakdown flows across three different web tasking interfaces were created (Figure 7 to Figure 12). The first step in conducting an HTA is to clearly define the tasks under analysis (Stanton, 2013). Table 4 shows the tasks used in the HTA. The tasks were representative of interactions with web tasking platforms and many were given as example tasks provided by their creators (available on their websites).

Table 4: HTA Tasks

	IFTTT	
#	**High Complexity**	**Low Complexity**
1	If a new step count is logged in Fitbit then send a new email from Gmail.	Select: Tweet when you achieve your daily step goal in Fitbit.
2	If iPad price changes at Best Buy then a post a tweet.	Select: If it's going to rain tomorrow, send me an email from Gmail.
3	If a Facebook new status message is posted by you then create a text post in Tumblr.	Select: Share new links you post on Facebook to Twitter.
4	If the IBM stock price rises above $160, then send an email to ekittel@uwaterloo.com from Gmail.	Select: If Google Stocks price drops, then tweet stock is dropping.
	Zapier	
#	**High Complexity**	**Low Complexity**
1	When you post a new tweet post it to your Facebook timeline.	Select: Send me an email monthly on a specific day of the month.
2	Send an email via Gmail for new tweets from a @UWHFstudentgirl.	Select: Tweet new RSS feed item.
3	When you star an email in Gmail post it to your Facebook timeline.	Select: Posts the Day 1 forecast from the Storm Prediction Center to my Facebook page each morning at 9 am.
4	If it's going to rain tomorrow, send me an email from Gmail.	Select: Get an email for Zapier updates.

	Scribble	
#	**High Complexity**	**Low Complexity**
1	If fit bit weight is met AND University of Waterloo GPA is met, then buy me an iwatch from Best Buy and notify me. Set this Scribble to run every day.	Select: Find movies listings.
2	If IBM stock is > $175 AND exchange rate is met then notify me. Set this scribble to run biweekly.	Select: Take the next bus to the airport if the weather is clear.
3	If my RBC bank account balance is less than $1000 then notify me. Set this Scribble to run every Friday at 10:00 a.m.	Select: Buy twitter stock check.
4	If my RBC bank account balance is less than $2000 then buy an iPad from Best Buy. Set this Scribble to run on the 28th day of every month.	Select: Book a vacation.

3.4 Results

The results of the HTA study are presented the subsequent sections.

3.4.1 Hierarchy of Functions

The flow charts in Figure 7 to Figure 12 are organized in a hierarchy of tasks where each box represents a task currently available to the user. Each level contains all of the items available to the user at each point in the task. Each level can be considered as one step or screen needed to complete a desired goal for the high level and low level entry tasks. The text in green indicates the action needed to complete the step. For example, the user must <Click Enter button> or < Continue> to complete the step.

The HTA follows the hierarchy of functions necessary to complete both levels of complexity of the two types of entry tasks (based on the tasks identified in Table 4). Other functions of the web tasking systems are included where they are encountered (indicated in grey), however any functions not needed to complete the tasks of interest are not broken down any further. Generic HTA structures for each program are discussed in the Section 3.4.2.

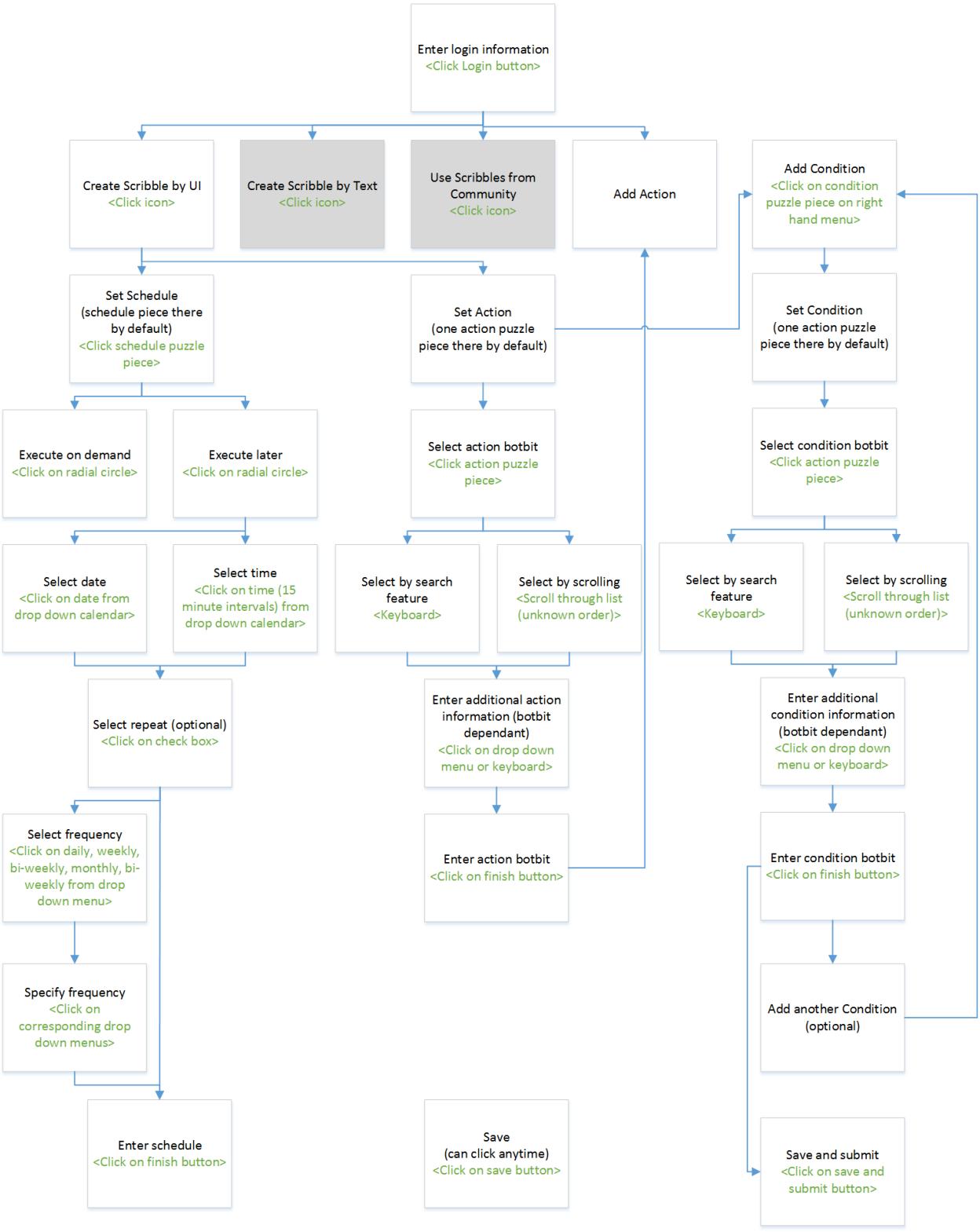


Figure 7: Scribble High Complexity Task Breakdown

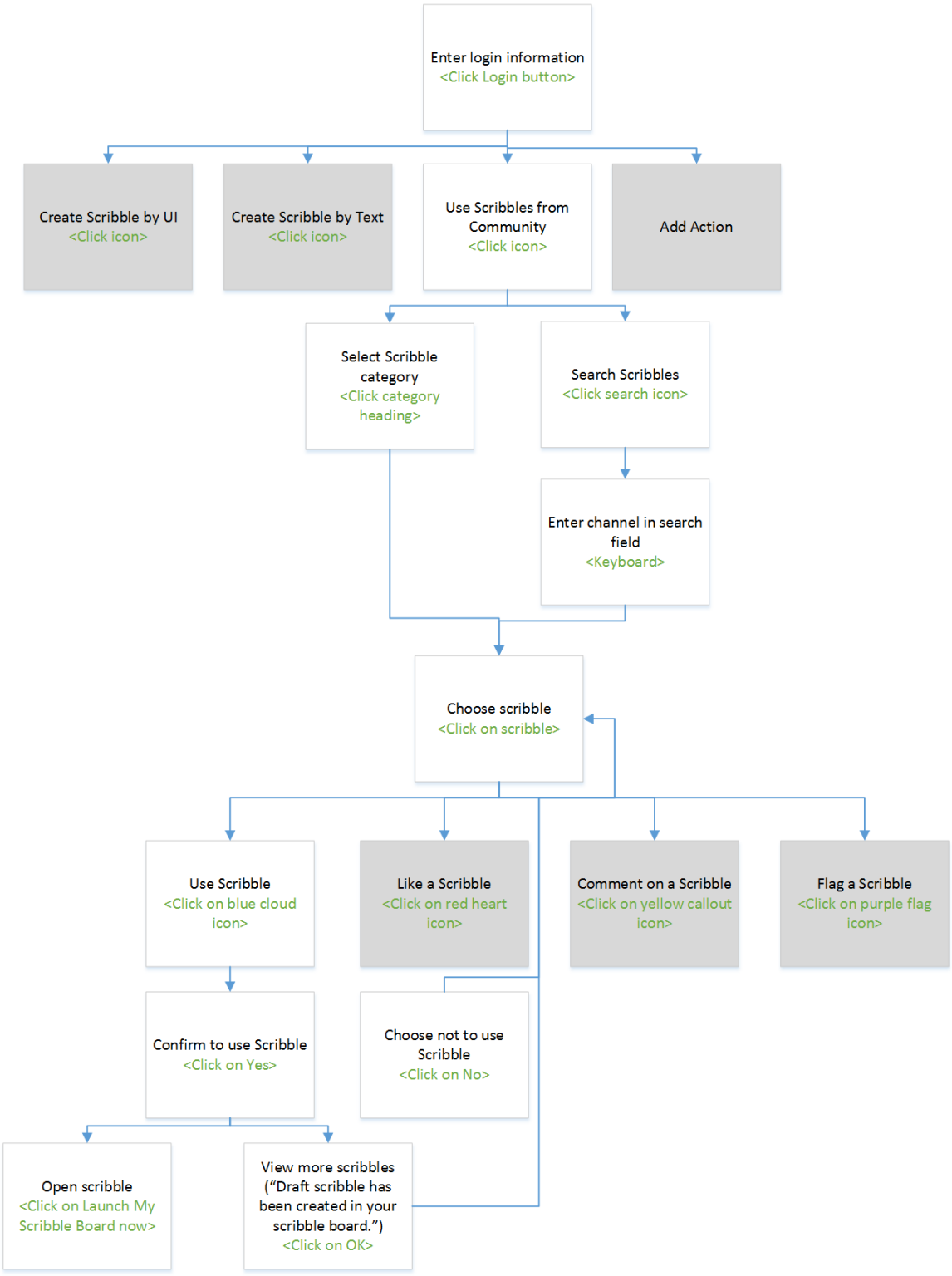


Figure 8: Scribble Low Complexity Task Breakdown

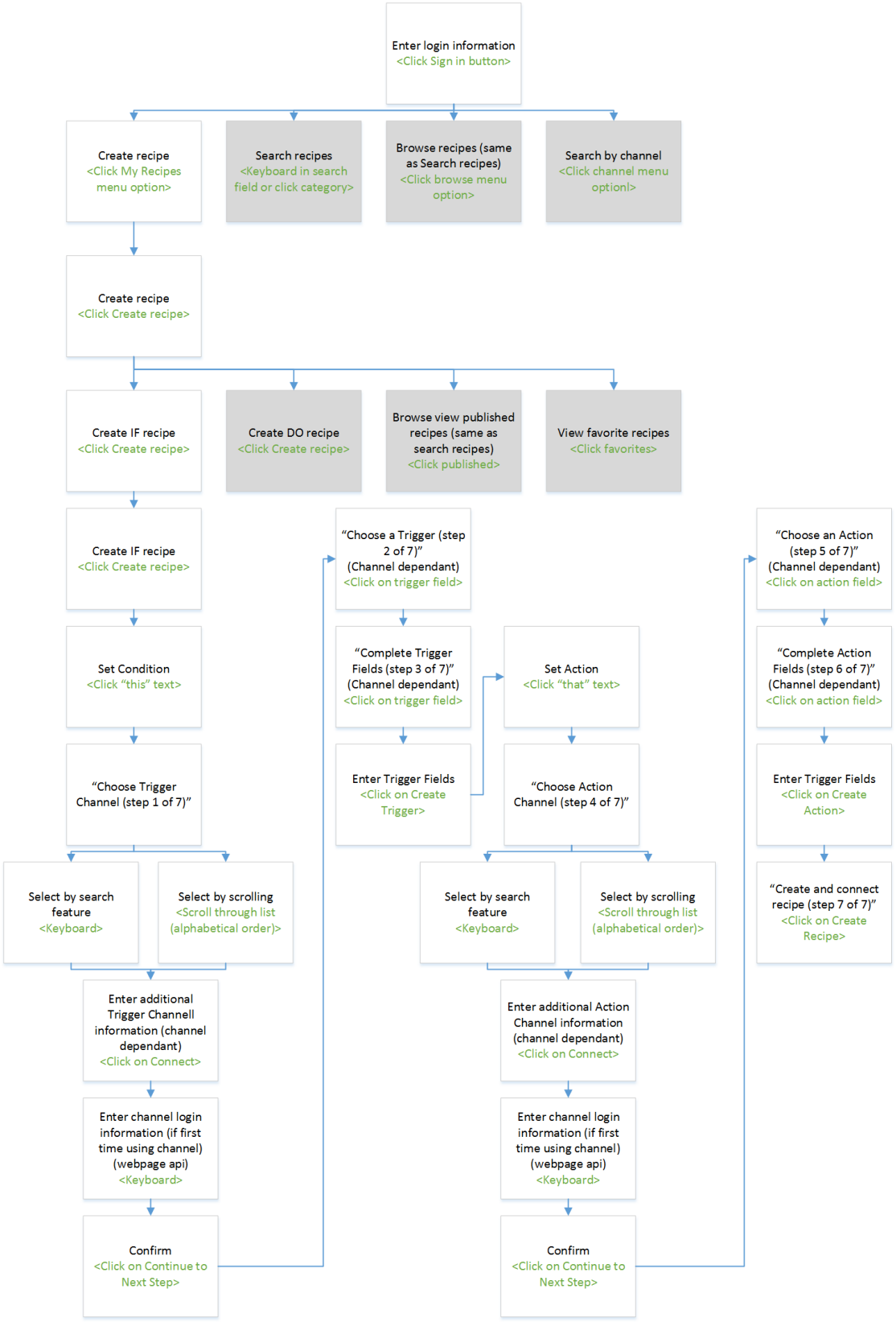


Figure 9: IFTTT High Complexity Task Breakdown

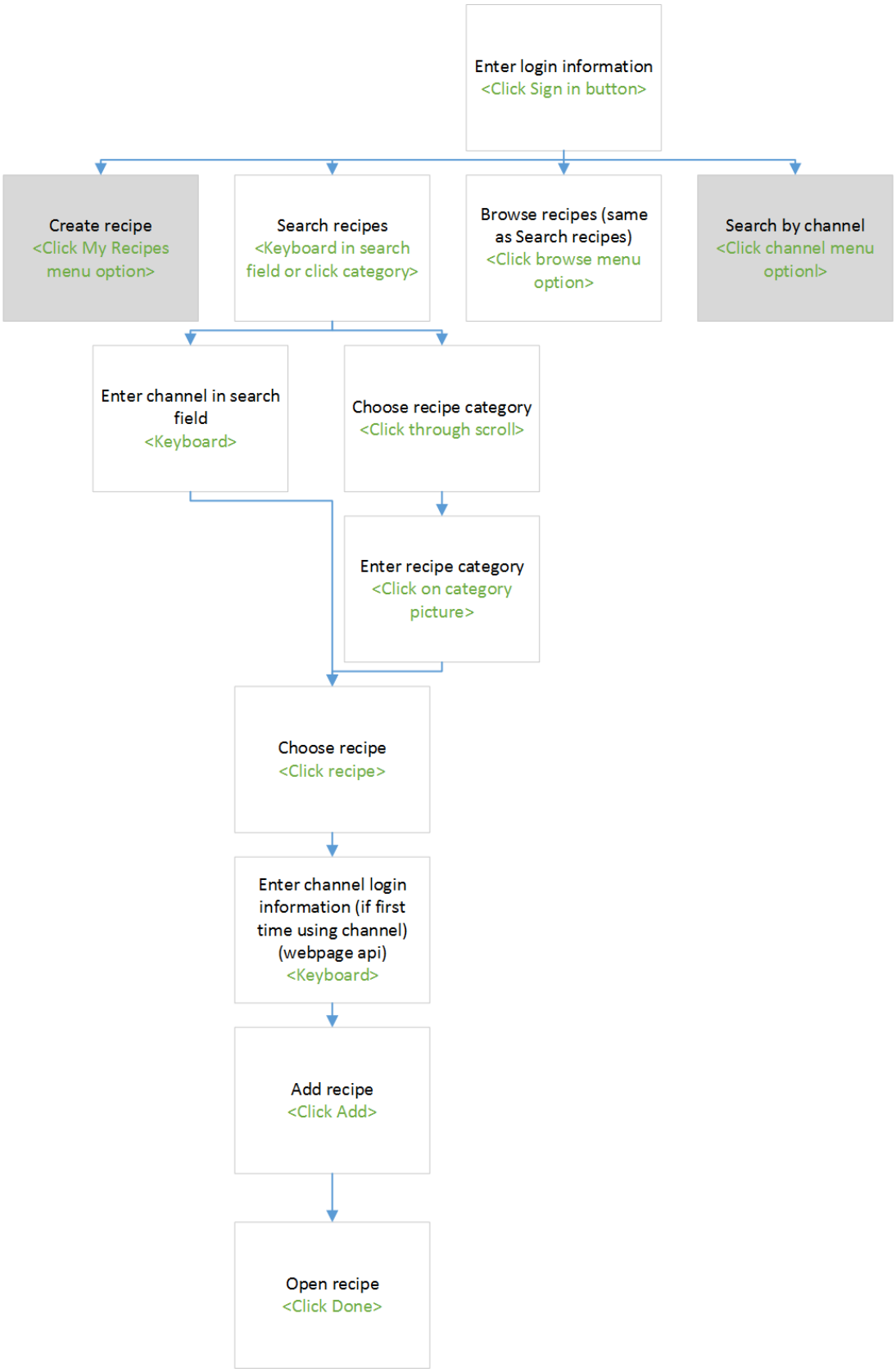


Figure 10: IFTTT Low Complexity Task Breakdown

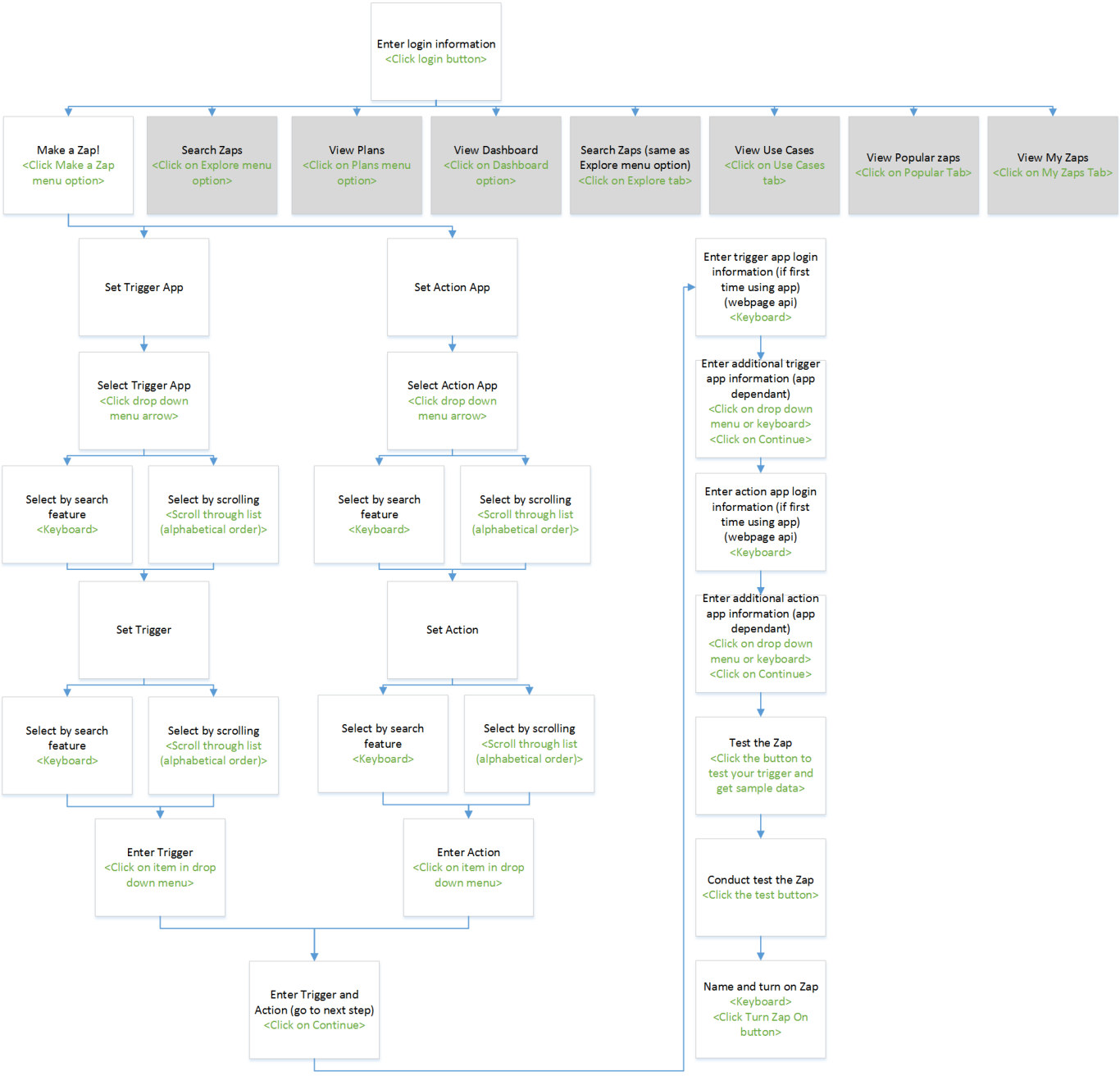


Figure 11: Zapier High Complexity Task Breakdown

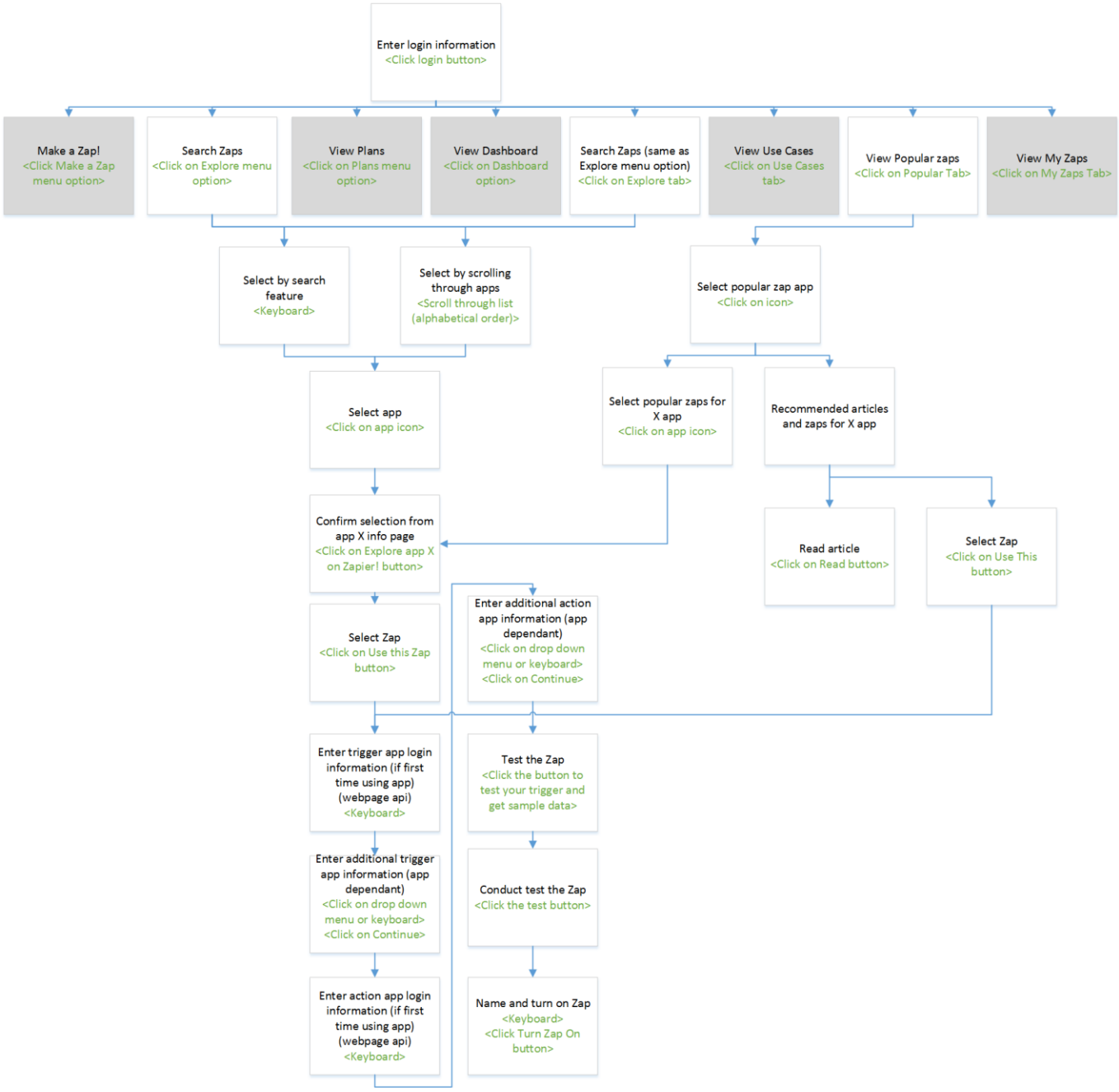


Figure 12: Zapier Low Complexity Task Breakdown

3.4.2 Task Breakdown

The aim of the HTA study was to evaluate the discrete steps and screens required by each of the three action-trigger and web tasking entry tasks. The goals of this study included: development of a hierarchy of functions and generalized structure of each, identification of trigger and action entry structure and sequence/order, the number of steps in a given task, item selections needed to complete a given task, interface components. The flow charts (Figure 7 to Figure 12) were created to aid in accomplishing these goals. They revealed variable structures in depth and breadth, representing various task sequences. The HTA provided detailed information needed to compare tasks as well as showing differences between high and low complexity task levels based on the HTA structures. It was hypothesized that HTA as a function of clicks and scrolls is a credible predictor of task time. This was proven to be true statistically as shown in the results of the empirical study in Chapter 5. Unlike the Keystroke Level Model (KLM) method that uses a number of pre-defined operators to predict expert error-free task execution times (Stanton, 2013), an elementary approach was taken in this study of counting the clicks and scrolls without association these with an execution time.

3.4.2.1 Scribble

The HTA for Scribble showed that a user could set schedule, condition, or action in any order. This was the only program that allows multiple conditions and actions in the HTA study. Having this flexibility resulted in a wider and longer HTA structure. The wider breadth sequence as shown in Figure 7 indicated more freedom to choose sequence steps. Thereby showing that it was relatively easy to correct mistakes or makes changes if user changes his/her mind. It was noted that the order of conditions and actions in the icon scroll 'list' should be was not obvious (e.g. alphabetical).

3.4.2.2 IFTTT

The HTA for IFTTT revealed a linear sequence as seen in Figure 9 and Figure 10. The user must enter condition first followed by an action. This program only permits one condition and one action per web task. It was noted that channels were organized in alphabetical order for condition and action selection. The interface of IFTTT is one long page (as opposed to separate pages per step). Users could scroll up to view previous steps. They only present current information on screen (i.e. the current step the user is on), and it was difficult to find the back button.

Unlike Scribble, all recipes run about every 15 minutes or sooner by default. There is no way to program a schedule for your recipes. The HTA revealed that the layout encouraged using published recipes, as this option is available on many pages (e.g. browse published recipes or favourites page) as seen in Figure 9 and Figure 10.

3.4.2.3 Zapier

The HTA for Zapier showed that the user had the flexibility to select either trigger app or action app first. As seen in the HTA flows Figure 11 and Figure 12 Zapier had the widest structure. This interface presents many links/options on the homepage. It provides articles, use cases, and recommended Zaps (based on app selection) information under Explore menu option, Use Cases tab, and Popular tab. Zapier makes heavy use of dropdown menus. It was noted that it presents trigger/action choices in no particular order (IFTTT uses graphical squares and user clicks on selection). The trigger and action are seen on the same page/screen (like Scribble).

Similar to IFTTT, the task input sequence is one long page (can scroll up to view previous steps) and only present current information on screen. There is no back button. User must scroll up to previous step to correct or change previous selections.

Unlike Scribble, all zaps run about every 5 minutes by default. There is no way to program a schedule for your zaps. A unique feature of Zapier is that it allowed testing with test data so the user could elect to test the zap to ensure the task would execute.

The HTA revealed that the low level complexity task involved many steps. From the HTA it could be seen that the low complexity task was similar to a high complexity task examining all the steps involved.

It was also noted that the HTA revealed that the layout encouraged using published recipes, as this option is available on many pages (e.g. view use cases, view popular zaps).

3.5 Keystroke and Mouse Click Counts

The number of keyboard keystrokes and mouse clicks were tracked for 4 high complexity and 4 low complexity tasks that were examined in the HTA, using Mousotron[7] software. Tasks assumed each app/BOTBit has already been initialized (i.e. user information and permissions were granted).

There was an evident clustering of high complexity tasks and low complexity tasks. The high complexity trials had notably greater number of both key presses and scrolls. From this observation, it can be hypothesized that the high complexity tasks will take longer than the low complexity tasks. For the high complexity tasks the number of key presses and scrolls were variable, implying that there would be a higher standard deviation for the high complexity tasks than the lower complexity tasks. The results of the usability study support this assertion, detailed in Section 5.8.2.

With IFTTT and Zapier there was the capability of using the search function to search for apps (i.e. by entering the app name in the search field), or the user can scroll through the list of apps to find the

[7] www.blacksunsoftware.com/mousotron.html

desired one. The results of the web tasking keystroke and mouse clicks and scroll were consolidated and presented in Figure 13. It is evident from these results, that Zapier had the highest counts, IFTTT had the lowest number of count, while the Scribble interface was in the middle. Interestingly, this count to some extent matched the breadth and depth in the structures created in the HTA (i.e. Zapier was the widest, Scribble was second widest, and IFTTT was narrowest).

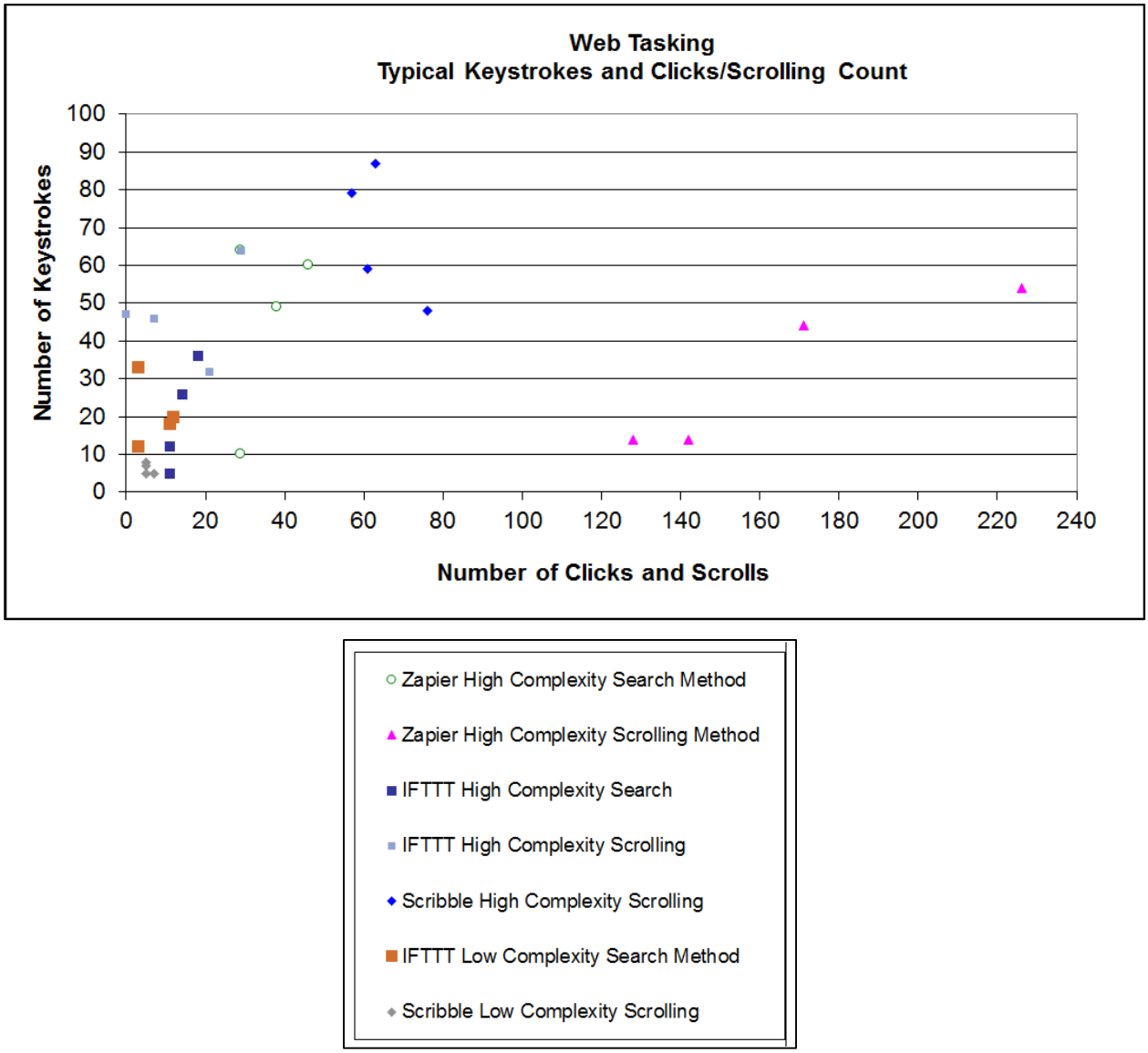


Figure 13: Web tasking keystrokes and mouse clicks and scrolls count

3.6 Conclusions and Recommendations

The HTA successfully showed the hierarchy of actions users must undergo to complete high and low complexity tasks in several web tasking interfaces. It showed generalized structures of tasks and has demonstrated that the task structures in HTAs are telling of the complexity of tasks. The wider the structure was (particularly in Zapier), the more options the user may have perhaps making the task more difficult for a user. The results from the empirical study show this in Chapter 5 . The narrower the HTA structure, the more prescriptive a task is, thereby making it simpler to complete. The HTA showed the selections needed to complete a given task and inherent inherently showing potential areas for human error. The HTA was also a helpful tool in thoroughly investigating each interface and identifying differences in interaction types at the task level. For example, Zapier uses drop down menus, and IFTTT uses click and scroll selection of icons.

This study has shown that HTA was descriptive and showed differences in high and low complexity task levels. In addition, as seen in the results of the Keystroke and Mouse Click and Scrolls count, recording this count during an HTA can be a powerful preliminary tool in predicting performance in a given web tasking interface. A variety of approaches, representing a considerable range of complexity, exist to assess interface performance. HTA can be used as a simple tool to achieve the same results. This is further investigated empirically in Chapter 5, where it was hypothesized that a simple sum of task input requirements (counting keystrokes and mouse clicks and scrolls) would be closely related to performance task time.

Chapter 4
A New Web Tasking Interface: WebTasker

Users seem to expect easy to use and easy to learn interfaces, especially when it comes to apps. Consideration of users' expectations or needs and different sources of input should go into a design of a web tasking interface. The goal of creating WebTasker was to address HF issues in existing web tasking interface and to model an easy to use and learn metaphor. There were several inputs into the researcher's design of WebTasker (see Figure 14).

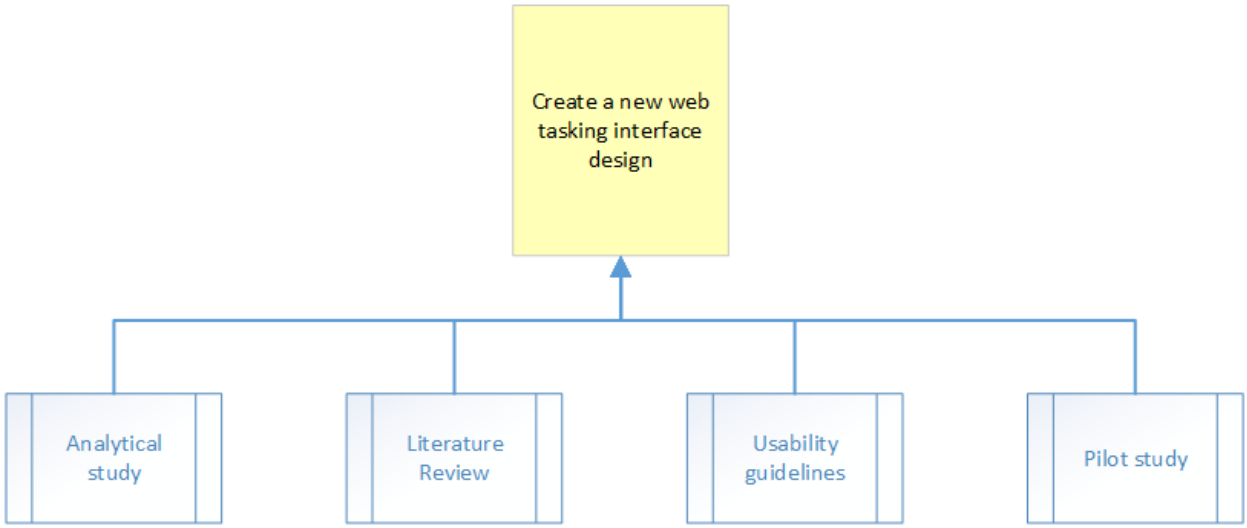


Figure 14: Design input to WebTasker

The information and findings collected from the HTA (analytical study) were used as a starting point for the WebTasker design. Through conducting the HTA, the researcher became familiar with current existing interfaces (i.e. what worked well, what she liked/didn't like, and design attributes to carry-over into a new design).

Scribble was the only program/interface that allowed multiple conditions and actions input for composition of a web task. Scribble allowed entry of schedule, condition, or action in any order. Other findings for Scribble included that it was easy to correct mistakes, it had a wide breadth sequence (indicates more freedom to choose sequence steps), however having more choices made this interface more prone to errors, and the order of conditions and action apps in the icon scroll 'list' should be in a specific order (e.g. alphabetical).

IFTTT's interface was simple and prescriptive, having a linear HTA structure. Entry of a condition must be first, followed by an action. Channels were organized in alphabetical order for condition and action selection. It was relatively difficult correct mistakes (i.e. find the back button).

All recipes run about every 15 minutes or sooner by default and there is no way to program a schedule for your recipes. IFTTT's interface layout seems to encourage using published recipes.

In the Zapier interface the user is presented with many links/options on the homepage. The user can select either trigger app or action app first and it makes heavy use of drop down menus. The dropdown menu interaction is not ideal when there are many items in a list to choose from. It was relatively difficult to recover from mistakes, as there was no back button (user must scroll up to previous step to correct or change previous selections). There is no schedule capability and by default zaps are run every 5 minutes. Zapier had a unique optional test feature to allow users to test their zap with test data provided by Zapier. A main finding from the HTA was that low complexity task involves many steps and appeared more as a high complexity task based on the HTA structure. The literature from Section 2.3 fed into the design as well. Along with some usability guidelines, specifically used in this thesis were Nielsen's Heuristics. After the completion of the heuristic review, a pilot study with a handful of participants was conducted. The pilot study is described in detail in Section 4.2.

4.1 Usability Guidelines

Nielsen's Heuristics (Nielson, 1994) were used as a guide to find usability problems in the current user interface designs of the platforms. A Heuristic Evaluation is a well-known usability engineering method for finding usability issues so that they can be attended to as part of an iterative design process. Typically heuristic evaluation involves having a small set of evaluators examine the interface and judge its compliance with recognized usability principles. In this study the researcher made usability observations on each interface by performing the tasks used in the HTA in Table 4 and recorded them against each heuristic and used a simple scoring system to evaluate the three existing web tasking interfaces.

Scoring:

✓+ =Acceptable or good: no usability issues against this principle were observed and this design principle was incorporated well.

✓ =Room for improvement: one or two minor usability issues were observed.

✓- =Needs attention: one or more major usability issues or three or more minor usability issues were observed.

A summary of the results of this heuristic review are presented in Table 5, along with design recommendations in the last column made by the researched based on this review.

Table 5: Simple Heuristic Review of IFTTT, Zapier, and Scribble

#	Nielsen's Heuristic	IFTTT	Zapier	Scribble	Design Recommendation for WebTasker
1	**Visibility of system status** The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.	✓+ -Step counter in flow and task information entered is displayed.	✓+ -Step counter in flow and task information entered is displayed.	✓- -No indication if puzzle piece information is complete. -No prompt or suggestion of what information to enter next (novice users need this).	Incorporate task status in design (graphically if possible).
2	**Match between system and the real world** The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.	✓ -Search of recipes is based on channel names (not recipe titles). -Variable names appear as programming language.	✓- -Makes heavy use of drop down menus with text versus icons. -Variable names appear as programming language.	✓ -Variable names appear as programming language. -BOTBit is not an intuitive term compared to "trigger app" or "action app". -Use of dropdown was not efficient for some items (e.g. selecting time).	Use clear icons with corresponding labels of condition and action apps (like IFTTT). Avoid drop down menus, where there are long lists. Hide any code or variable names that may be confusing to the user.
3	**User control and freedom** Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.	✓- -Difficult to find "back" button	✓- -There is no "back" button. User must scroll up to previous step to correct or change previous selections.	✓ -No "undo" or "redo" functions.	Provide a way to easily edit task components.

#	Nielsen's Heuristic	IFTTT	Zapier	Scribble	Design Recommendation for WebTasker
4	**Consistency and standards** Users should not have to wonder whether different words, situations, or actions meant he same thing. Follow platform conventions.	✓	✓ - Presents trigger/action choices in dropdown menu in no particular order.	✓ -Order of conditions and actions in the icon scroll 'list' should be is not obvious (e.g. alphabetical).	Use consistent conventions.
5	**Error prevention** Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.	✓ -Difficult to initially locate "back" button. -Using the Back browser button will give a navigation warning. If you leave the page, current recipe user is working on will be lost.	✓ -Offers testing with sample data. -Using the Back browser button will give a navigation warning. If you leave the page, current recipe user is working on will be lost.	✓ Easy to correct mistakes or makes changes if user changes his/her mind.	Provide minimal ways users can click away from composing a web task. Only present necessary information.
6	**Recognition rather than recall** Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.	✓+ -Can scroll up to view previous step. -Captures full goal/task in simple sentence If THIS THEN THAT.	✓ -Final task is not displayed in final step of naming Zap or on confirmation page. -Captures goal/task in two sentences "when this happens"… "do this".	✓+	Display task composition on one screen if possible (like Scribble puzzle metaphor).

#	Nielsen's Heuristic	IFTTT	Zapier	Scribble	Design Recommendation for WebTasker
7	**Flexibility and efficiency of use** Accelerators -- unseen by the novice user -- may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.	✓ - Must enter condition first.	✓ -Can set trigger or action app in any order.	✓ -Can set schedule, condition, or action in any order. - Only program that allows multiple conditions and actions.	Consider incorporating an accelerator for complex task (with more than one condition and action).
8	**Aesthetic and minimalist design** Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.	✓+ -Captures full goal in simple sentence If THIS THEN THAT. -Aesthetically pleasing use of icons.	✓- -Too much information presented to user on one page.	✓ -Wording in dialogues can be more concise. E.g. "Information needed for this condition" and "Specify conditions" may be redundant.	Aim for minimalist design that is graphically pleasing (like IFTTT).
9	**Help users recognize, diagnose, and recover from errors** Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.	✓	✓	✓	Provide help link/button from any point in composing web tasks.

#	Nielsen's Heuristic	IFTTT	Zapier	Scribble	Design Recommendation for WebTasker
10	**Help and documentation** Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.	✓	✓	✓	Provide help link/button from any point in composing web tasks.

4.2 Pilot Study

A pilot usability study with existing web tasking interfaces was conducted in August 2015. The intent of the pilot study was to explore the research questions:

- How can users successfully engage in creating web tasks with existing interfaces?
- Does having multiple conditions and/or actions in a web task affect user performance in putting together a web task?

There were four participants in total, two with basic programming experience, one with expert programming experience, and one with no programming experience. Each participant was given two tasks at each complexity level in three interfaces: IFTTT, Zapier, Scribble, and one task at high complexity for Node-RED[8]. This is because Node-RED is not a web tasking platform. It was evaluated as part of the pilot study to examine its viability as a web tasking platform and the researcher was only able to produce one web task to test. Node-RED is a browser-based flow editor that wires together flows using the wide range nodes, then flows can be deployed to the runtime. There were 19 trials in total.

4.2.1 Pilot Study Results

The results of the pilot study were recorded qualitatively by the researcher by notetaking through observation. Below is a summary for each interface.

4.2.1.1 IFTTT

Participants enjoyed using IFTTT in general, as participants verbally reported this as they used the IFTTT interface. All tasks completed under 5 minutes (most 2-3 mins). Three out of four made comments regarding the fact you could see the html code, the one with no programming experience said it looks confusing (e.g.
 and variable names in Figure 15 and variable names shown in Figure 16.

Three of the participants noted that the search field does not accept "if this then that" statements, but worked only for keywords or app names. Nevertheless, the participants were able to use the search function successfully.

[8] www.nodered.org

All participants verbally reported that IFTTT was aesthetically pleasing and makes good use of icons. Participants also reported that they liked that IFTTT offered recommendations for other recipes, as the program offered them recipes they may not have searched for themselves.

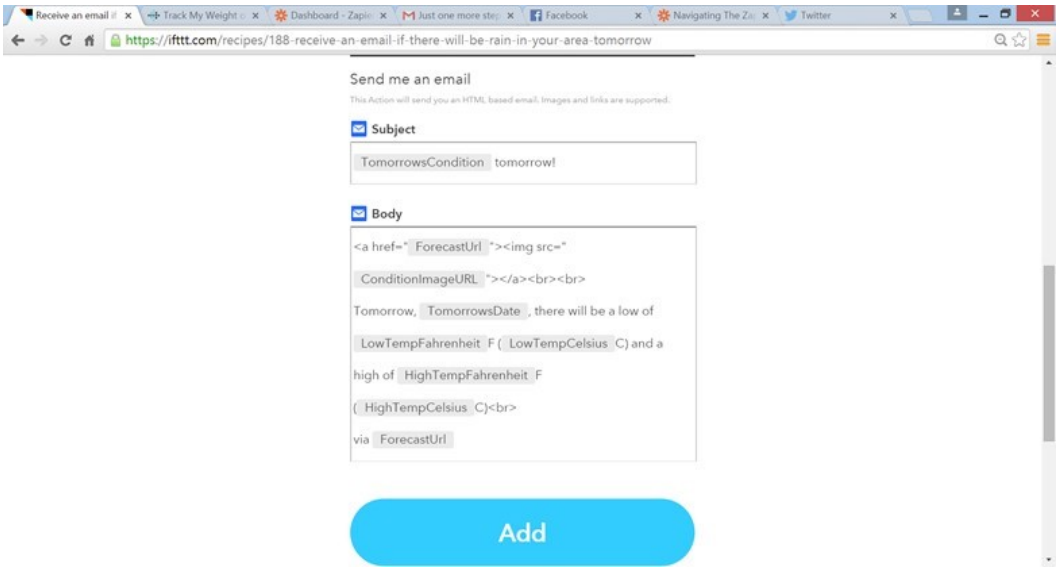


Figure 15: IFTTT Screenshot with html code

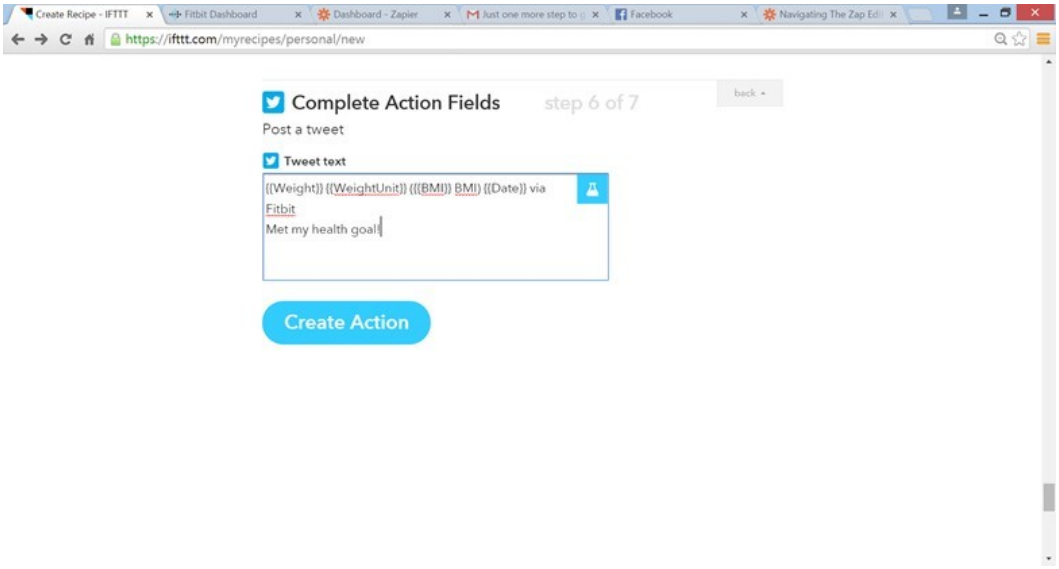


Figure 16: IFTTT Screenshot with variable names displayed

4.2.1.2 Zapier

The pilot study results for Zapier were in line with the findings from the HTA. Participants found that there was "too much" information and "too many" fields are displayed at one time (as seen in the example in Figure 17). In the HTA, Zapier had the widest structure (Figure 11 and Figure 12), demonstrating the amount of information shown to the user in one step or screen.

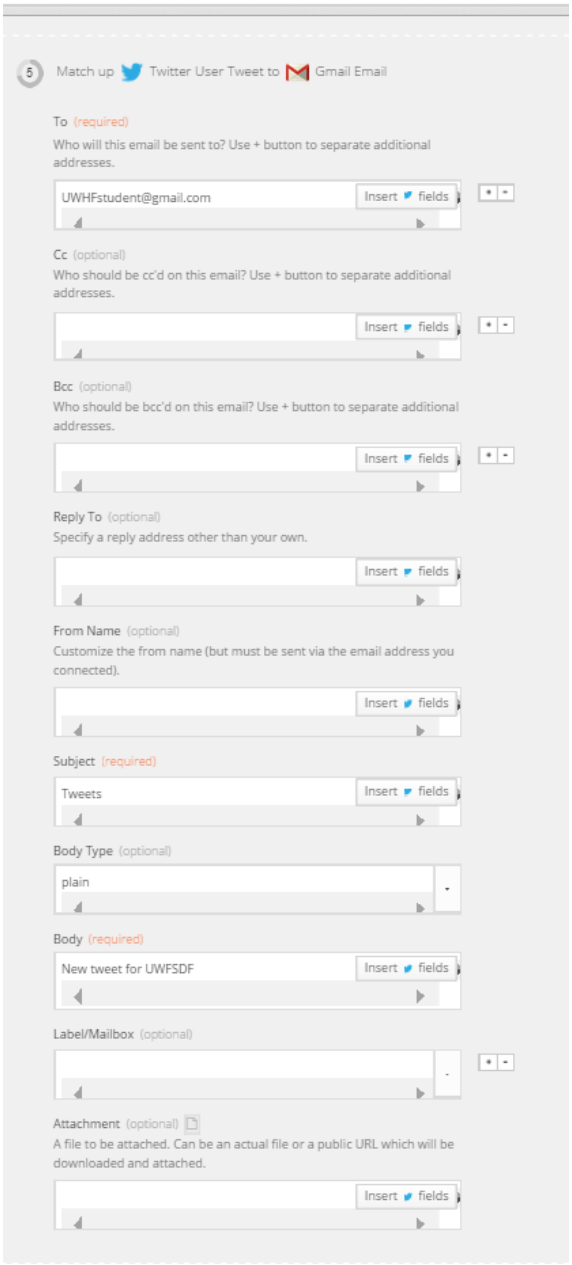


Figure 17: Zapier screenshot of input fields

In Figure 17, it is difficult to immediately see which fields are mandatory fields to be filled out by the user. Mandatory fields should be grouped at the top or optional fields should be hidden until prompted by the user.

Zapier offered some advanced capabilities where filters can be applied to some apps. The "filters" were reported by the participants to be confusing and they were not able to tell what they should be used. An example of the filter capability can be seen in Figure 18.

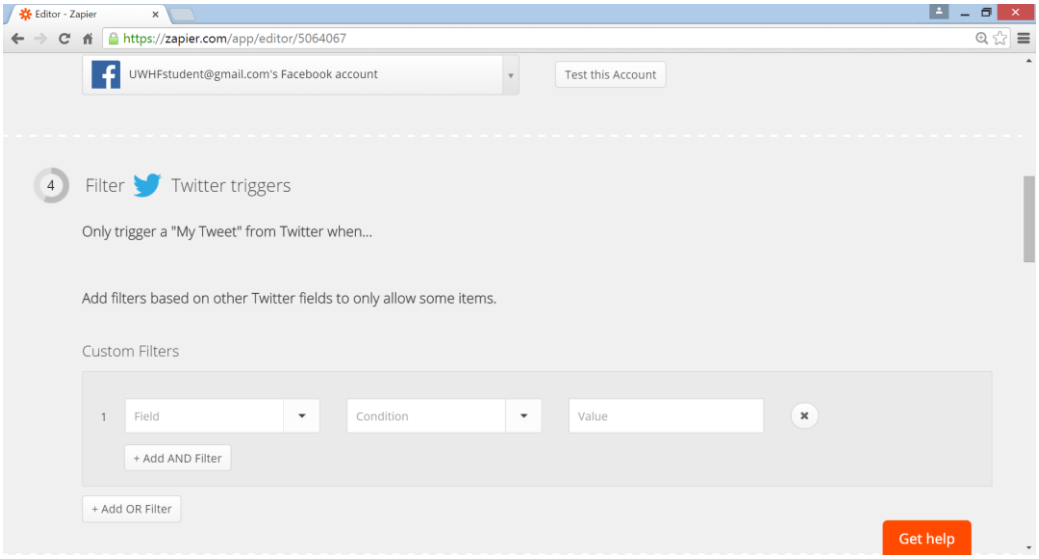


Figure 18: Zapier screenshot of filter use

Half the participants found the search feature (to search published zaps) difficult to initially find. One way to search is to click "explore" and users did not know this was the search.

Most high complexity tasks took over five minutes to complete. Some tasks took 10-15 minutes to complete. Participants found creating zaps from scratch (high complexity) easier than searching for pre-made/published zaps. All participants found the search function difficult to use. In general, participants felt there were too many steps involved to create a task in Zapier.

Among some of the other issues reported were that, there was no homepage to click on once you visit your dashboard page; Zapier blog, video tutorial, and learning center buttons are displayed in "explore" and seem out of context here; participants were unsure what the "test" function was. The test function allows the user to test their zap with test data to ensure that the apps are responsive and the task will be executable. Half of the participants tried it, the other half did not bother.

4.2.1.3 Scribble

It was observed during the pilot study that this interface seemed to have the most learning involved, as the researcher observed participants struggling to put together a task. Participants had to learn the puzzle piece metaphor, where they had to associate each puzzle piece colour with a part of the web task (i.e. red piece for action, orange piece for condition, and blue piece for schedule). Tasks took about 5 to10 minutes for high complexity (longer than IFTTT).

It was not apparent to participants that schedule, conditions, and actions could be entered in any order. Half the participants users thought that you had to enter schedule first (since it is displayed first on the "action track"). All participants tried to drag the condition piece instead of clicking to add.

Scribble had the most observed errors (e.g. putting the wrong information in the wrong place; putting the condition BOTbit in the action puzzle piece). It was reported by participants to the terminology should be clear, for example "Execute Later" in schedule makes user think it will only be done once later, but it can be recurring. Another example is "Notify someone" was not explicit enough as if it notifying by email it should say email and not generally say "notify".

The search task (low complexity) was easily done in Scribble, as there are currently a few published Scribbles in the library and the search works on keywords.

4.2.1.4 Node-RED

Participants were asked to complete one task in Node-RED. The task was: if you get a new tweet send an email. None of the participants were able to complete this task. The nodes were to be connected then clicked into to complete the programming of the node to run. The best performance observed was to drag and drop the twitter and email nodes, under the "social" category, shown in Figure 19. Unlike the other web tasking interfaces in this pilot study, Node-RED was not a viable interface without any training. The one expert programmer participant said this platform has potential to be a good web tasking platform if packaged nicely (i.e. used templates and had no API programming necessary).

From the pilot study results, Node-RED should not be utilized as a web tasking platform in its current form.

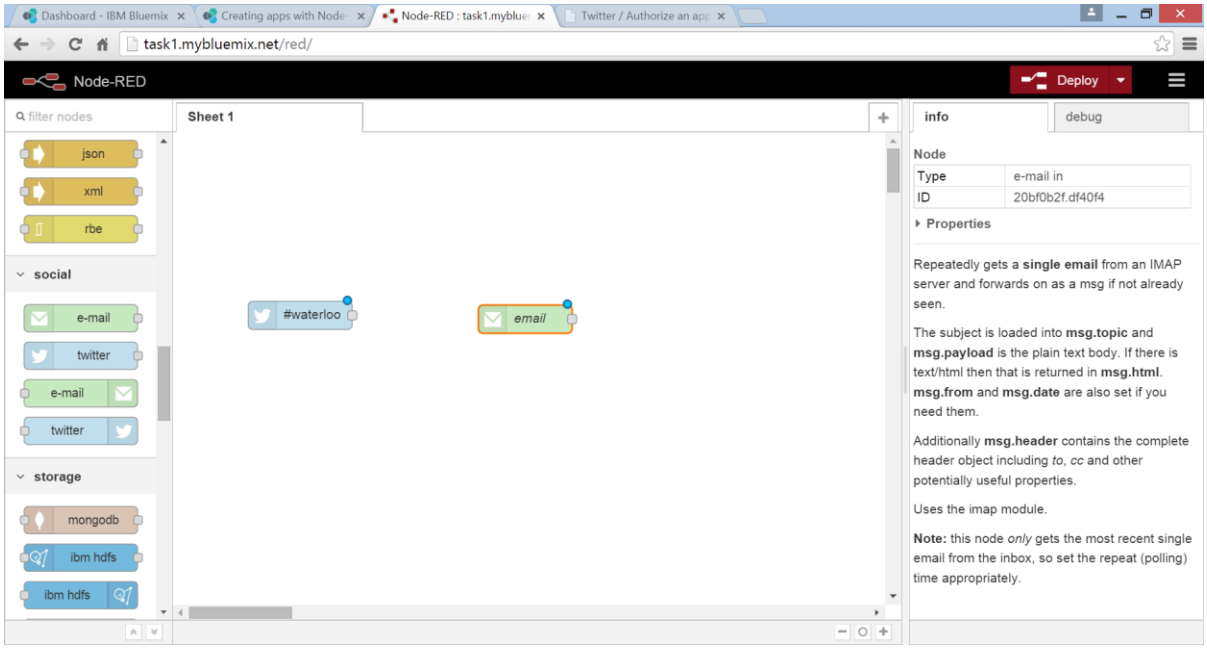


Figure 19: Node-RED screenshot

4.3 Early web tasking concept ideas and prototypes

As the researcher hypothesized that users' would prefer composing a web task by entering the condition first then the action, the early concepts for a new web tasking interface all revolved around a cause and effect model. A cause and effect model usually consists of three attributes: (1) temporal precedence, (2) whenever the cause happens, the effect must also occur, (3) no plausible alternative explanations (Trochim, 2005). The initial idea for the prototype was to have a gear metaphor, as seen in Figure 20.

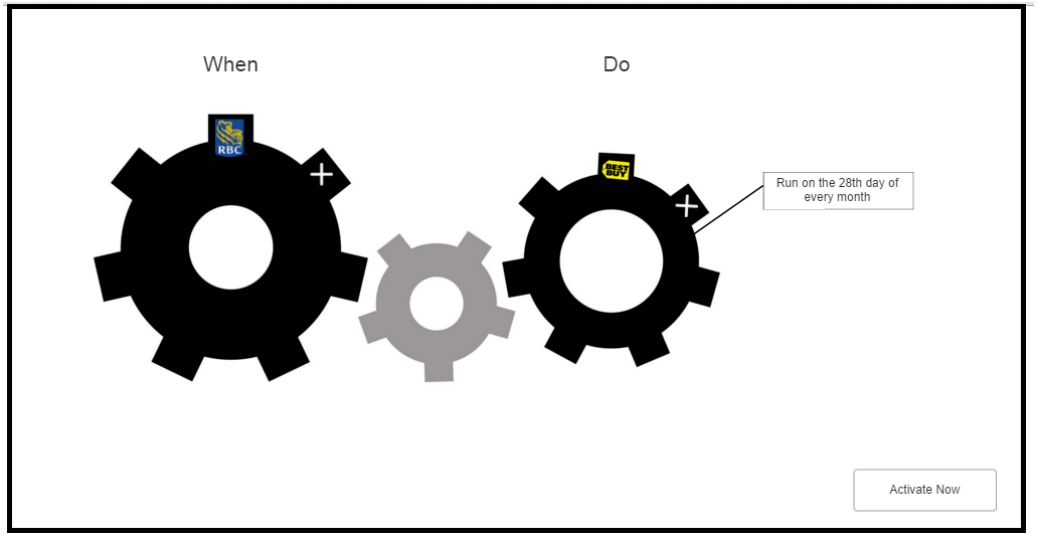


Figure 20: WebTasker early prototype gear metaphor

The conditions were placed on the left gear and the actions on the right gear with a connecting gear in the centre to attach the two. The second iteration of the gear metaphor was going to include more salient plus sign (to indicate to the user to add more conditions or actions), text in middle gear to indicate next possible steps or instructions, text of condition and action statement, or model a chain to connect gears instead of middle gear. However, there was a fundamental flaw in this design. The gear model failed to be sufficient for a use case in Scribble where specific conditions could be associated to a specific action (as per the stacking of condition pieces atop of the associated action in the puzzle metaphor). The gear assumed that all conditions would cause all actions. The incorrect and correct mental models are illustrated in Figure 21 and Figure 22, where C=condition and A=action.

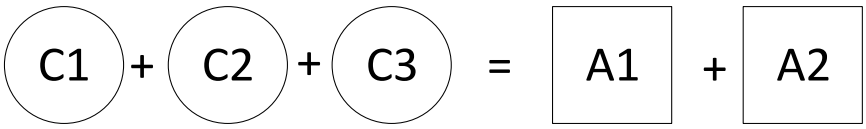


Figure 21: Incorrect web tasking mental model

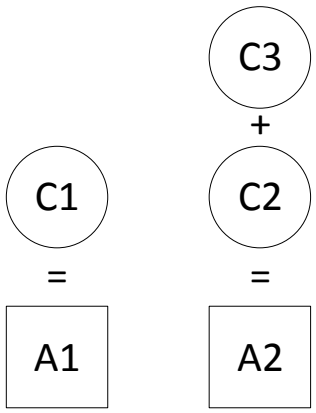


Figure 22: Correct web tasking mental model

After this realization, the gear metaphor was no longer pursued.

Other metaphors were then considered, such as chain link or bubble model where the action in the center with conditions around them. There was one metaphor that was clear and easy to interpret – the journey line or track and vehicle metaphor. The chain link metaphor was difficult to visually distinguish conditions from conditions because the chain links were too similar another coding method, (such as colour) would have to be introduced. In the bubble metaphor, it was difficult to model the links between the specific conditions and affiliated actions when it came to having more than one condition and action. The journey line design is explained in Section 4.4.

4.4 Web Tasking Interface Prototype: WebTasker

The role of prototyping in the software design process is to explore and evaluate the design. It also gives the designer the opportunity to communicate ideas, and it can be a form of a "design specification". Prototypes can be used for early usability testing, and be changed many times before the final design is achieved; thereby final systems can be developed much faster and cheaper (Nielson, 1993). Input of the design through prototyping in conjunction with user testing is one of its main roles in the software design process. In this thesis, the user testing with prototypes was used with actual tasks. An interactive medium fidelity prototype of WebTasker was created using Axure software.

The journey line metaphor was selected for the new web tasking interface, WebTasker. Table 6 explains the design attributes or features of WebTasker and provides the rationale and reference source to support the design. Figure 23 to Figure 28 show screenshots from the WebTasker prototype. These figures are annotated with number bubbles that correspond to the design attributes

in Table 6 (note that these numbered bubbles were not on the actual prototype). Source names were abbreviated in this table and include NH= Nielsen's Heuristic, PS= Pilot Study, RI=Researcher's Idea, and HTA= Hierarchical Task Analysis. Figure 23 to Figure 28 are a small sample of the screens/scenarios used in WebTasker, and is not to be interpreted as a comprehensive set of screens used in the usability study.

Table 6: WebTasker Design Attributes and Rationale

Prototype Bubble #	Design Attribute	Description/Design Rationale	Source
1	Logo and link to homepage	Provide a link to the homepage at all times	PS - Zapier had no link to homepage, and users expected to have this link. NH3: User control and freedom.
2	Link to create new task	Clear and salient link/button to create a new task	NH8: Aesthetic and minimalist design.
3	Menu -My Dashboard -Recommendations -Settings -Help	Keep terminology simple and do not provide too many choices	HTA – interfaces with wide structures were more difficult to use, so we minimized menu items to four. HTA and PS – showed interfaces encouraging use of published tasks, a well-received feature, so we included "Recommendations". NH10- Help and documentation, should be made easy to access.
4	Search	Simple and salient	PS- search was not salient enough in Scribble or Zapier, thus we made it big and left a lot of white space around it to increase salience.

Prototype Bubble #	Design Attribute	Description/Design Rationale	Source
5	Recommended (published) Tasks	Provide recommendations as a way to show users task examples	PS -a well-received feature so we display some on the homepage in addition to the link always in the menu.
6	Car icon	Should move to indicate task execution status (not working in current prototype)	RI – this does not necessarily need to be a car. It could be a simple ball/circle symbol. NH1: Visibility of system status
7	Journey Line	Line will encounter condition first followed by action	RI- inspired by cause and effect model
8	Add Condition link	Simple shape of a half ellipse. May be analogous to a bump in the road.	RI
9	Add Action link	Simple shape of triangle. May be analogous to a yield sign.	RI – chose shape to be distinctly different than condition
10	Set schedule link	Simple calendar icon to set schedule of running the task	RI – Schedule is placed at the end after user has decided on condition and actions.
11	Add new set of condition(s) and action(s) link	Big plus sign	RI – plus sign differs from half ellipse and triangle to indicate adding a new set.
12	Field to enter task name	Simple input field and save or save and submit button.	HTA- this task was necessary in all tasks in all interfaces.
13	App icon	Icon will be displayed in either the half ellipse (condition) or triangle (action) after user selects it.	HTA- this was common in all tasks in all interfaces.

Prototype Bubble #	Design Attribute	Description/Design Rationale	Source
14	Summary in text	The condition or action will be in text below the journey line and corresponding icon.	NH6: Recognition rather than recall. Instead of just displaying the icon, also provide the text at all times for the user. HTA – some interfaces did not have this summary and it is a useful feature.
15	Edit link	In case user makes mistake, a link to edit the information is always available before submission of the task.	NH9: Help users recognize, diagnose, and recover from errors
16	Search apps	Apps can be added by selecting from the list or by search. Search field should be at the top and salient.	PS – users had difficultly locating the search in Scribble and Zapier.
17	App icons and names (in condition and action selections)	In addition to the icons ensure names are there too.	PS – not all users recognize app logos.
18	User input fields	Fields are app and task specific.	HTA and PS: avoid dropdowns for long list selections.
19	Delete app link	Link provided to delete the app from condition or action if user makes a mistake.	NH9: Help users recognize, diagnose, and recover from errors

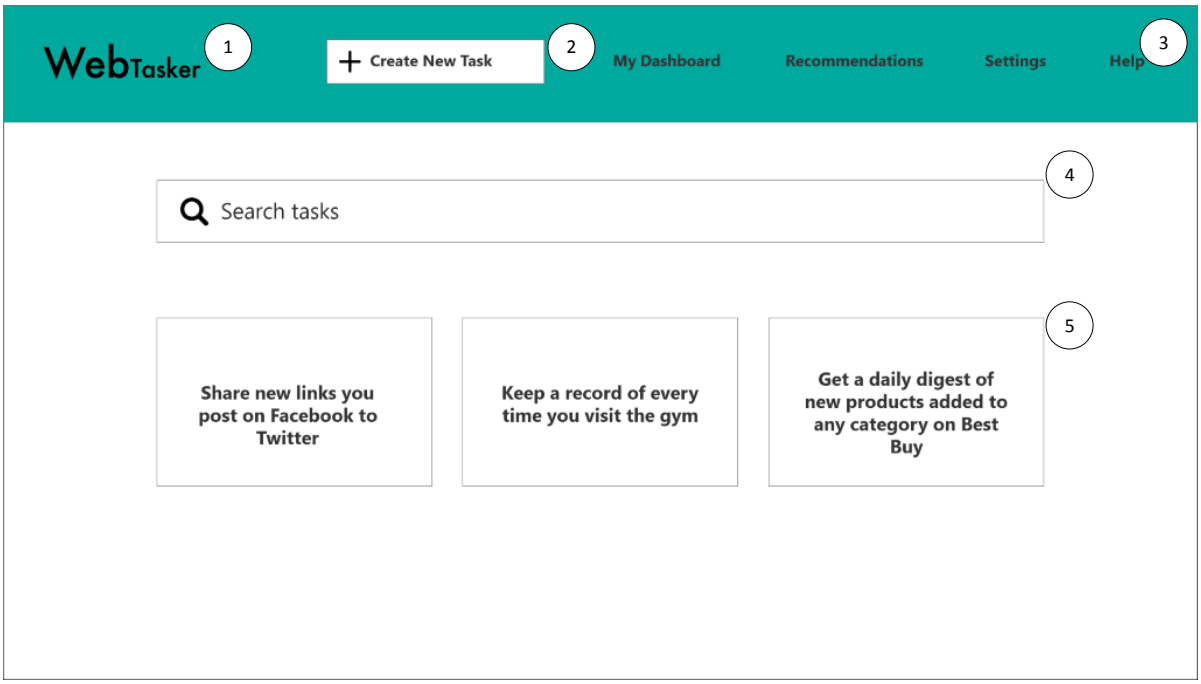


Figure 23: WebTasker Homepage

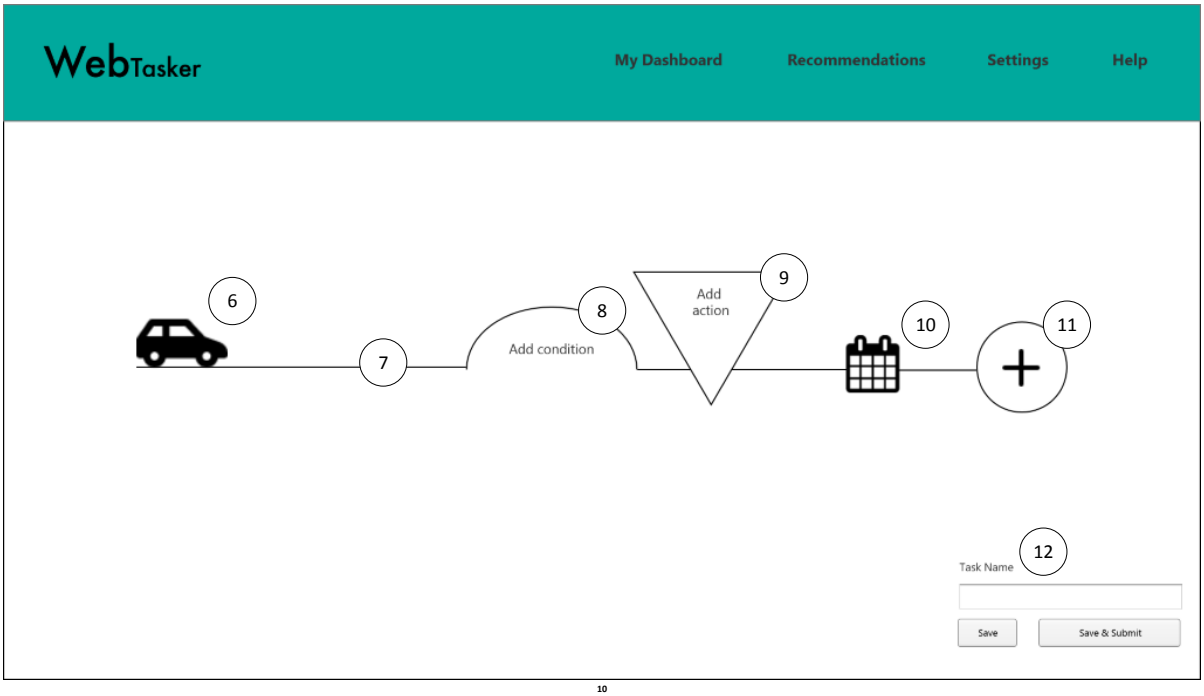


Figure 24: WebTasker Create New Task, Blank Journey Line

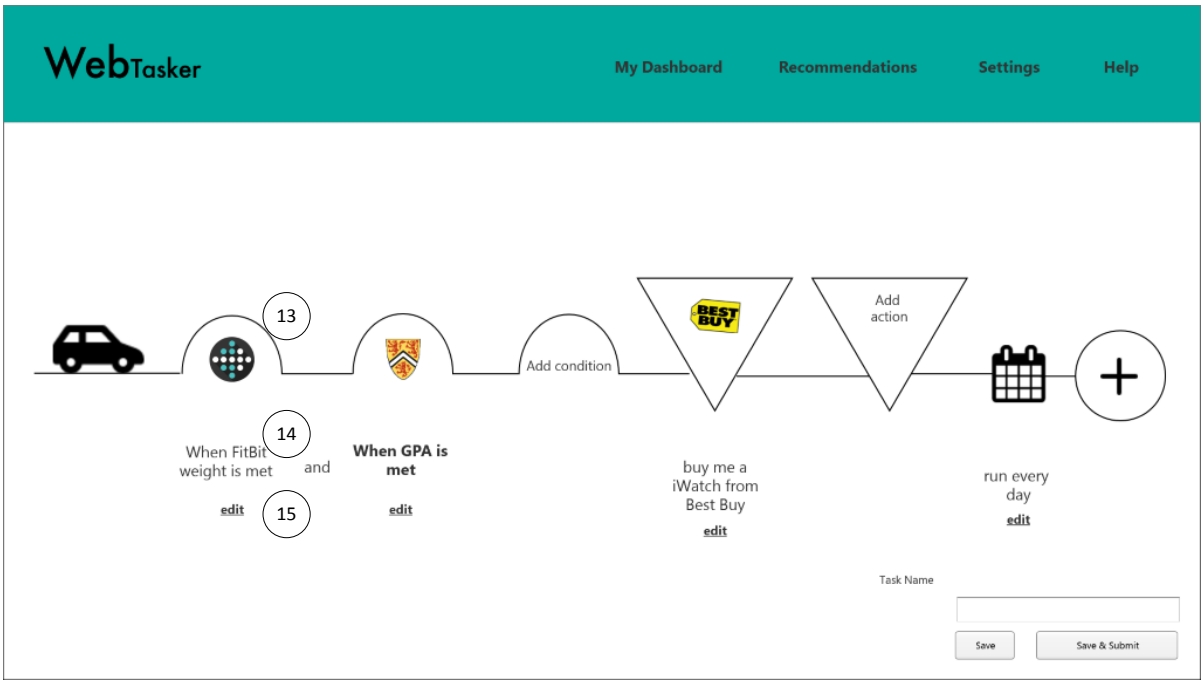


Figure 25: WebTasker Example Web Task, populated journey line

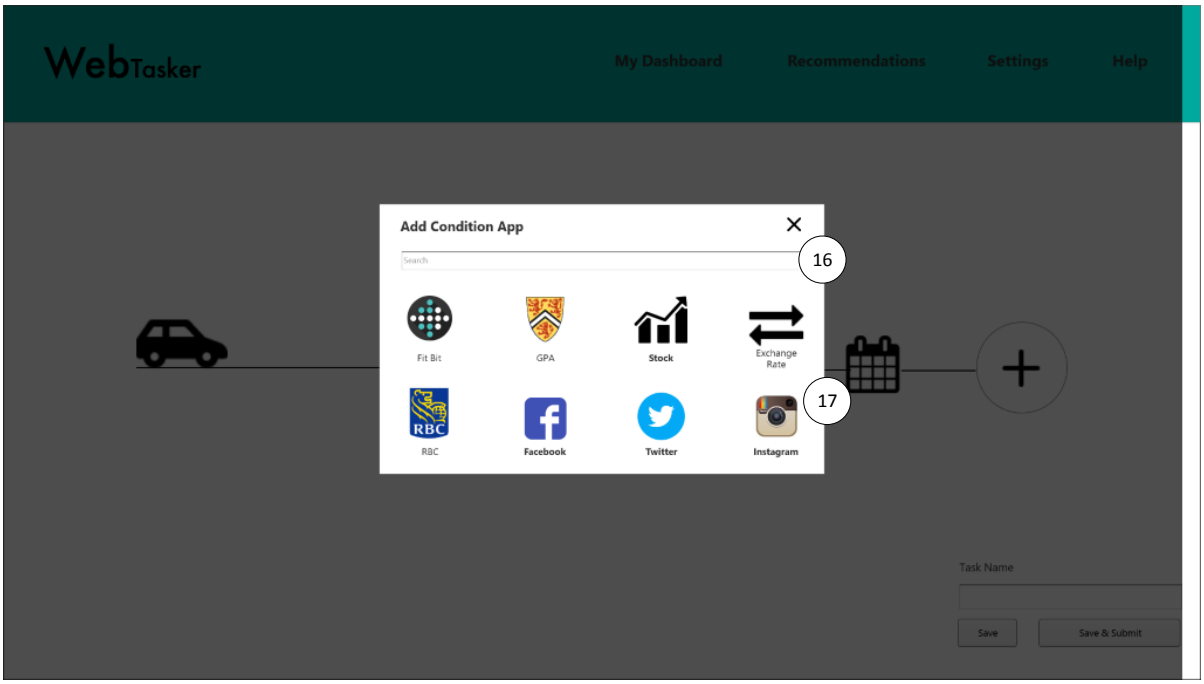


Figure 26: WebTasker Example Add App

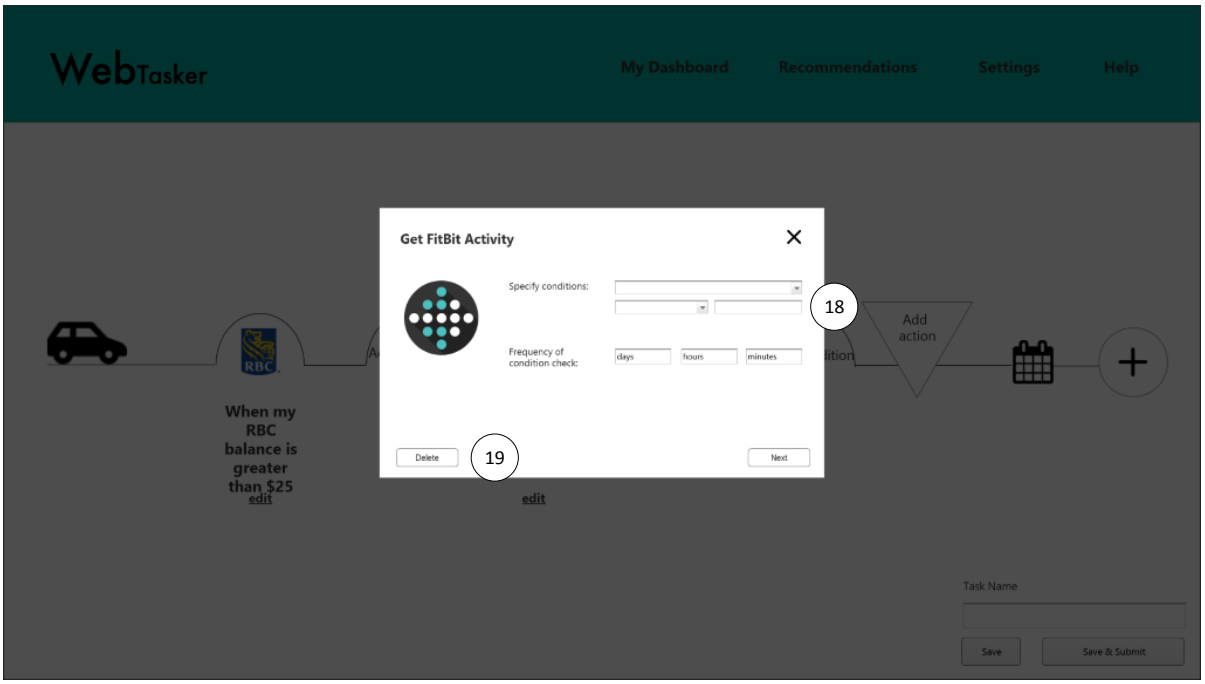


Figure 27: WebTasker Example User Information Input

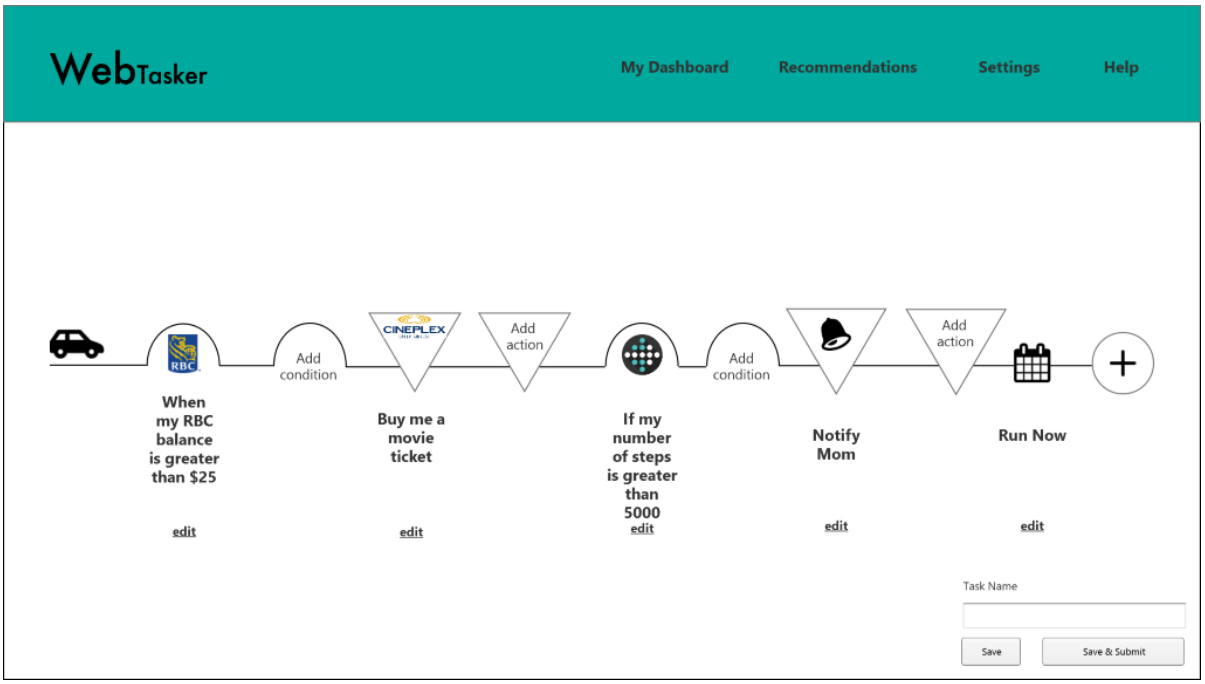


Figure 28: WebTasker Example Complex Task

4.5 Other Design Factors

Web tasking interfaces should be developed with the goal to eliminate the complexity of software engineering (i.e. programming), such that end users can utilize the interface to create and control tasks for themselves. The competence and capability of users should always be considered when designing products. The question of users' capability is usually encountered and whether end users are capable of programming applications or not, given that [some] do not have any training as programmers (de Souza et al., 2001). WebTasker was designed with this in mind, to make the programming aspect of composing a web task transparent to the user.

Another prevalent problem among end user tasking is the problem of representation of what they are supposed to do to achieve the task versus what they perceive or interpret they have to do. The sense-making process is based on the users' pattern recognition capacity, on language documentation, on computer literacy and cultural background, or a combination of all (de Sousa, 2001). Norman (1988) discusses affordances and the user's model otherwise known as a mental or cognitive model. The user's mental model is developed through interaction with the system, thereby allowing people to make predictions about how things will work. The design model (also known as the conceptual model) is the model that the designer conveys to the user through the interface. Norman (1988) explains that a good conceptual model is needed to:

- Allow users to predict the effects of their actions.
- Without one, users operate blindly.
- Users cannot fully appreciate why and what effects to expect.
- Users can manage when things go wrong.

The linear model utilized in WebTasker has a conceptual model that builds on a general audience recognition and language capability.

Another aspect of design that was briefly explored in this thesis was end user's risk tolerance. Users will "code" or attempt to program when needed, to expand functionality or make tedious tasks easier or quicker. This problem has to do with how much a user is willing to risk. An example is outlined in a study where a search and replace task in a document was carried out (Blackwell, 2001). The study found that, "Programming tasks require concentrated attentional effort, yet there is always a risk that the program will not work as expected (even after testing), and that a manual approach to

the same task might have been a better investment" (Blackwell, 2001, p. 481). A web tasking interface should be designed such that users feel confident in the tasks they are composing. Zapier addressed this by having a test feature to test the zaps with test data provided by Zapier. WebTasker could consider an optional test function for tasks in a future iteration.

Chapter 5
Usability Study

Eight of the common and representative tasks defined as part of the first phase in the HTA were used as tasks for testing. The tasks were representative of interactions with web tasking platforms and many were given as example tasks provided by their creators (available on their websites). Each task had a complexity manipulation, where high and low complexity task conditions were tested. These tasks and complexity manipulations were as follows:

- High level complexity tasks
    - Creating a web task (i.e. Scribble, Recipe, or Zap) from scratch without a template or starting from a 'published' Scribble, Recipe, or Zap.
- Low level complexity tasks
    - Involved selecting a previous Scribble, Recipe, or Zap either from a search or selecting it from a category.

Refer to Appendix A.1, for a comprehensive list of tasks and task steps used. Table 7 shows an example of a high and low level complexity task for each interface.

Table 7: Sample of tasks used in study

Interface	Example of high level complexity task/ Create task	Example of low level complexity task/search task
IFTTT	If daily step goal is achieved in Fitbit, then send a new email from Gmail.	Find: If it's going to rain tomorrow, send me an email from Gmail. Enter email address "ekittel@uwaterloo.ca"
Scribble	If fit bit step goal is met AND UW GPA is met, then buy me an iPad from Best Buy; run this task every day.	Find Take the next bus to the airport if the weather is clear.
Zapier	When you star an email in Gmail post it to your Facebook timeline.	Find: Email for a User's Twitter Tweets, Get an email via Gmail for new tweets from a specific user.
WebTasker	If my bank account balance is less than $1000 then notify me; run this task every Friday at 10:00 a.m.	Find: Notify me if Google Stock price changes.

5.1 Participants

Over the period of November 16 to 30, 2015, 16 people were recruited from the University of Waterloo to participate in this study. The number of participants was selected in order to achieve a balanced design. Seven males and nine females, ranging in age from 19 to 32 with a mean age of 22.5 years, participated. Half of the participants had computer programming experience and half of them did not. Those with computer programming experience ranged in experience from novice to expert in a variety of languages (e.g. MATLAB, C++, Java, etc.). Of the 16 people recruited, four of them had used IFTTT before. This information was collected with a demographics questionnaire (see Appendix A.3)

All participants met the criteria of having some experience using web applications (for example, email or weather apps).

The number of participants was selected in order to achieve a balanced design. For statistical usability studies, at least 10 to 12 participants per condition are needed, however, this depends on the desired reliability; standard statistical tests can be used to estimate the confidence intervals of test results and thus indicate the reliability of the size of the effects (Nielsen, 1993). The number of participants in this study yielded sufficient results to uncover the usability issues involved with each interface in the study.

5.2 Stimuli and Apparatus

The study took place in a controlled HF laboratory at the University of Waterloo. The study tasks were completed on a desktop computer with one 23" monitor with 1920x1080 screen resolution. The keystrokes and mouse clicks and scrolls were recorded in each trial using Mousotron[9] software.

The web tasking interfaces of these four designs have a range of different functionality and interaction styles – see Figure 29. Where interfaces have multiple interaction methods (e.g. scroll or search method), the participant was allowed to choose the preferred method.

[9] www.blacksunsoftware.com/mousotron.html

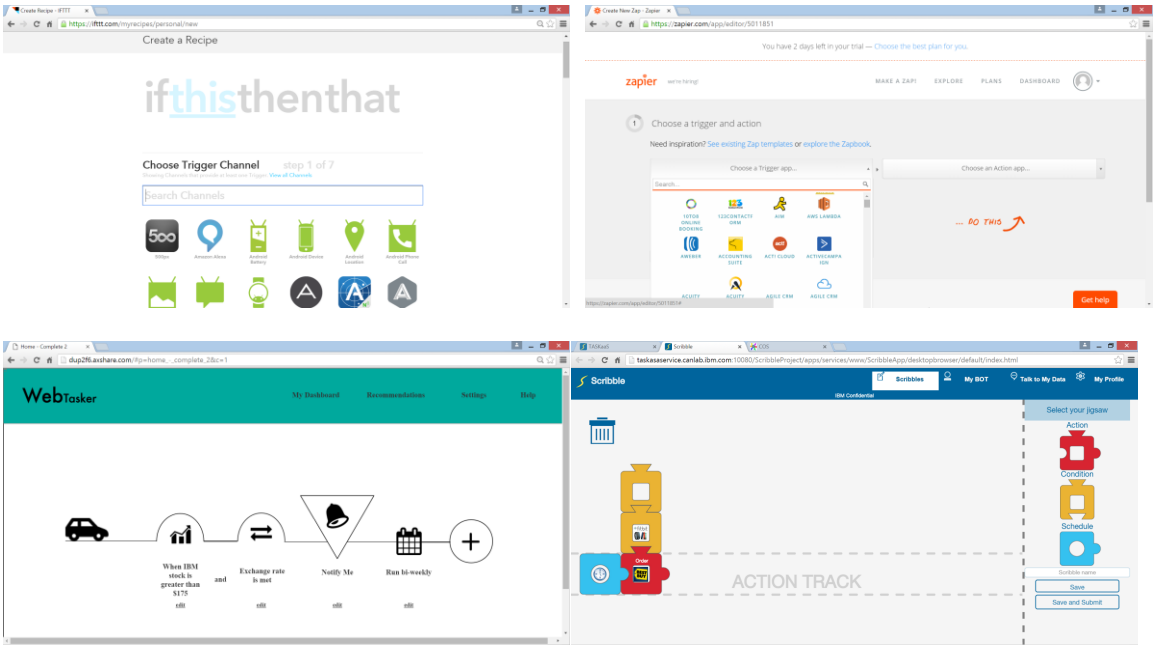


Figure 29: Interfaces (clockwise from top left) IFTTT, Zapier, Scribble, WebTasker

5.3 Experimental Design

The experiment followed a balanced repeated measure design. All participants were tested using all interfaces and completing all tasks at all complexity levels. Summary of the experimental design:

- 8 tasks (per interface) by
- 4 interfaces by
- 2 levels of complexity.

All factors are within-subject or repeated measures factors (because they represent repeated measurements on the same participant). Each trial lasted approximately 2 hours.

5.3.1 Independent Variables

Independent variables manipulated under this experiment included the four web tasking interfaces and two levels of task complexity (high and low). The order of presentation of the interfaces, order of tasks, and gender were partially counterbalanced.

The order of conditions is shown in Table 8.

Table 8: Order of conditions of study

Trial order #	Interface Order				Task Order		Gender	Computer Prog Exp
1	IFTTT	WebTasker	Scribble	Zapier	High	Low	Female	Some
2	Zapier	IFTTT	WebTasker	Scribble	High	Low	Male	None
3	Scribble	Zapier	IFTTT	WebTasker	High	Low	Female	None
4	WebTasker	Scribble	Zapier	IFTTT	High	Low	Male	Some
5	Zapier	Scribble	WebTasker	IFTTT	Low	High	Female	Some
6	IFTTT	Zapier	Scribble	WebTasker	Low	High	Male	None
7	WebTasker	IFTTT	Zapier	Scribble	Low	High	Female	None
8	Scribble	WebTasker	IFTTT	Zapier	Low	High	Male	Some
9	IFTTT	WebTasker	Scribble	Zapier	Low	High	Male	Some
10	Zapier	IFTTT	WebTasker	Scribble	Low	High	Female	None
11	Scribble	Zapier	IFTTT	WebTasker	Low	High	Male	None
12	WebTasker	Scribble	Zapier	IFTTT	Low	High	Female	Some
13	Zapier	Scribble	WebTasker	IFTTT	High	Low	Male	Some
14	IFTTT	Zapier	Scribble	WebTasker	High	Low	Female	None
15	WebTasker	IFTTT	Zapier	Scribble	high	Low	Female	None
16	Scribble	WebTasker	IFTTT	Zapier	high	Low	Female	Some

Under this design, participants completed 8 tasks (4 high and 4 low complexity) under one interface before moving to the other interface.

Participant's gender and programming experience were also independent variables.

5.3.2 Dependent Variables

The dependent variables of this experiment were be the following:

- Task Timing – the total amount of time to complete the task.
- Errors – a frequency count and classification of errors in task completion, such as incorrect menu or app selections, input of information in wrong field, etc.
- Usability score – System Usability Scale (SUS) questionnaire composed of 10 statements that are scored on a 5-point scale of strength of agreement. Final scores for the SUS can range from 0 to 100, where higher scores indicate better usability. Usability is measured along three dimensions: effectiveness, efficiency, and satisfaction. SUS actually measures two factors, usability and learnability (Lewis and Sauro, 2009). The 10 SUS statements are in Appendix A.4 SUS Questionnaire. The SUS questionnaire was programmed in C# and presented to the user after

every interface using a laptop thereby capturing participants' entries in real time. A screen shot of the questionnaire is provided in Figure 30.

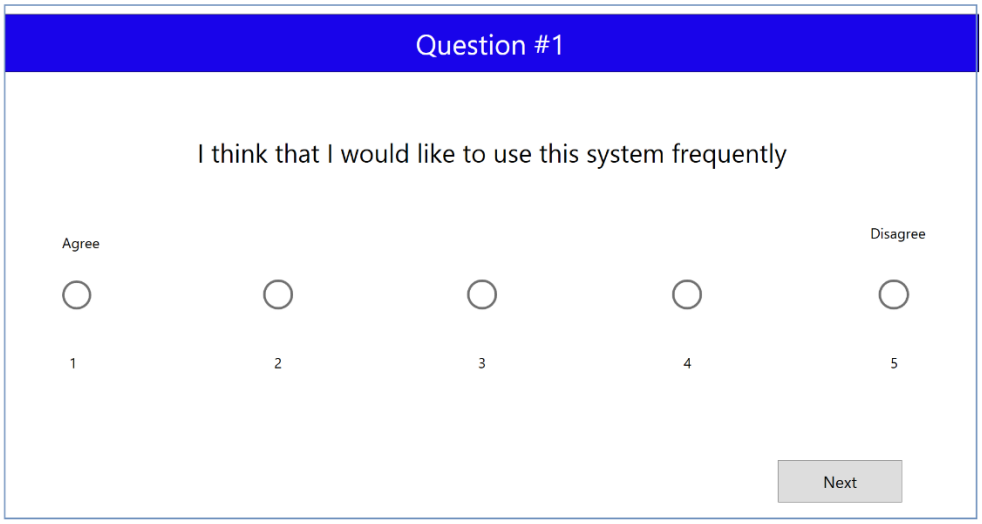


Figure 30: SUS questionnaire screenshot example

- Overall Likert Scale Rating – this rating was added at the end of the SUS questionnaire. The intent of this question is to provide a qualitative answer that can be used in conjunction with a SUS score to better explain the overall experience when using the SUS to summarize a user interface's usability (Bangor et. al, 2008). It was also used to compare answers and comments given.as part of the de-briefing (i.e. what is your favourite interface?). It used a 7-point scale with qualitative descriptors (see Figure 31).
- Table 9 shows the statistical plan summary.

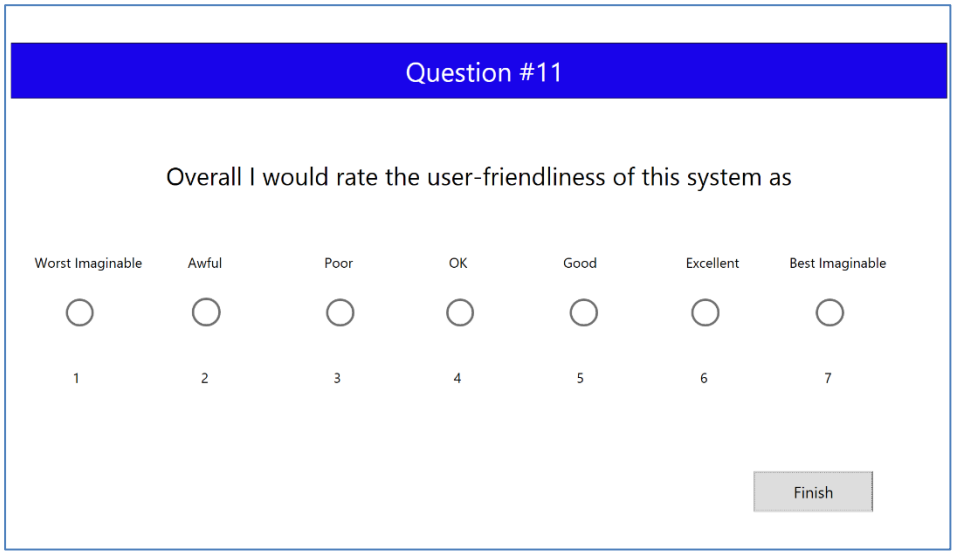


Figure 31: Likert Scale rating appended to SUS questionnaire

Table 9: Statistical Plan Summary

Measure	Method	Analysis
Task Time	Timing from timer on computer.	Descriptive statistics comparison. Parametric statistics comparing tasks and complexity levels.
Errors	Count & classification from experimental trials. Screen capture using software.	Descriptive statistics
Usability Score	SUS Questionnaire Overall Likert Scale Rating	Descriptive statistics comparison. Parametric statistics comparing tasks across interfaces.

5.4 Procedure

The participants were given a study information letter and asked to sign a consent form. They were then briefed individually on the nature of the test. Appendix A.2 shows the information letter, consent form, and briefing script read to the participant. They were told that this is a web tasking project about interface development and will be using a standardized briefing protocol and informed consent was obtained. Participants were tested one at a time on the same computer and computer monitor. All trials were recorded with screen capture software, Camtasia Studio[10] software.

[10] www.techsmith.com/camtasia.html

As part of the study briefing, a short demographics questionnaire (Appendix A.3) was then filled out by the experimenter. Participants had a list of tasks in front of them (Appendix A.1), the experimenter would provide the correct page at the right time for each trial.

Experimenters did not respond to any questions while participants were conducting their tasks, but were allowed to interact with the participants between each interface to offer any answers regarding that specific interface.

Testing was conducted by a team of two experimenters to ensure accurate protocol implementation and data collection, including interface presentation, timing measurements, and error recording.

Each participant was paid $20/hour (pro-rated to the nearest half hour) for his/her participant in the study. Most (13 out of 16) participants took 2 hours to complete the entire study and the remainder took approximately 1.5 hours.

5.4.1 Timing

Experimental trials had a time limit of interacting with the web tasking interfaces. The high complexity tasks had a time limit of 5 minutes and the low complexity tasks had time limit of 2 minutes. These time limits were set based on preliminary findings with the pilot study.

5.4.2 Debriefing

Upon completion of the interface trial, each participant spent a few minutes to fill out a Systems Usability Scale (SUS) Questionnaire (Appendix A.4). This was administered by the facilitator on the computer. The participant had the opportunity to give any verbal feedback on the platforms and this was recorded as qualitative data. The debriefing questionnaire is in Appendix A.5, and the results are presented in Section 5.9.

5.5 Analysis of Errors

Errors were recorded and analyzed for each trial for each interface. Six categories of errors emerged from these records: 1) Typo Submission, 2) Selection of the Item, 3) Entry of Data in the Wrong Section, 4) Severe errors, redid 5) Time Out Errors 6) Other. These categories were developed to facilitate the error analysis and were not mutually exclusive (i.e. several different types of errors could occur in a single trial). The majority of individual trials were error free. The descriptions below detail what was involved in each category of error.

5.5.1 Typo Submission Error

A typo was counted if the participant entered information required to complete a task with a misspelling of a word or number occurred due to an addition, absence or reversal of the letters, digits, or symbols. For example, for the task "if the IBM stock prices rises above $160, then send an email from Gmail", a typo submission error was counted if the participant typed in the "$" in the dollar amount field, for the task "if the IBM stock prices rises above $160, then send an email from Gmail", an typo submission error was counted if the participant typed in the "$" in the dollar amount field, as the system would not accept the data with the "$" symbol and would return an error.

5.5.2 Selection of the Wrong Item

Selection of the wrong menu item involved selecting the incorrect item by accident or as a genuine mistake. For example, participants made this error were during selection adding an action piece instead of a condition piece in the Scribble interface.

5.5.3 Entry of Data in the Wrong Section

Entry of data in the wrong section was chosen as a category for those occasions where participants attempted to enter information in the wrong place. For example, entering condition information in an action field.

5.5.4 Severe Errors, redid

Severe errors occurred when participants were not able to recover from their mistakes and had to restart the trial. The severe errors occurred when a participant got lost on the interface or clicked away from the desired page and had to return to the appropriate page (or homepage) to start the task again.

5.5.5 Time Out Errors

Each trial had a time limit for visually interacting with the interface for 5 minutes for high complexity tasks and 2 minutes for low complexity tasks. When the trials were not completed within the time limit, the trial was assigned the maximum value. These trials were not redone once the maximum time limit was reached. Most of trials with a time out error were observed to have at least one other type of error in addition to the time out error.

5.5.6 Other

This category was created for the rare errors that did not fit the other categories, so they have been grouped together here. For example, if a user forgot to include some information to complete the

task (e.g. almost submitting the task but forgot to set the schedule) and did not time out or submit the task with error, then remembered to add the information; this type of slip error was counted here.

5.6 Analysis of Results

Descriptive statistics and Analysis Of Variance (ANOVA) and post hoc tests for all task times in each interface were performed using Statistica 64 software. The ANOVAs compared tasks and complexity levels. Tukey's HSD (honest significant difference) post-hoc tests were used because of its statistical power and widely accepted use (Kromrey and La Rocca, 1995). Errors were also recorded and were provided in a frequency count. Descriptive statistics were also calculated with Statistica 64.

5.7 Usability Results

ANOVAs were conducted to examine the experimental factors: gender (male vs. female), computer programming experience (some vs. none), and task complexity (high vs. low). Where significant effects were found with more than two levels, post-hoc Tukey's tests were used to identify differences between conditions. Box and whisker plots are provided for task timings that show the mean, the "box" shows the standard error (the standard error of the mean is the theoretical standard deviation of all sample means), and the "whiskers" show the standard deviation (measure of variation). Refer to Appendix B for ANOVAs, and post hoc tests. Descriptive statistics results are presented in each subsequent section.

5.7.1 IFTTT Results

The ability of the participant to successfully complete a task was determined by the task time. An ANOVA analysis was conducted for differences in participants' task time under different conditions based on task and complexity. Figure 32, Table 10, and Table 11 show the results of the mean and standard deviation of the task time (in minutes) for IFTTT. The mean for all high complexity tasks was 1.15 minutes and for low complexity it was 0.96 minutes. Significant factors were observed for task times, $F(7, 56)= 3.14$, $p=0.007$. No interaction effects were observed. No significant difference was observed between genders. The post-hoc Tukey's test revealed that there was:

- Generally no significant difference between low level and high level tasks, except in these cases:
    - H1 and L2
    - H1 and L3
- No significant difference within the low complexity tasks (L1 to L4)

- Significant difference between high complexity tasks H1 and H3 (slight learning effect).

As seen in Figure 32, there does not appear to be a discernable different between high and low level complexity tasks. Therefore, a slight learning effect was observed, since H1 and H2 decreased and there was a significant difference between H1 and H3. The mean for H2 (1.08 minutes) was less than the mean for H1 (1.46 minutes). However, H4 yielded a higher task average task time than H2 and H3. This could be attributed to the additional typing to enter information to complete this task (i.e. participants had to type in a stock ticker symbol and an email address, where they did not have to do so in the other tasks).

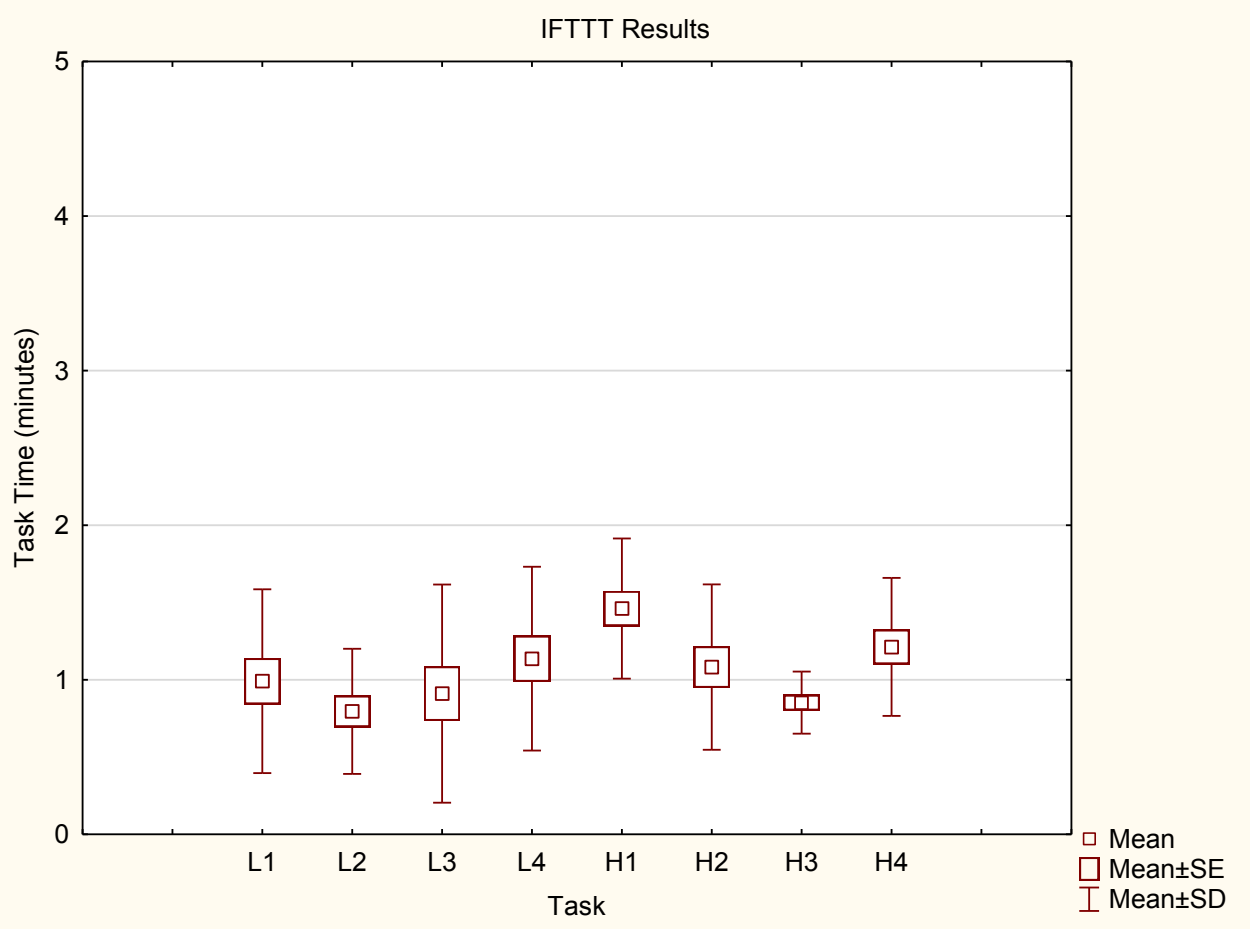


Figure 32: IFTTT Results

Table 10: IFTTT Results

Variable	Descriptive Statistics (IFTTT)							
	Mean	Valid N	Median	Mode	Minimum	Maximum	Std.Dev.	Variance
H1	1.46	16	1.33	1.25	0.88	2.57	0.45	0.21
H2	1.08	16	0.95	multiple	0.67	2.95	0.53	0.29
H3	0.85	16	0.81	multiple	0.58	1.30	0.20	0.04
H4	1.21	16	1.08	1.08	0.63	2.28	0.45	0.20
L1	0.99	16	0.93	2.00	0.30	2.00	0.59	0.35
L2	0.80	16	0.71	multiple	0.35	2.00	0.41	0.16
L3	0.91	16	0.59	.42	0.22	2.00	0.71	0.50
L4	1.14	16	1.03	2.00	0.50	2.12	0.59	0.35

Table 11: IFTTT Results by Task Complexity

Variable	Aggregate Results Descriptive Statistics (IFTTT)						
	Task Complexity	Mean	Valid N	Median	Minimum	Maximum	Std.Dev.
Task Time	H	1.151823	64	1.041667	0.583333	2.950000	0.472573
Task Time	L	0.958333	64	0.700000	0.216667	2.116667	0.584500

5.7.1.1 Error Results

Error frequencies were recorded for the 6 error categories. Figure 33 shows the error frequency results. Relative to the other interfaces, IFTTT had the least number of errors overall. In general, more errors occurred during the high complexity tasks. There were no time out errors for the high complexity task but 8 timeouts in the low complexity. These errors were from three different participants, the main reason being s/he was not aware of the search bar to complete the search task and conducted the search by click exploration (e.g. clicking on different categories, recommendations, popular recipes, etc.).

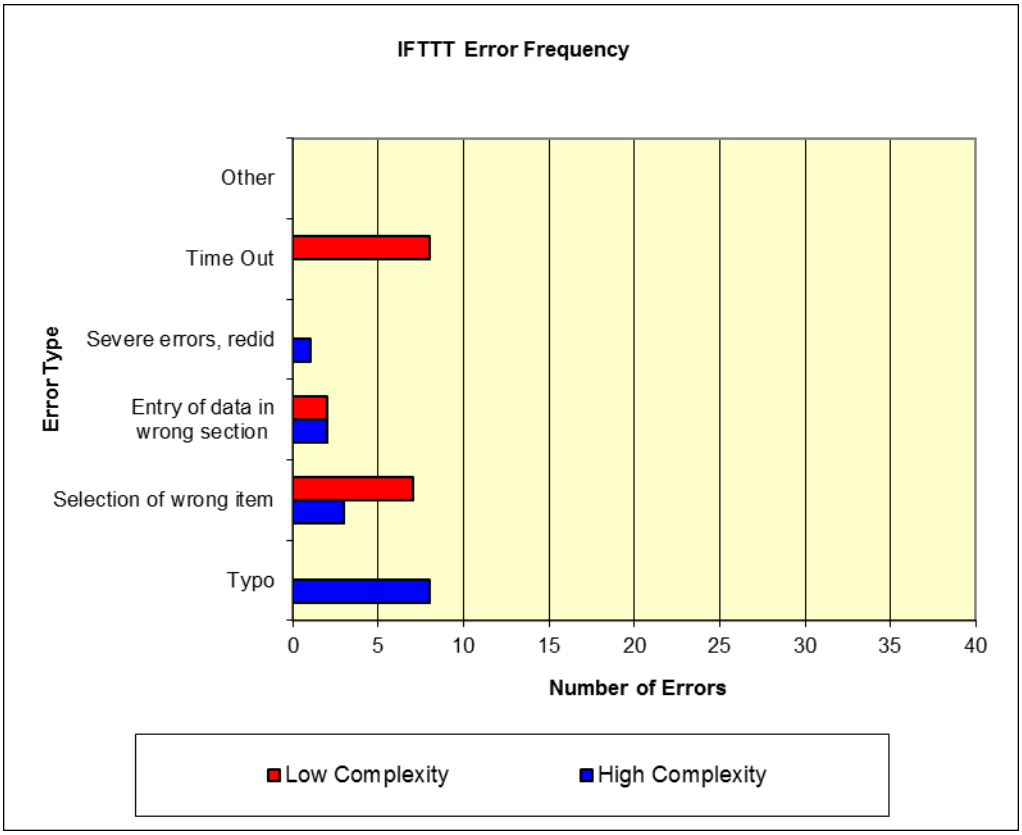


Figure 33: IFTTT Error Frequency

5.7.2 Scribble Results

Figure 34, Table 12, and Table 13 show the results of the mean and standard deviation of the task time (in minutes) for Scribble. The mean for all high complexity tasks was 3.45 minutes and for low complexity it was 0.41 minutes. The high complexity task, H2, had the greatest variability as indicated by the larger standard deviation. Significant factors were observed for task times, $F(7, 84) = 350.48$, $p=0.00$. No interaction effects were observed. No significant difference was observed between genders. The post-hoc Tukey's test revealed that there was:

- Significant difference between low level and high level tasks (i.e. high level tasks took longer)
- No significant difference within the low complexity tasks (L1 to L4)
- Significant differences between all high complexity tasks (H1 to H4)
- Significant differences between H1 to H3 (i.e. evidence of learning effect).

As seen in Figure 34 there appears to be a learning effect as task time decreases between H1 to H3. However, the task time for H4 increases, not continuing the learning trend. This was anticipated as the H4

task was to create a web task with one condition and associated action and another set of one condition and associated action (in one web task). H1 and H3 either had one or two conditions and one related action. H4 had more steps to carry out, in addition to the user learning to compose a web task that consisted of multiple conditions and actions. H2 included to add a schedule component to the web task. This may be why there is greater variability in this task, as users were learning how to set the schedule.

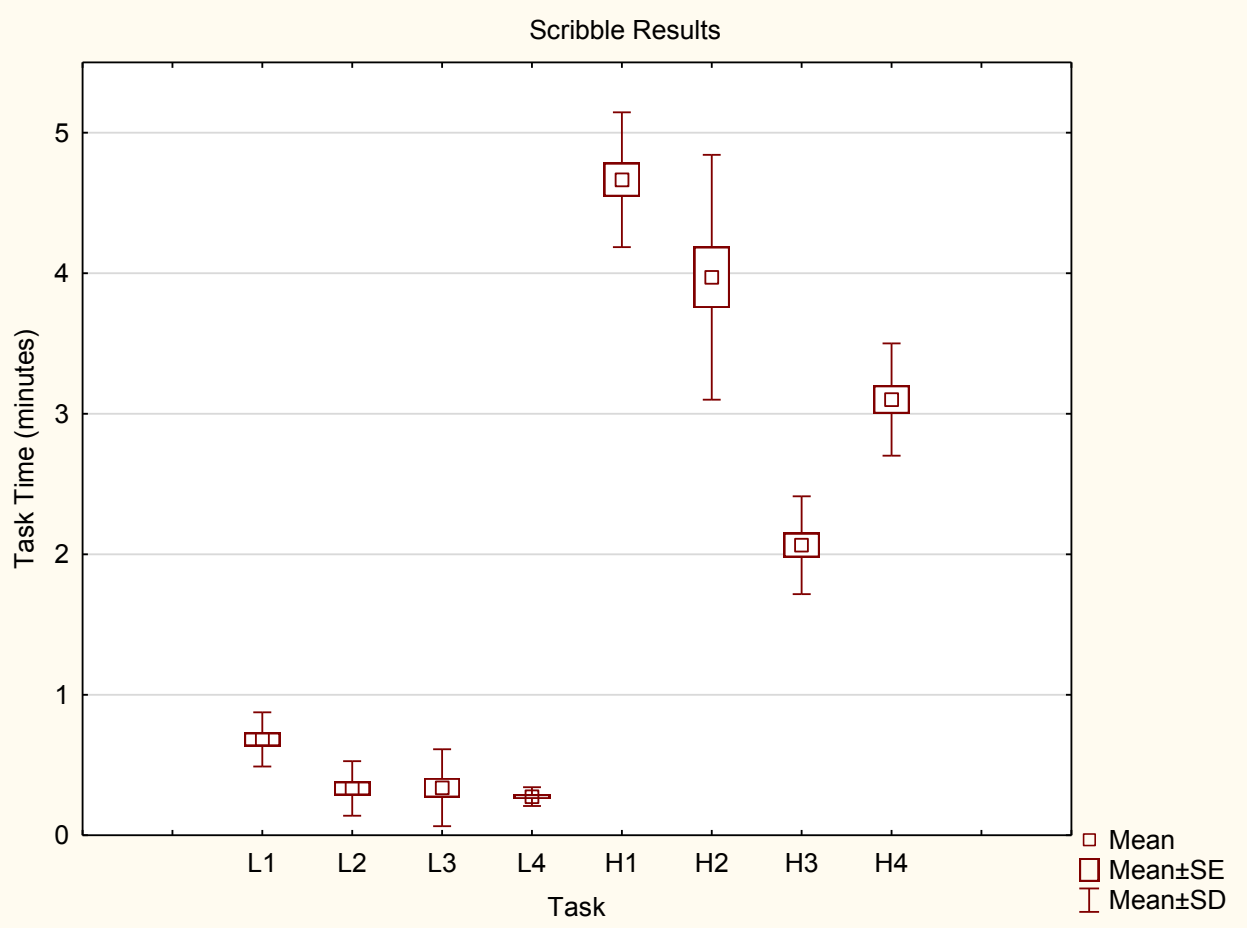


Figure 34: Scribble Results

Table 12: Scribble Results

Variable	Descriptive Statistics (Scribble in Scribble data)						
	Mean	Valid N	Median	Minimum	Maximum	Std.Dev.	Variance
H1	4.67	16	5.00	3.73	5.00	0.48	0.23
H2	3.97	16	3.99	1.85	5.00	0.87	0.76
H3	2.06	16	2.05	1.27	2.77	0.35	0.12
H4	3.10	16	2.98	2.50	3.75	0.40	0.16
L1	0.68	16	0.65	0.33	1.02	0.19	0.04
L2	0.33	16	0.27	0.17	0.77	0.19	0.04
L3	0.34	16	0.28	0.17	1.33	0.27	0.08
L4	0.28	16	0.28	0.17	0.42	0.07	0.00

Table 13: Scribble Results by Task Complexity

Variable	Aggregate Results Descriptive Statistics (Scribble Data2)							
	Task Complexity	Mean	Valid N	Median	Minimum	Maximum	Std.Dev.	Variance
Task Time	H	3.45	64.00	3.43	1.27	5.00	1.12	1.27
Task Time	L	0.41	64.00	0.32	0.17	1.33	0.25	0.06

5.7.2.1 Error Results

Figure 35 shows the error frequency for Scribble. Scribble had the most errors overall, mostly in the high complexity trials. This aligns with the results from the pilot study as described in Section 4.2.1.3. There was only one error (a typo) in the low complexity tasks. The most common error was 'selection of the wrong item' during the high complexity tasks. Initially, users did not know how to select an action or condition piece and some observed learning took place. For example, many users did not know they had to drag the condition piece to stack it on the action piece—which led to the user selecting the action piece when the intention was to enter a condition. This was demonstrated in the second highest error, the 'entry of the data into the wrong section'. Scribble also had the highest number of time outs for a high complexity task.

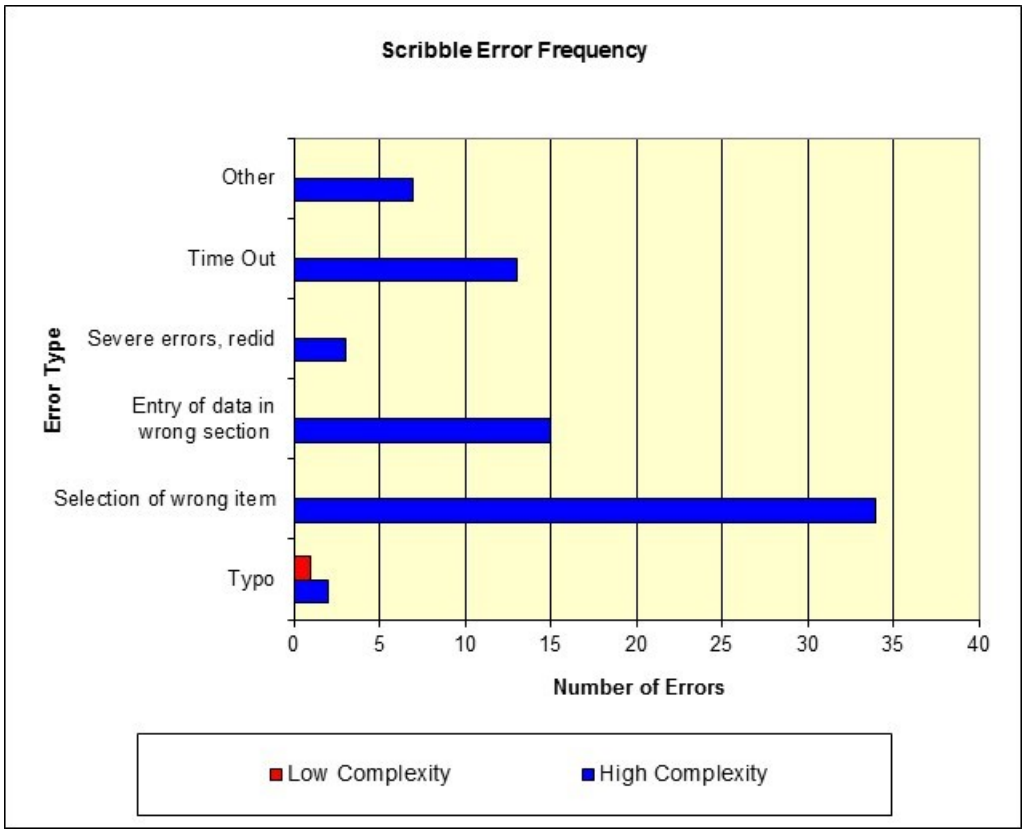


Figure 35: Scribble Error Frequency

5.7.3 Zapier Results

Figure 36, Table 14, and Table 15 show the results of the mean and standard deviation of the task time (in minutes) for Zapier. The mean for all high complexity tasks was 2.23 minutes and for low complexity it was 1.51 minutes. The high complexity task, H2, had the greatest variability as indicated by the larger standard deviation. Significant factors were observed for task times, $F(7, 84)= 24.60$, $p=0.00$. No interaction effects were observed. No significant difference was observed between genders. The post-hoc Tukey's test revealed that there was:

- Significant difference between high complexity tasks
    - H2 was significantly higher than all other tasks (i.e. H2 took the longest)
    - tasks H3 and H4 took less time to complete than H1 and H2 (could be somewhat of a learning effect)
- L4 was significantly different from L1, L2, and L3 (it had the lowest task time). The means of low complexity tasks were high for L1, L2, and L4 with many participants maxing out their time)

- L1 and L4 were significantly different than H2 and L4 was significantly different than H4 (keep in mind that low level tasks max out at 2 minutes).

It is speculated that H2 took the longest for two reasons. The main reason was the H2 task was the only task in the entire study that had a slightly different wording, "Send an email with Gmail for new tweets from @UWHFstudentgirl"; where the action was indicated before the condition. Interestingly that this minor change caused a higher task time, perhaps indicating a higher cognitive load. The second reason was that as there was more information entry required. Users had to enter a twitter ID in addition to an email address.

The search task (low level complexity) in the Zapier interface was evidently difficult for participants. Zapier only allows search by app name, for example, if users would type "tweet" in the search bar, the twitter app would not appear. It is important to note that there were no significant differences (despite the maximum time of low level tasks capped at 2 minutes) between L2 and L3 and H3 and H4. This demonstrates that creating a task is just as "easy" as finding an existing task/zap.

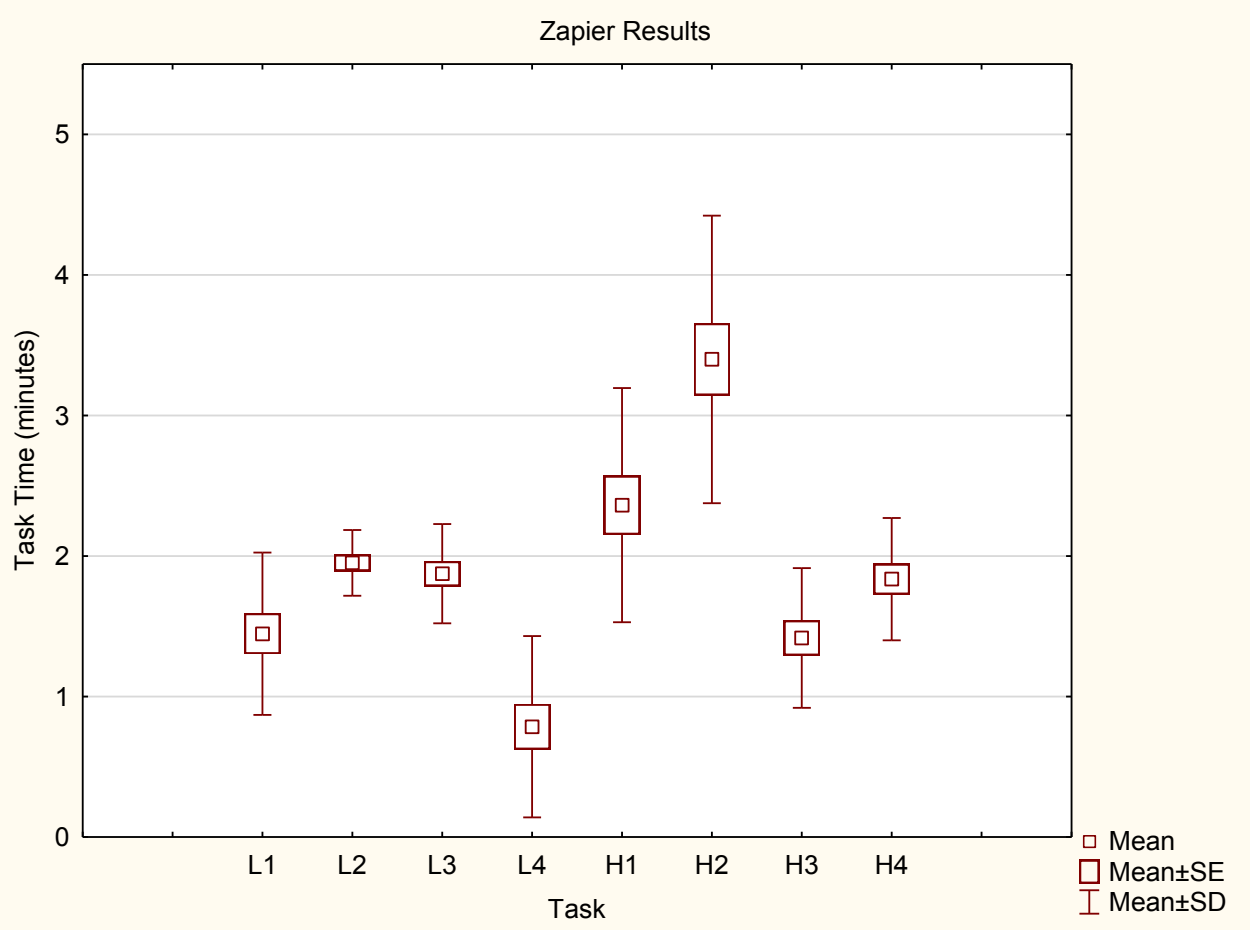


Figure 36: Zapier Results

Table 14: Zapier Results

Variable	Descriptive Statistics (Zapier data)							
	Mean	Valid N	Median	Mode	Minimum	Maximum	Std.Dev.	Variance
H1	2.36	16	2.13	1.93	1.60	5.00	0.83	0.69
H2	3.40	16	3.10	multiple	1.77	5.00	1.02	1.05
H3	1.42	16	1.28	multiple	0.85	2.43	0.50	0.25
H4	1.84	16	1.79	no mode	1.10	2.75	0.44	0.19
L2	1.95	16	2.00	2.00	1.08	2.13	0.23	0.05
L1	1.45	16	1.57	2.00	0.63	2.00	0.58	0.33
L4	0.79	16	0.49	.33	0.32	2.00	0.65	0.42
L3	1.87	16	2.00	2.00	0.62	2.00	0.35	0.12

Table 15: Zapier Results by Task Complexity

Variable	Aggregate Results Descriptive Statistics (Zapier data)					
	Task Complexity	Valid N	Mean	Minimum	Maximum	Std.Dev.
TaskTime (min)	H	64	2.25	0.85	5	1.04
TaskTime (min)	L	64	1.51	0.32	2	0.66

5.7.3.1 Error Results

Figure 37 shows the error frequency results for Zapier. Overall, Zapier had the second highest number of errors. Zapier had the most number of errors for low complexity tasks (i.e. time out errors). This is attributed to a limited search capability in the platform. It is only searchable by app name not zap title, so if a user typed in a keyword it would only find the associated app if the keyword is contained in the app name.

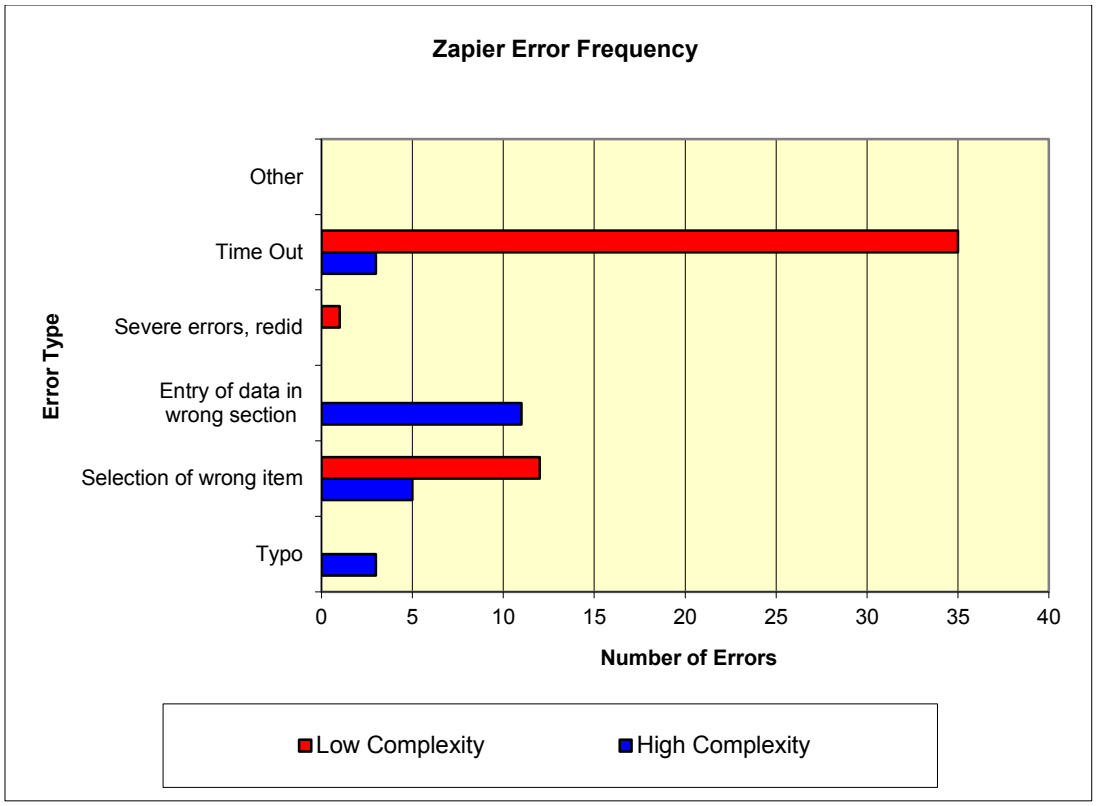


Figure 37: Zapier Error Frequency

5.7.4 WebTasker Results

Figure 38, Table 16, and Table 17 show the results of the mean and standard deviation of the task time (in minutes) for WebTasker. The mean for all high complexity tasks was 2.56 minutes and for low complexity it was 0.28 minutes. Significant factors were observed for task times, $F(7, 84)= 146.45$, $p=0.00$. An interaction effect was observed between computer programming experience and gender. The post-hoc Tukey's test revealed that there was:

- Significant differences between all high complexity tasks, except H1 and H2.
- Significant difference between H1, H2 and between H1 and H3 (task time the lowest for H3). This shows some learning effect.
- Significant difference between H1 (taking the longest) and H4. H4 was a more complicated task.
- H4 was significantly lower than H1 and H2; and H4 was significantly higher than H3.
- Significant differences between all low level tasks and high level.

The results for the WebTasker high complexity tasks were similar to those seen for the Scribble interface. As seen in Figure 38 there appears to be a learning effect as task time decreases between H1 to

H3. However, the task time for H4 increases, not continuing the learning trend. This was anticipated as the H4 task was to create a web task with one condition and associated action and another set of one condition and associated action (in one web task).

H1 and H3 either had one or two conditions and one related action. H4 had more steps to carry out, in addition to the user learning to compose a web task that consisted of multiple conditions and actions.

Significant differences between all low level tasks and high level. This was expected as the search task in the WebTasker interface is not a true searchable platform. The search task is a mock-up of what the ideal search results would yield, thereby making it easy to complete the task. The limitation of the WebTasker prototype explains the low task times for L1 to L4.

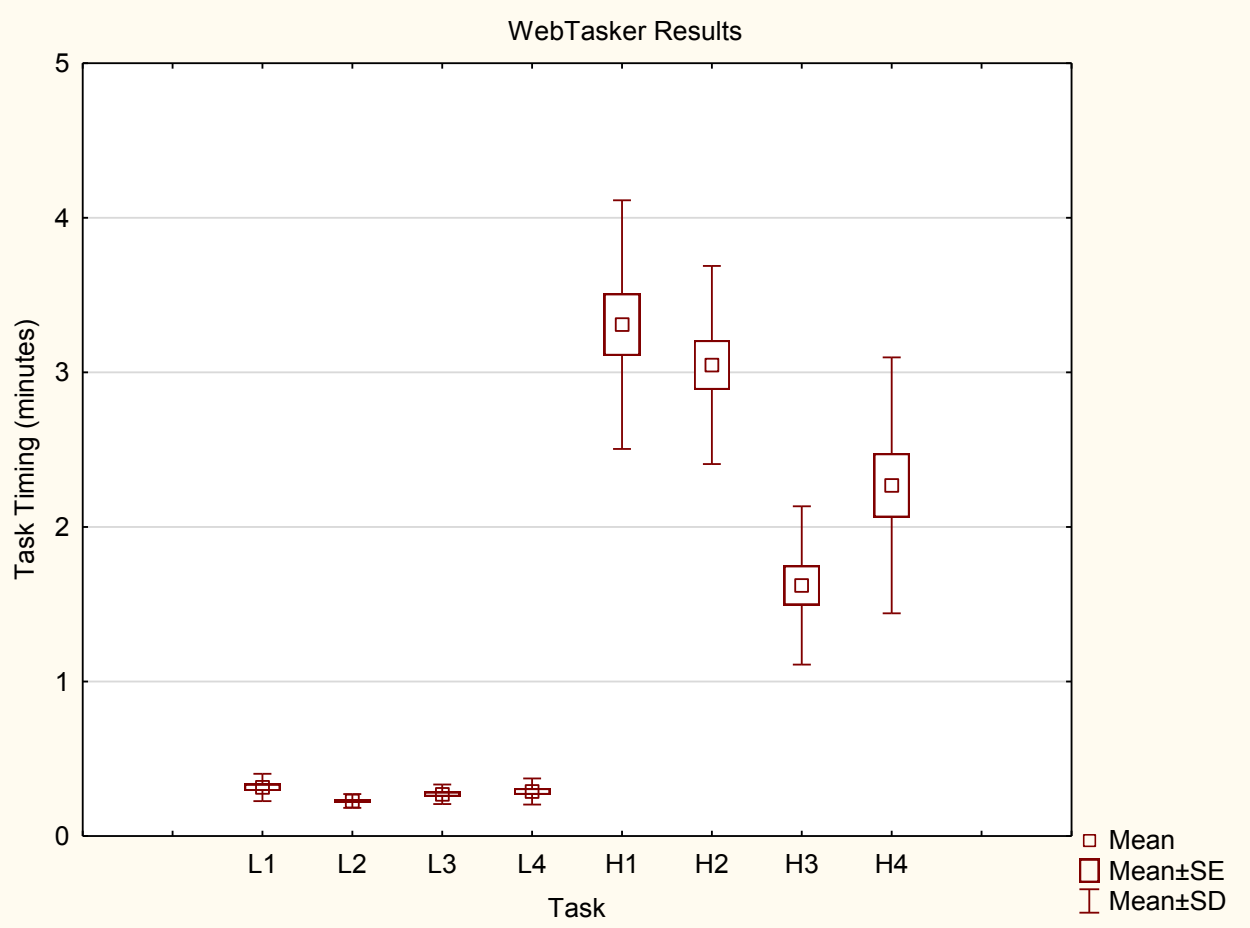


Figure 38: WebTasker Results

Table 16: WebTasker Results

Variable	Descriptive Statistics (WebTasker repeated measures in WebTasker data)							
	Mean	Valid N	Median	Mode	Minimum	Maximum	Std.Dev.	Variance
H1	3.31	16	3.12	multiple	2.42	4.98	0.80	0.65
H2	3.05	16	2.92	2.92	2.27	4.80	0.64	0.41
H3	1.62	16	1.47	multiple	1.17	2.97	0.51	0.26
H4	2.27	16	2.03	multiple	1.57	5.00	0.83	0.69
L1	0.31	16	0.29	.38	0.18	0.50	0.09	0.01
L2	0.23	16	0.23	multiple	0.17	0.32	0.04	0.00
L3	0.27	16	0.28	.28	0.17	0.38	0.06	0.00
L4	0.29	16	0.29	multiple	0.17	0.47	0.08	0.01

Table 17: WebTasker Results by Task Complexity

Variable	Aggregate Results Descriptive Statistics (WebTasker)								
	Task Complexity	Mean	Valid N	Median	Mode	Minimum	Maximum	Std.Dev.	Variance
TaskTime (min)	H	2.56	64	2.50	multiple	1.17	5.00	0.96	0.93
TaskTime (min)	L	0.28	64	0.27	.2833333	0.17	0.50	0.08	0.01

5.7.4.1 WebTasker Interaction Effects

Although factors (gender and programming experience) may not have significant effect when examined individually, they may have a different effect when considered in combination. There was a two-way interaction effect between computer programming experience and gender $F(1, 12)=6.62$, $p=0.024$. The interaction plot in Figure 39 reveals some interesting findings. For the WebTasker interface males with some computer programming experience took longer than males with no computer programming experience. The opposite is true for females, those with some programming experience were faster than those without. Furthermore, a slight 3-way interaction effect, $F(7,84)=2.24$, $p=0.039$, was observed between computer programming experience, gender and task time (Figure 39).

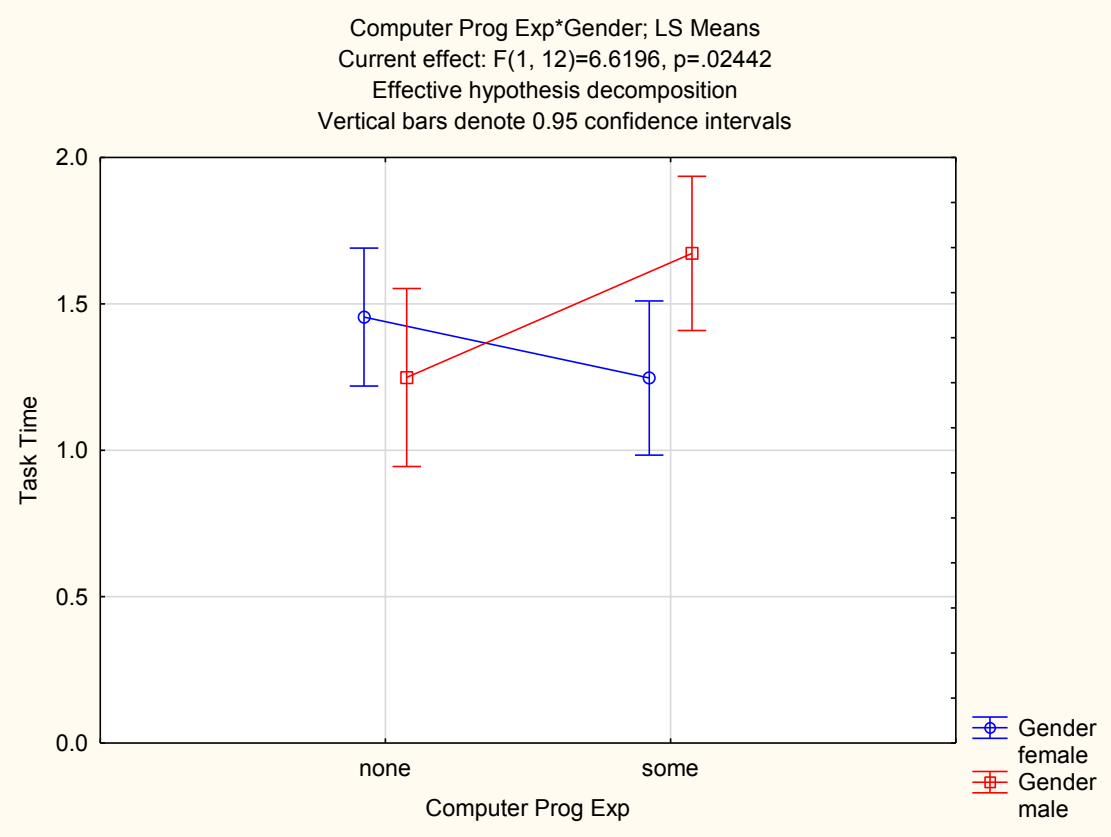


Figure 39: WebTasker Interaction Effect, computer programming experience and gender

There was also a 3-way interaction effect found between task time, computer programming experience, and gender for WebTasker (Figure 40). It appeared that males with computer programming experience had generally longer task times than females with computer programming experience. Females without computer programming experience had longer task times than males without computer programming experience. It is speculated that the interaction effects were spurious, as WebTasker was the only interface that had an interaction effect between gender and computer programming experience. This interaction effect was not seen in other interfaces. Although this interaction effect is speculated to be spurious, it could be partially attributed to the small sample size and disproportion of females to males (sample size was 9 females and 7 males).

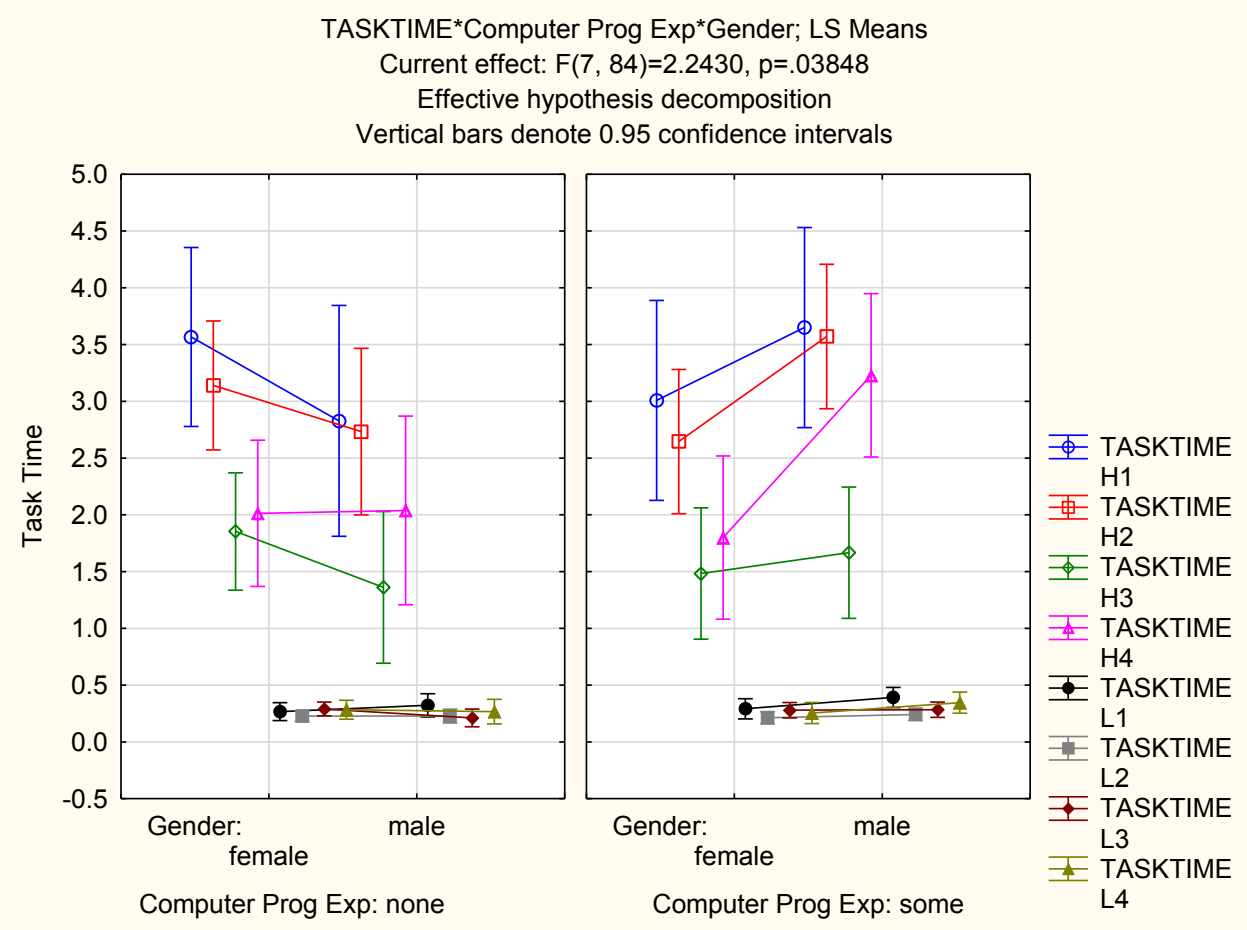


Figure 40: WebTasker Interaction effect for computer programming experience, gender, and task time

5.7.4.2 Error Results

Figure 41 shows the error frequency for WebTasker. WebTasker had the highest number of 'selection of wrong item' error with a frequency of 36 (with Scribble second at a frequency of 34). Many of the 'selection of wrong item' error in WebTasker can be attributed to repeat clicking of an item. Since WebTasker is not a fully functioning prototype, users would often repeatedly click the same icon or link and not receive any feedback (if was not the correct link to click for a specific step in the task). Each repeated click on the same wrong item was counted as an individual error.

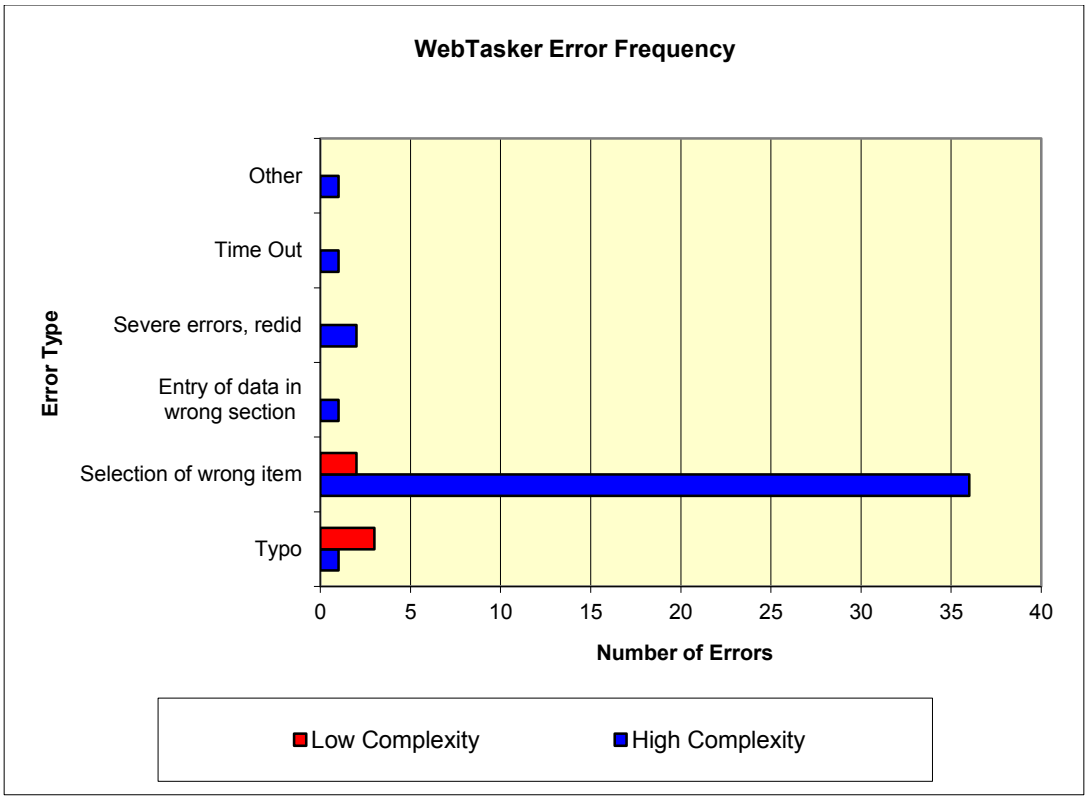


Figure 41: WebTasker Error Frequency

5.8 System Usability Scale and Overall Likert Ratings Results

The tasks used in each interface used in this study were similar in context and composition; however they were not the same tasks across interfaces. Due to this variance, statistically comparing task times across interfaces would not yield comprehensive results. That is why the researcher decided to implement a SUS questionnaire, as a measure that is comparable across interfaces. Participants rated the usability of each interface with this standardized questionnaire. In addition to the SUS scores, a 7-point likert rating was used as another comparable means. If the SUS scores correlated with the likert rating, then this would increase the SUS scores' validity.

Table 18 shows a summary of results of the SUS scores and overall likert ratings. IFTTT received the highest mean SUS score (88.66), followed by WebTasker (85.47), then Scribble (63.13), and Zapier (53.44). The same rank order was found for the overall likert scale rating.

A MANOVA analysis was conducted for differences in participants' SUS scores under each interface. Significant factors were observed for SUS scores $F(6, 98)= 5.28$, $p=0.00$. No interaction effects were observed. No significant difference was observed between genders. The Tukey's post hoc test revealed:

- Significant difference between the IFTTT and Scribble (IFTTT had the highest mean SUS Score).
- Significant difference between IFTTT and Zapier.
- No significant difference between IFTTT and WebTasker.
- WebTasker and Scribble (WebTasker had the highest mean SUS Score)
- WebTasker and Zapier (Zapier had the lowest SUS Score).

Table 18: SUS Score and Overall Likert Rating Results by Interface

Variable	Aggregate Results Descriptive Statistics (SUS questionnaire results)							
	Interface	Mean	Valid N	Median	Mode	Minimum	Maximum	Std.Dev.
SUS Score	IFTTT	88.66	16	93.75	100.00	50.00	100.00	13.88
Overall Likert Rating	IFTTT	5.75	16	6.00	6.00	4.00	7.00	0.68
SUS Score	Scribble	63.13	16	60.00	87.50	20.00	97.50	26.00
Overall Likert Rating	Scribble	4.56	16	4.50	4.00	3.00	6.00	1.09
SUS Score	WebTasker	85.47	16	87.50	multiple	62.50	100.00	10.69
Overall Likert Rating	WebTasker	5.56	16	6.00	6.00	4.00	7.00	0.73
SUS Score	Zapier	53.44	16	45.00	42.50	15.00	85.00	21.73
Overall Likert Rating	Zapier	4.13	16	4.00	4.00	1.00	6.00	1.31

5.8.1 Correlation of SUS Scores and Likert Scale Ratings

A correlation analysis was conducted on the SUS scores and overall likert scale ratings (Figure 42). There was as strong positive correlation between the two measures. This demonstrates the likert scale ratings is supported by the SUS measure.

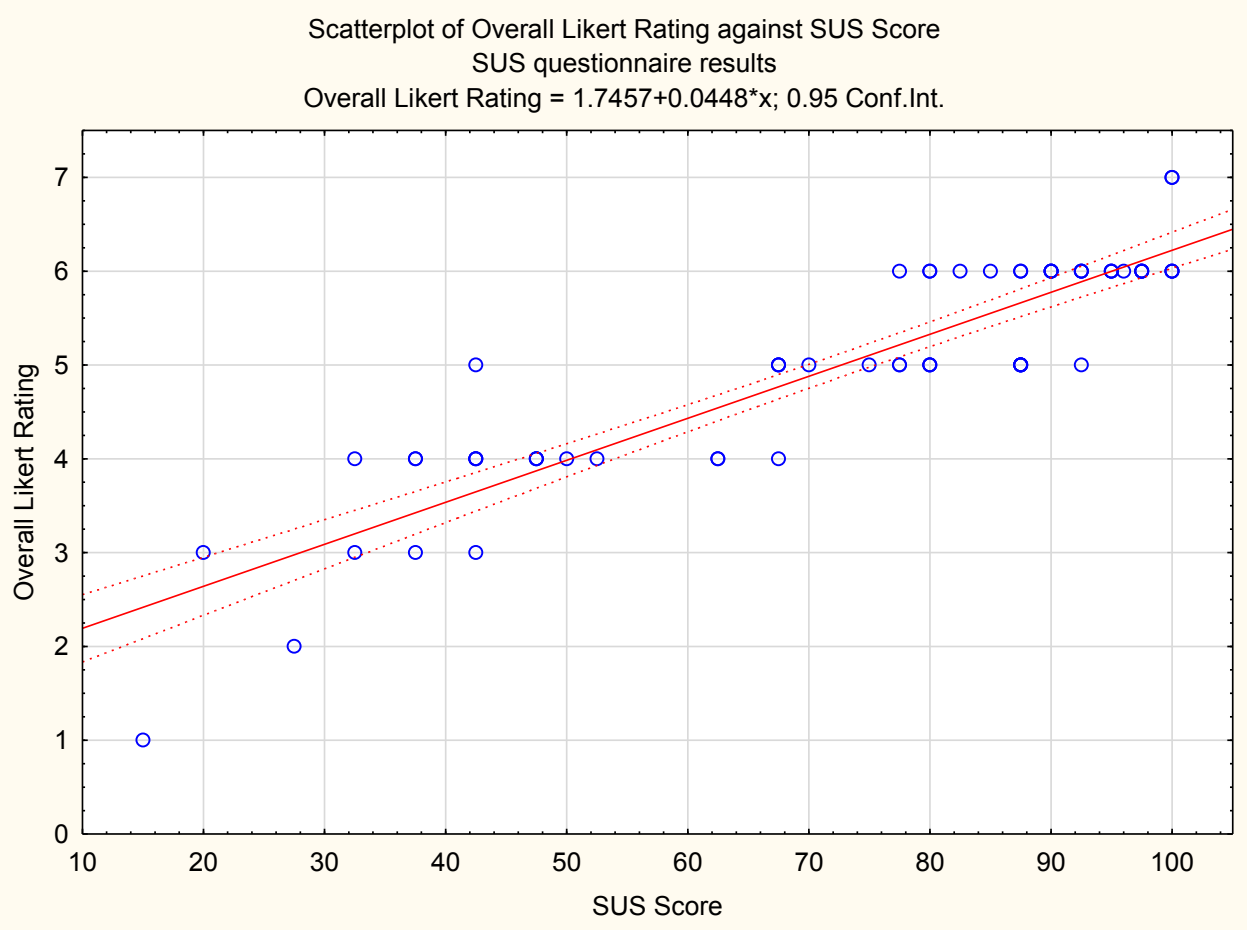


Figure 42: Correlation between SUS Score and Overall Likert

It should be noted that during the debriefing interview, participants' were asked to choose their favourite interface (Question 4, Section Question 4: Favourite Interface). 56% of participants' answers matched their SUS score (the SUS score means are shown in Table 19 and Figure 43). However, of the 44% that did not match their debriefing answer, their second highest SUS score was the interface they chose as their favourite, with a mean difference of 12.5.

Table 19: SUS Scores Descriptive Statistics

Interface	Valid N	Mean	Minimum	Maximum	Standard Deviation
IFTTT	16	88.66	50	100	13.88
Scribble	16	63.13	20	97.5	26.00
Zapier	16	53.44	15	85	21.74
WebTasker	16	85.47	62.5	100	10.69

Figure 43: SUS Score Means

5.8.2 Keystrokes and Mouse Clicks/Scroll Count Correlation to Task Time

The keystrokes and mouse clicks and scrolls were recorded in each trial using Mousotron software. In order to validate the connection between the number of key presses and scrolls to the results obtained from the Usability Study (task times), statistical correlation analysis was conducted. A correlation matrix was generated, representing a value (in the range of -1.00 to 1.00) that reflected the relation between variables (correlation coefficient). Individual task times were used in this analysis, versus an average of these task times (aggregated data). The reasoning behind this was by using more data points the results will reflect a more accurate correlation value, whereas using fewer data points will tend to inflate correlations. Aggregate data also does not consider the distribution of individual participant's performance, and may not accurately reflect true correlations (i.e. grouped and individual data may not agree).

Significant correlations ($p < 0.05$) occurred for all interfaces where r=0.63 for keystrokes and r=0.62 for mouse clicks and scrolls. Figure 44 and Figure 45 plots the correlation. The cluster of points in the lower left corner correspond to the low complexity (search) tasks.

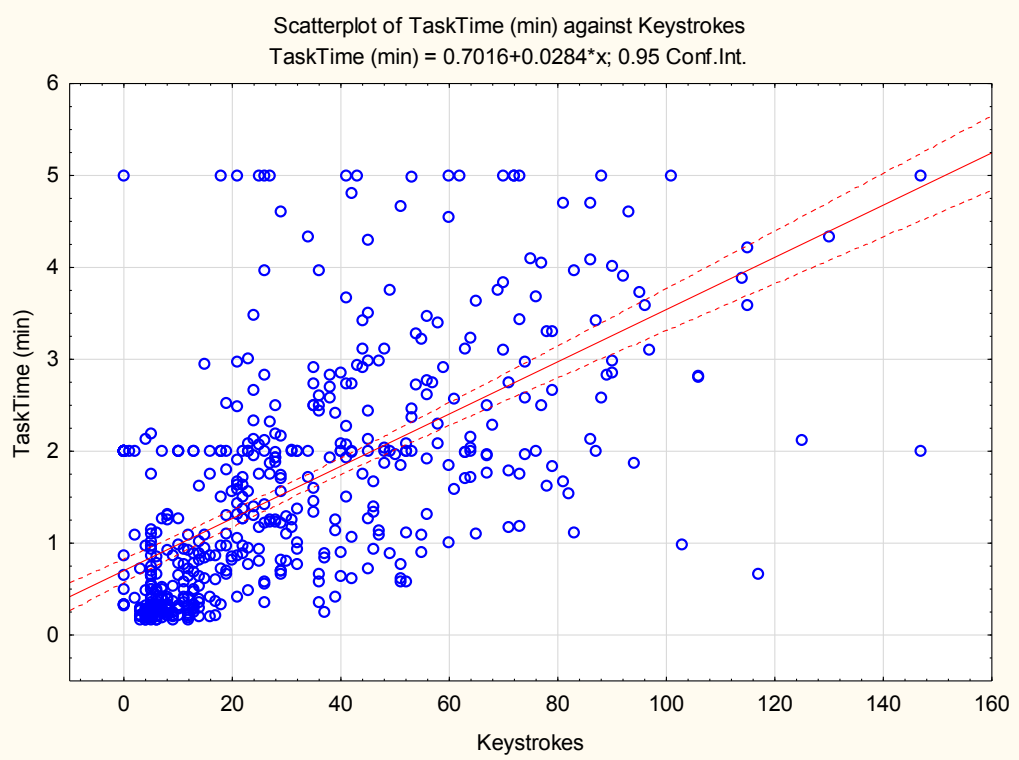


Figure 44: Keystrokes correlation to task time

Figure 45: Mouse clicks and scrolls correlation to task time

5.9 Debrief Questionnaire Results

A four question questionnaire was completed in an interview style at the end of each trial with each individual participant. The subsequent sections present the results the debriefing.

5.9.1 Question 1: Mental Model

The first question was, "Did you have a preference to enter actions or conditions first? If yes, why?". This question was to probe the participant's mental model of the situation. 81% of participants said they would rather enter conditions first, 6% said actions first, and 13% had no preference (see Figure 46). Of the participants who reported they prefer to enter condition first, their reasoning behind it was:

- "It is the temporal order of how things work" or "it is chronological"; similarly, "If that never happens then don't get to the next thing."
- "Conditions should come first- because based on programming, it's better to write conditions first -like an 'if else' statement"
- "It makes sense to define the condition before you tell the system what to do."
- "It seems logical".

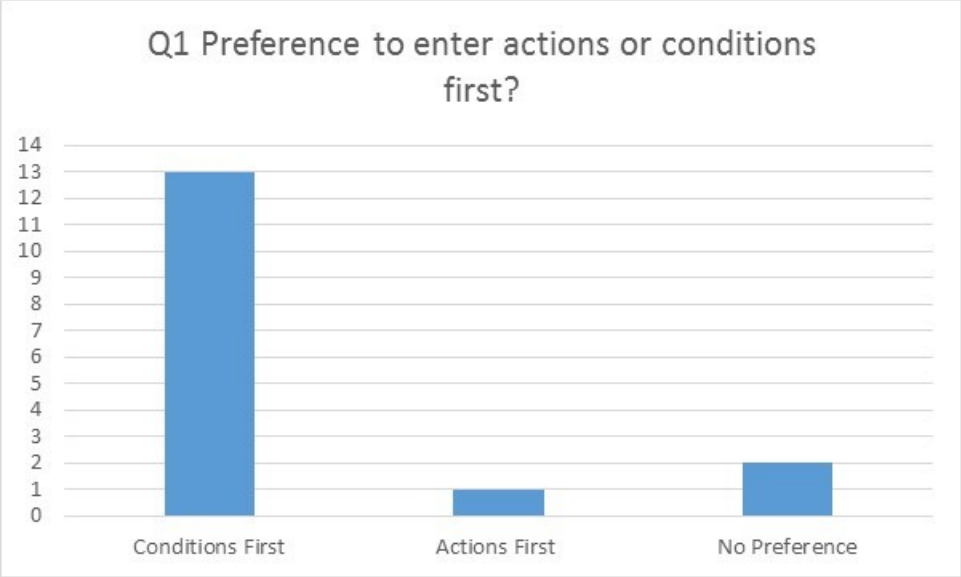


Figure 46: Debrief Question 1 Mental Model

5.9.2 Question 2: Features and Functions

Question 2 was, "What did you like about the functionality and features of each web tasking platform? Is there anything else you would like to see in terms of functionality?".

5.9.2.1 IFTTT

IFTTT received many positive comments during the debriefing (see Figure 47). Most reported that it was easy to use. 25% of participants said it was aesthetically pleasing, and 19% liked it was prescriptive in nature (it prompted the user) and enjoyed the search function. IFTTT did not receive much negative feedback (Figure 48). 19% reported that there was "too much white space". 13% reported it was difficult to find the search function.

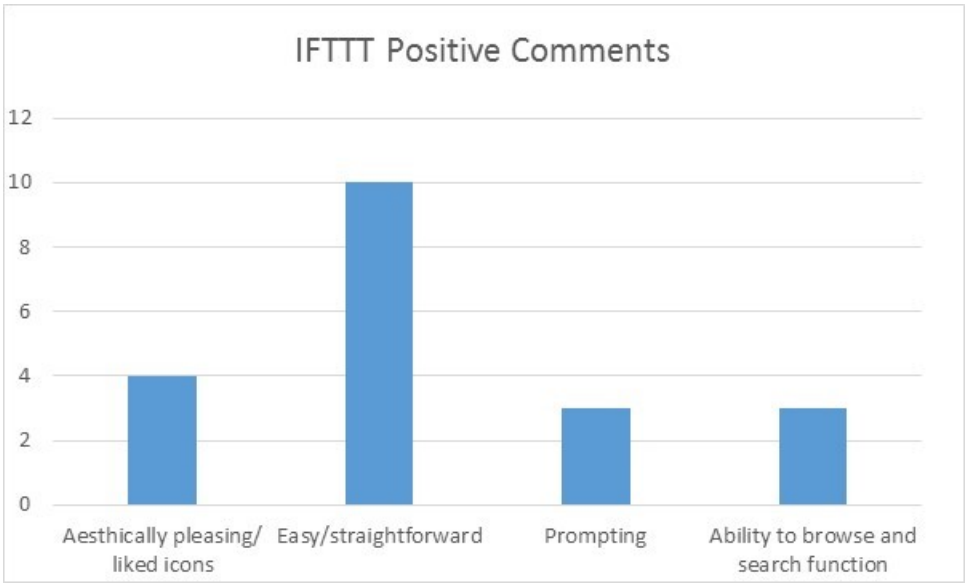


Figure 47: Debrief Question 2 Feature and Function IFTTT Positive

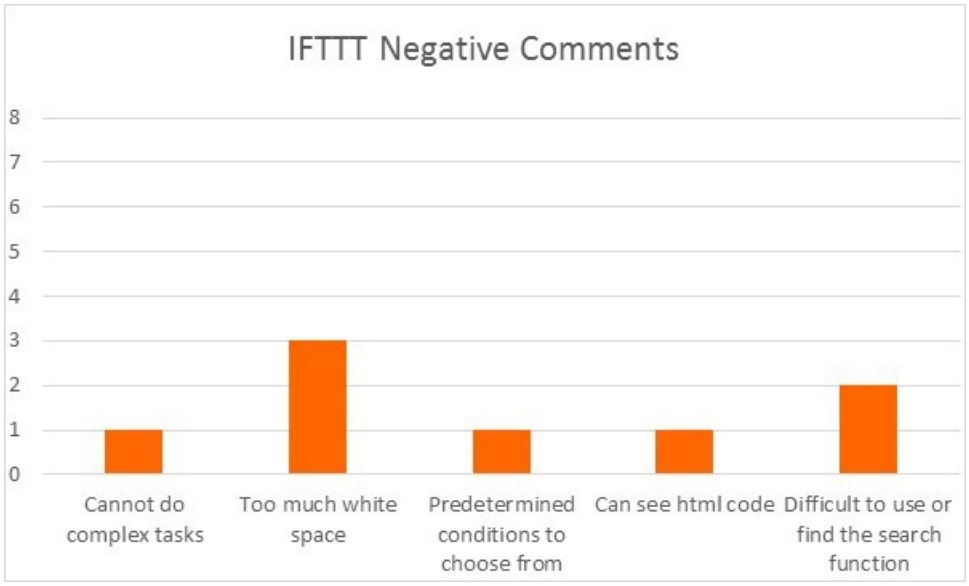


Figure 48: Debrief Question 2 Feature and Function IFTTT Negative

5.9.2.2 Scribble

Scribble had a few positive comments during the debriefing (see Figure 49). 19% liked the graphical or visual nature of the interface. However, this was not particular to the puzzle metaphor, they simply appreciated that the format was presented graphically. Only 13% found it easy to use. Scribble did receive much negative feedback (Figure 50). 31% of participants reported that they did not understand the layout, or reasoning behind having conditions stacked on top of action pieces in the puzzle metaphor. 25% reported that it was not visually appealing.

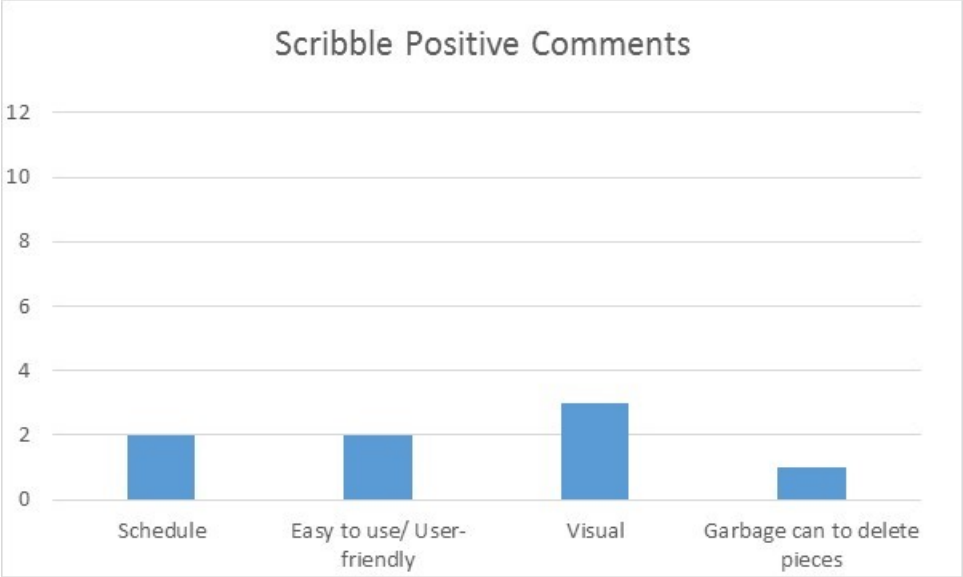


Figure 49: Debrief Question 2 Feature and Function Scribble Positive

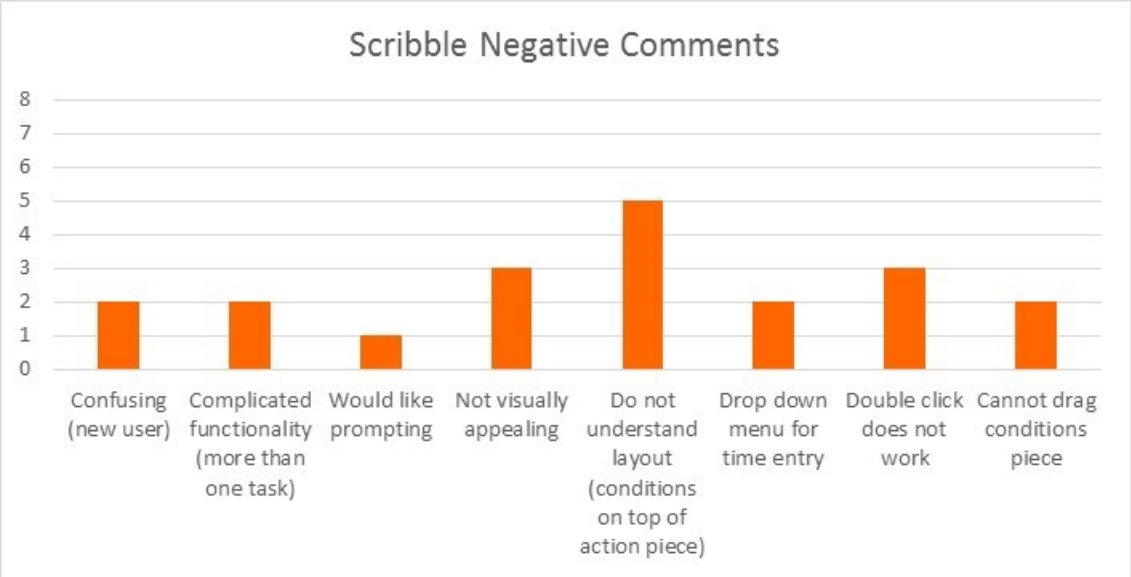


Figure 50: Debrief Question 2 Feature and Function Scribble Negative

5.9.2.3 Zapier

Zapier did not receive many positive comments during the debriefing (see Figure 51). 19% liked that there were many apps available (there are over 400 apps available at the time of this writing). Zapier did receive much negative feedback (see Figure 52). 44% of participants reported that Zapier had a bad search function, as low complexity tasks were difficult to complete. This is in line with the HTA

findings. 31% reported that there was too much information displayed at one time – a finding also in line with the HTA results.

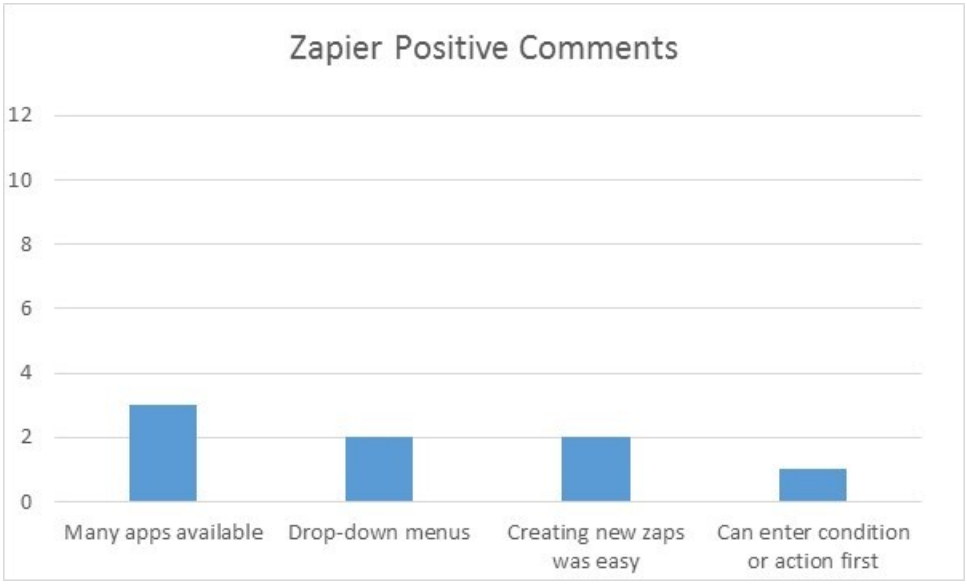


Figure 51: Debrief Question 2 Feature and Function Zapier Positive

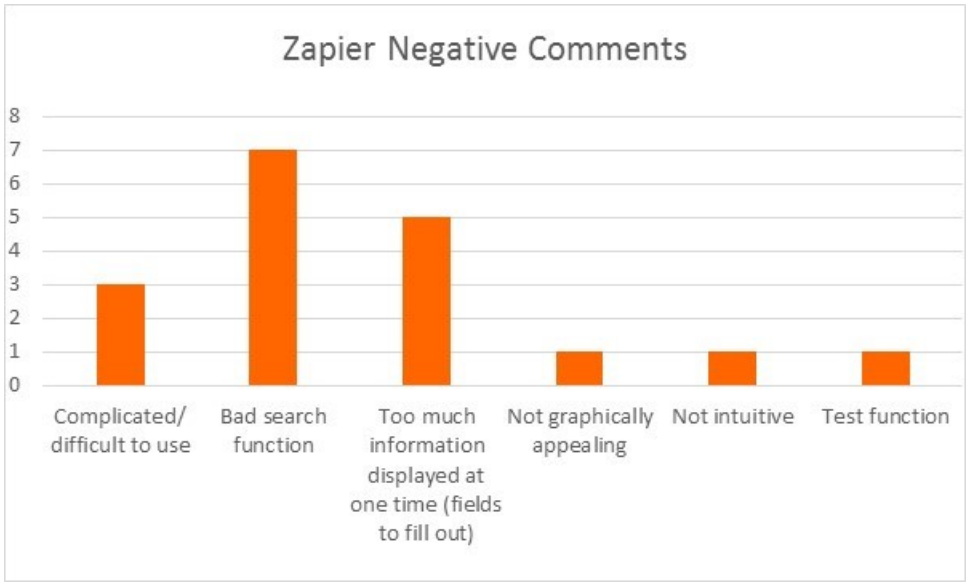


Figure 52: Debrief Question 2 Feature and Function Zapier Negative

5.9.2.4 WebTasker

WebTasker received many positive comments during the debriefing (see Figure 53). 50% of participants reported that it was easy to use and that it was visually appealing. 44% of participants appreciated the linear order and the fact that the order meant something in terms of task execution. WebTasker did not receive much negative feedback (see Figure 54). 25% reported that having the capability to input more than one set of conditions and actions in the same task was complicated. 19% reported it was unfavorable to use arrows for time entry when setting the schedule (arrows were used in the prototype as it was a default time setting feature in the Axure software).

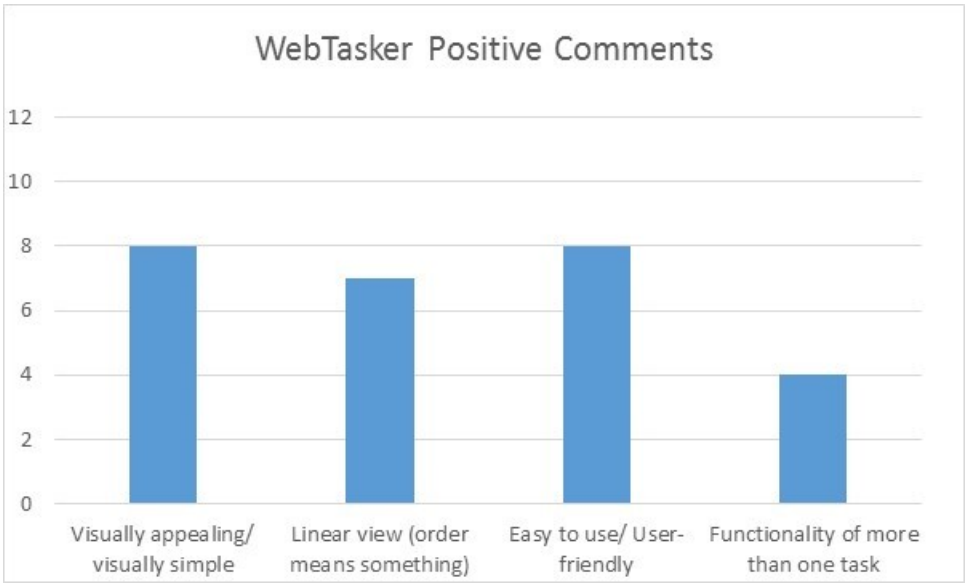


Figure 53: Debrief Question 2 Feature and Function WebTasker Positive

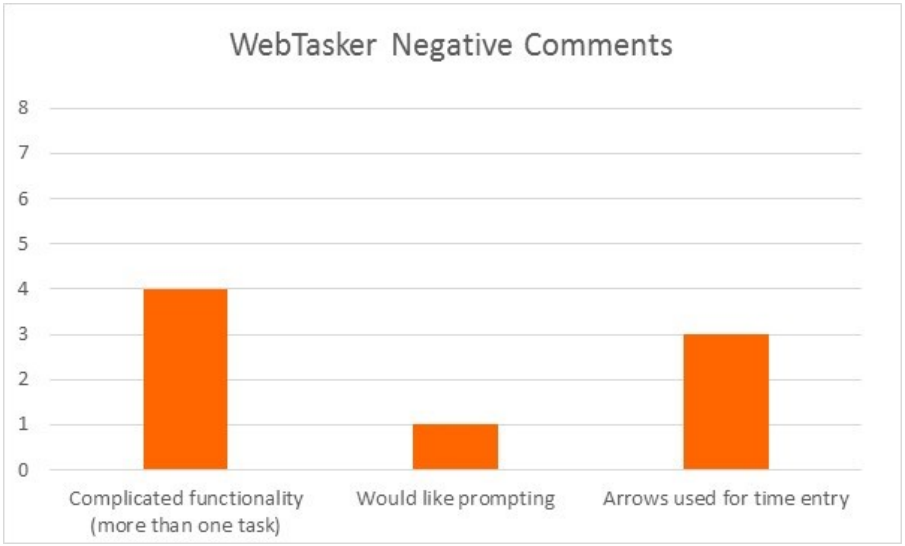


Figure 54: Debrief Question 2 Feature and Function WebTasker Negative

5.9.3 Question 3: Scheduling Feature

Question 3 was, "How could we improve setting the frequency of condition check and setting the task schedule?", which was only applicable to Scribble and WebTasker. 38% of participants said they would not change a thing with it (in either Scribble or the WebTasker interface). 25% reported to have the time field type-able, versus the arrows used in the WebTasker interface and the drop-down menu used to set time in the Scribble interface. Only one person reported they would disable scheduling altogether. Recall that IFTTT and Zapier execute the recipes and zaps on a default basis of 5 or 15 minutes and do not currently have any scheduling capabilities.

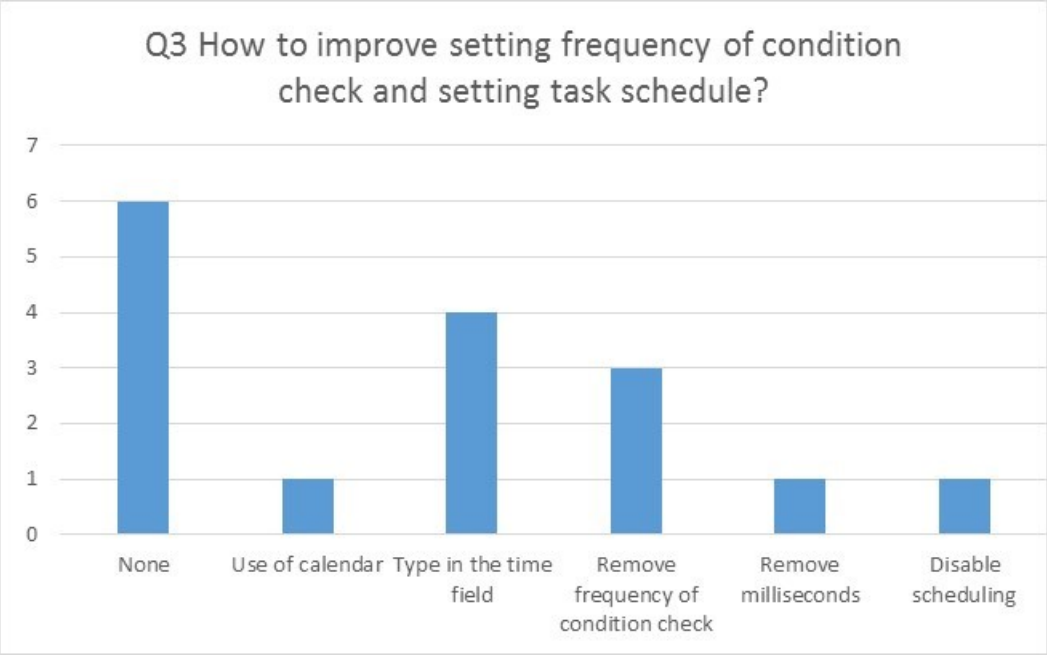


Figure 55: Debrief Question 3 Scheduling Feature

5.9.4 Question 4: Favourite Interface

Question 4 was, "Which interface did you like the most and why?". The most favoured interface was IFTTT, followed by WebTasker. Three participants said they their favourite was both IFTTT and WebTasker and that is why N=19 (instead of 16) in Figure 56. Two people's favourite interface was Scribble.

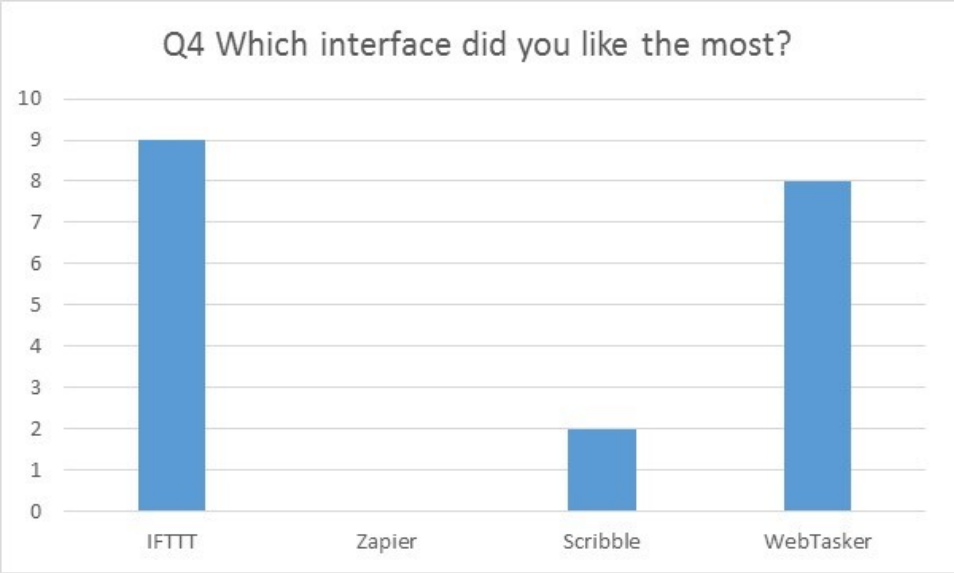


Figure 56: Debrief Question 4 Favourite Interface

5.10 Usability Study Summary

The usability study revealed important information regarding composing web tasks. It was seen that participants can successfully engage in creating web tasks with all web interfaces with one condition and one action. Scribble and WebTasker had the capability to compose web tasks with multiple conditions and actions. Within these two interfaces, participants took longer to compose these web tasks and 13% reported they did not appreciate this added functionality (multiple conditions and/or actions) in the Scribble debriefing and 25% reported this in the WebTasker debriefing.

Generally, participant task times were significantly shorter for low level task complexity tasks except for Zapier, where searching for a zap was comparable to composing a new web task. There were less errors for low level complexity tasks compared to high level complexity tasks. Scribble garnered the most errors overall with the majority of the errors being in the high complexity tasks. Zapier had the most number of errors for low complexity tasks (i.e. time out errors). This is attributed to a limited search capability in the platform.

Users with computer programming experience did not have a significant effect on task time. However, there was an interaction effect noted for WebTasker where it appeared that males with computer programming experience had generally longer task times than females with computer programming experience. Females without computer programming experience had longer task times than males without computer programming experience. Although this interaction effect is speculated to be

spurious, it could be partially attributed to the small sample size and disproportion of females to males (sample size was 9 females and 7 males).

The number of keystrokes, mouse clicks, and scrolls were recorded and were found to be significant to the task times. SUS proved to be an indicative measure in this web tasking interface study, as the results of SUS correlated to the likert scale question. However, only 56% of participants' answers matched their SUS score. Of the remaining 44% that did not match their debriefing answer to their highest SUS score rating, their second highest SUS score was the interface they chose as their favourite, with a mean difference of 12.5.

Chapter 6
Discussion

Several research questions were posed at the introduction of this thesis. Below is a summary of the study results within the context of the research questions.

Part 1: What are the current Human Factors (HF) issues with existing web tasking platforms? Analytical HF analysis through Hierarchical Task Analysis (HTA) was performed on three web tasking interfaces: IFTTT, Zapier, and Scribble. This analysis revealed several HF issues such as freedom for user actions and task structures as demonstrated in linear versus wide HTA structures. Issues identified included: wide menu structures may indicate that there is too much information presented to the user at one time; option to choose next step leaves room for human error (e.g. user inadvertently selecting to enter action first if his intention was to selection condition first), and different interaction types may impact user performance (scrolling versus using a search to find apps). The HTA was a helpful tool in thoroughly investigating each interface and identifying all the user steps at the task level.

Part 2: How do people perceive composing web tasks? What makes up the ideal user control metaphor for web tasking? A pilot usability study was conducted on existing web tasking interfaces and the results were used as input into the researcher's design of a unique web tasking interface, WebTasker. Results of a literature review on mental models, end user programming, and usability guidelines were also sources of design input. An interactive prototype of WebTasker was created based on all the design input. The WebTasker design used a journey line control metaphor, which aimed to minimizes cognitive loading (no need to match pieces to what type, i.e. condition, action, symbol), minimize clicks and scrolls to fill in user information fields, utilized recognized symbols, and had capability to display task execution status to the user.

Part 3: How does the new design, WebTasker, compare to the existing web tasking interfaces? A full scale experimental study was conducted with 16 participants evaluated four web tasking interfaces for approximately 2 hours per session. Each participant performed 4 high complexity tasks and 4 low complexity tasks on 4 different interfaces (32 distinct tasks). Metrics to study end user interaction included: task timings, errors, and ratings from a System Usability Scale (SUS) questionnaire. This usability study showed that participants had poorer performance and found it more difficult to create web tasks in web tasking platforms with multiple conditions and actions. The results showed that there was no difference between performances between participants with computer programming experience than those with none. The results of the SUS ratings highly agreed with the quantitative measures showing that IFTTT received the highest mean SUS score (88.66), followed by WebTasker (85.47), then Scribble

(63.13), and Zapier (53.44). A main takeaway was that WebTasker was the only interface that simulated showing task status (e.g. when first condition was met), and participants feedback indicated this is a feature that they would like to see included in a web tasking interface.

6.1 Mental Models and User Performance

From the result of the debriefing questionnaire, users' preference was to enter conditions first then actions. This complemented the performance metrics used in the empirical study where users had the best performance in IFTTT and WebTasker when composing web tasks (high complexity). The mean for all high complexity tasks was 1.15 minutes in IFTTT and was IFTTT and WebTasker 2.56 minutes for WebTasker. IFTTT had the least number of errors (14 total errors) followed by WebTasker (22 total errors) for high complexity tasks. These results imply that the more prescriptive or constrained decision-making an interface is, there will be less occurrence of human error. In addition, the 'favourite' interfaces were IFTTT followed by WebTasker from the debriefing questionnaire.

6.2 Task Description

It is speculated that the second high complexity task (H2) in Zapier took the longest (had the greatest task time) for two reasons. The main reason was the H2 task was the only task in the entire study that had a slightly different wording, "Send an email with Gmail for new tweets from @UWHFstudentgirl"; where the action was indicated before the condition. Interestingly, this minor change caused a higher task time, perhaps indicating a higher cognitive load. This is speculated to cause a higher cognitive load because the task description did not match a user's mental model of condition first then action. The second reason why this task may have taken the longest was users had to type the twitter ID "@UWHFstudentgirl", thus this task involved slightly more typing relative to other tasks.

6.3 Task Complexity

In the empirical study (Chapter 5) task complexity was accounted for as an experimental factor. There was two levels of complexity (low and high). This was simply distinguished by a search task versus a composition task. However, complexity could have been broken down even further for the high complexity/ web task composition. In terms of web tasking, it appeared that the more conditions and actions there were the more 'complex' the composition task became. The only two interfaces these could be investigated on were Scribble and WebTasker.

In Scribble tasks H1 to H3 had two conditions and one action. H4 had two sets of one condition and one action. There was a learning effect observed with the H1 to H3, then H4 had a faster average task time than H1 and H2 and slower task time than H4. In WebTasker this is evidence that task

complexity and learning are likely linked. There was as similar trend as in Scribble. There was a statistically faster task time between H3 and H1 and H2 (some learning), then with the more complicated task, H4 with two sets of one condition and one action, the task time was significantly higher than H3, but still significantly lower than H1 and H2. To observe these trends refer to Figure 34 and Figure 38.

6.3.1.1 Need for more than one trigger

A recent study by Ur et al. (2015) examined the average users' interaction of trigger-action programming in the smart-home domain. It involved the participants in making up the tasks to be used in the usability study. The study found that 22% of programming behavior required more than one trigger or action. The IFTTT and Zapier platforms only allow one trigger and one action per recipe or zap. This is a case where Scribble would be superior in functionality. According to the debriefing comments in this usability study 25% of participants reported that they liked the appreciated the functionality to have more than one condition and/or action.

6.4 Correlation from Task Timing

Mean task times from each individual trial were correlated with the collected keystrokes and mouse clicks/ scrolls from each individual trial. Significant correlations ($p < 0.05$) occurred for all interfaces where $r=0.63$ for keystrokes and $r=0.62$ for mouse clicks and scrolls. It is postulated that the count of keystrokes and mouse clicks/scrolls in an HTA could yield highly predictive results in terms of task timings. Simple task input counts could be used as an early usability predictor in web tasking interfaces, potentially aiding designers with optimizing interfaces at early design stages.

HTA and counting the number of clicks and scrolls proved to be a pivotal analysis in this study. It coincided with many (not all) of the findings through the empirical study, such as:

- IFTTT's linear structure indicated that there was not much room for error (IFTTT yielded the least number of errors).
- IFTTT had the narrowest HTA structure which indicated that this type of interface promotes efficiency.
- Zapier low complexity/search task was difficult to execute.
- Zapier had too much information displayed at one time.
- Zapier would have the longest task timings.

6.5 Limitations

Several limitations of the usability study and analyses are discussed in the subsequent sections. The limitations viewed in these sections may potentially be considered as future lines or research.

6.5.1 Keystroke and Mouse Click/scrolls Count approach

Unlike the Keystroke Level Model (KLM) method that uses a number of pre-defined operators to predict expert error-free task execution times (Stanton, 2013), an elementary approach was taken in this study of counting the clicks and scrolls without association these with an execution time. One limitation of the simple keystroke and mouse click/scrolls counting approach is that it only considers physical tasks and it does not capture context or any other cognitive demands. This is also a limitation of KLM. This approach also does not account for input errors. Some systems may require more visual attention or cognitive processing in addition to manual inputs. These analytical estimates of could be further enhanced with other measures such as the number of available response options (e.g., menu items, number of buttons) as well as a correlation with a mental workload measure.

6.5.2 Prototype limitations

WebTasker was created using prototype software and was able to be interactive by mocking up linked webpages. The WebTasker prototype simulated the experience of creating a web task quite well, to the point where participants were not aware this was only a prototype. However, there were several limitations to the WebTasker prototype. It did not sufficiently handle user errors, as error messages were not simulated in the current prototype of WebTasker. If users clicked on the wrong icon, for example, they received no error message. This type of observed error was recorded by the researchers conducting the experiment. For future versions of the WebTasker prototype, error messages should be incorporated and should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

A related limitation of the prototype was that not all areas were "clickable" which caused observed frustration during the usability study. As explained in the error results section for WebTasker (Section 5.7.4.2) since WebTasker is not a fully functioning prototype, users would often repeatedly click the same icon or link and not receive any feedback (if was not the correct link to click for a specific step in the task). Each repeated click on the same wrong item was counted as an individual error.

Lastly, Axure had a time entry field that used arrows to set the time that was used as part of setting the task schedule. This caused much reported frustration with participants in addition to increasing the number of mouse click counts for the high complexity tasks. In a future version of WebTasker, the time entry should be a type-able field based on participants' feedback.

Chapter 7
Future Research

7.1 Future work on the usability study

As previously discussed, task complexity can be further defined. The variance in task complexity could be developed in more detail for web tasking interfaces noting the research done by Castaneda et al., 2013 of task complexity factors: number of web interaction, knowledge about the task, information about previous task simplifications and the number of information inputs. Furthermore, tasks could be varied or more targeted towards the participant audience. This study tried to use general apps that many students would be familiar with (social media, email, weather, etc.). A future version of this study could be specifically tailored to smart home users with tasks with IOT home items (smart thermostat, fridge, lighting, etc.), for example.

As noted in the Discussion Section 6.2, the way the task was presented to the user made a significant difference in task timing for Zapier H2. H2 had the action was presented before the condition which undoubtedly caused the high variability within the task timing as well as contributed to the greater task time. A study could be conducted specifically looking at this, controlling any variability in presenting that task to the participant.

It was difficult to compare two beta/prototype interfaces (Scribble and WebTasker) to two established programs (IFTTT and Zapier). Although, much of the statistical analysis was done within interface comparison was completed that yielded significant results, it would have been interesting to have valid and reliable data to compare across interfaces. The tasks used in each interface used in this study were similar in context and composition; however, they were not exactly the same tasks across interfaces. Due to this variance, statistically comparing task times across interfaces would not yield comprehensive results. This study could be repeated using the same tasks in each interface, if possible at a future time. If tasks are the same, then statistical tests (e.g. paired t-test comparison) would allow for further insights to the degree of improvement in WebTasker over existing interfaces.

7.2 WebTasker design

Web Tasker could incorporate a delete function (e.g. such as the garbage can on the main screen used on the Scribble interface) to support NH9: Help users recognize, diagnose, and recover from errors. With the current design of WebTasker, the user has to click into the condition or action

and then delete it from there. Another option would be to add a delete link under the edit link (as seen in Figure 27) to delete the app from the task composition screen.

The current linear control metaphor model of WebTasker is shown in Figure 57. Based on some feedback during the debriefing it was suggested that instead of one long line it should be broken up and shown in parallel as in Figure 58 (refer also to Figure 28 for a WebTasker screenshot of a complex task). This should be taken into consideration for a future revision of the WebTasker interface.

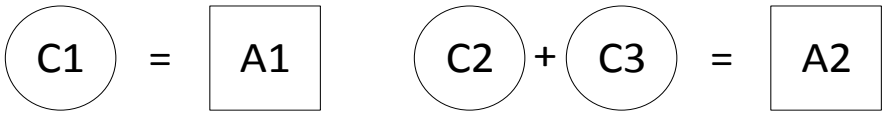


Figure 57: Current linear model of WebTasker

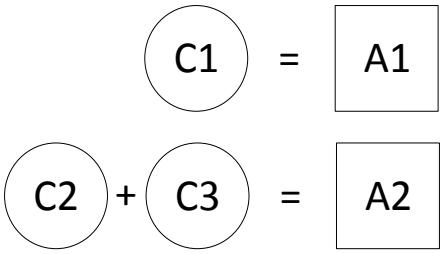


Figure 58: Potential parallel model of WebTasker

Another aspect that WebTasker could incorporate is the exploration of a test feature. This could be implemented as an optional feature for users to feel more confident in the tasks they are composing. Zapier addresses this by having a test feature to test the zaps. WebTasker could consider and optional test function for tasks in a future iteration.

7.3 Beyond this study

The subsequent sub-sections discuss a few ideas for future web tasking projects that were beyond the scope of this thesis.

7.3.1 Design of dashboard

Through the researchers' interaction with the various web tasking interfaces it was observed that none of them had an optimized interface for the task dashboard. The dashboard is where the user would control tasks (turn them off off), modify them, and view them. It is essentially the task

management hub. An interesting project to undertake would be to design the ideal web tasking dashboard. One approach that could be taken is Ecological Interface Design (EID). EID involves a systematic method of designing interfaces for complex systems. The dashboard is where task status could be displayed and ideally users' could tell with a glance the status of each task (i.e. where the car is on the journey line). Graphical metaphors, colours, screen layout etc. would have to be defined in this project. This would be an exciting project to undertake.

7.3.2 Security and trust

The issues regarding security concerns about automating tasks is related to the end user's risk tolerance. Web tasking platforms use something called ''pseudo-authentication'' to access other sites; when the user initially selects the site they want to grant access to, the automated web service contacts that website and the user can grant it limited access but not with full credentials or privileges (Hoy, 2015). Security was not an issue that was mentioned by any participants in the pilot or usability study. However, security is an important issue that should be addressed by software designers. A study from a HF perspective on trust in automation (in web tasking) could be a worthwhile venture. A test feature (like in Zapier) may address some of the trust issues associated with web tasking. To put this in context, if a user has set up a task to reward himself with the purchase of a new iPad when the condition of a 4.0 GPA is met at the end of the school year, the user would probably want to ensure he is not purchasing an iPad every term.

7.3.3 Improvements to Current Scribble Interface

Scribble is the only current web tasking program that offers the functionality of entering multiple conditions and actions. Some small interface element changes could increase its usability, as discovered through the various studies. In the short term, the current puzzle metaphor and interface could be improved in the following ways (the study source is indicated in brackets):

- Have more entry points to find published Scribbles (HTA).
- Order condition and action apps in alphabetical order when users are selecting these apps (HTA).
- Make app icons bigger and more visually appealing (pilot and usability study).

- Change puzzle pieces to be distinctly different in shape in addition to colour to help users distinguish condition piece and action piece more easily. This could decrease learning time (pilot and usability study).
- Change click interaction of adding puzzle pieces to dragging (pilot and usability study).
- Change some terminology to be clearer (e.g. “Execute Later” in schedule makes user think it will only be done once later, but it can be recurring) (pilot study).
- Change the drop down menu for time entry in the schedule to be a type-able field (usability study).
- Enable double clicking for selecting (usability study).

Chapter 8
Conclusion

User adoption of web apps has become widespread, being integrated into everyday life by the majority of computer and smart phone users. An integration across apps is being achieved through web tasking. Since web tasking is a relative new area of development, interaction analyses is key in advancing and developing this area to result in an increase in user adoption. The outcomes of this research included: collection of data on task timings, errors, SUS scores, and learning involved; a collection of various typical tasks used in web tasking including a HTA of steps and actions involved; correlation of keystroke and mouse clicks/scroll correlation to web task timings; and the 'ideal control metaphor' of a web tasking interface; WebTasker.

The usability study results showed that participants can successfully engage in creating web tasks with one condition and one action. However, participants have poorer performance and find it more difficult to create web tasks in web tasking platforms with multiple conditions and actions. More prescriptive interfaces, IFTTT and WebTasker, had better user performance than those that allowed more freedom (Zapier and Scribble), and had less frequency of human error. Participants' feedback from the debriefing interview corresponded to the user performance in the usability study. The best performance was in IFTTT and WebTasker and these were also the reported favoured interfaces by the majority. An interesting finding was revealed during the debriefing when 44% participants expressed their appreciation of the linear order and the fact that the order meant something in terms of task execution in the control metaphor used in WebTasker. This is an important finding that should be further integrated in future design revisions to WebTasker.

The usability study did find a significant difference between high (create a new web task) and low complexity tasks (search published tasks). This study has the potential to be repeated with refinements made to task selection and differences in varying number of conditions and actions (increasing complexity). In addition, interface design improvements to WebTasker could be made with a delete app function available on the task composition page (e.g. garbage can icon or link to delete), consideration of test feature, and a change in the layout of a complex task showing different sets of condition with associated actions on parallel lines instead of in series.

Other potential future web tasking projects include design of a dashboard (task management page) and an HF study on trust of task automation in the context of web tasking.

Bibliography

Annett, J., & Stanton, N. (2000). *Task analysis*. London ; New York: Taylor & Francis.

Atzori, L., Iera, A., & Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*, 54(15), 2787-2805.

Bangor, A., Kortum, P., and Miller, J. (2009). Determining what individual SUS scores mean: adding an adjective rating scale. J. *Usability Studies 4*, 3, 114-123.

Blackwell, A. (2001). See what you need: Helping end- users to build abstractions. *Journal of Visual Languages and Computing; J.Vis.Lang.Comput., 12*(5), 475-499.

Brancheau, J. C. (1993). Management of end- user computing: status and directions. *ACM Computing Surveys, 25*(4), 437-482.

Brooke, J. (1996). SUS: A "quick and dirty" usability scale. In P. W. Jordan, B. Thomas, B. A. Weerdmeester, & I. L. McClelland (Eds.), *Usability evaluation in industry* (pp. 189–194). London: Taylor & Francis.

Burnett, M., Myers, B., Rosson, M., and Wiedenbeck. S. (2006). The next step: from end-user programming to end-user software engineering. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems* (CHI EA '06). ACM, New York, NY, USA, 1699-1702.

Castañeda, L., Muller, H. A., & Villegas, N. M. (2013). Towards personalized web-tasking: Task simplification challenges. *Proceedings - 2013 IEEE 9th World Congress on Services, SERVICES 2013,* 147-153.

Castañeda, L., Villegas, N. M., & Müller, H. A. (2014). Self-adaptive applications: On the development of personalized web-tasking systems. *9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS 2014 - Proceedings, 49-54.

de Souza, C., Barbosa, S., & Da Silva, S. (2001). Semiotic engineering principles for evaluating end- user programming environments. *Interacting with Computers;Interact.Comput1 3*(4), 467-495.

Costabile, M.F., De Ruyter, B., Mehandjiev, N., and Mussio. P. (2010). End-user development of software services and applications. In *Proceedings of the International Conference on Advanced Visual Interfaces* (AVI '10), Giuseppe Santucci (Ed.). ACM, New York, NY, USA, 403-407.

Hartmann,B., Doorley, S., and Klemmer, S. R. (2003). Hacking, Mashing, Gluing: Understanding Opportunistic Design. *IEEE Pervasive Computing* 7, 3, 46-54.

Hoy, M. B. (2015). If this then that: An introduction to automated task services. *Medical Reference Services Quarterly*, 34(1), 98-103.

IDC (2014), *Telus/IDC Internet of Things Study 2014*. Retrieved from http://resources-business.telus.com/cms/files/files/000/000/698/original/InfoDoc_Telus.pdf

Kirwan, B., & Ainsworth, L. (1992). *A Guide to task analysis*. London ; Washington, DC: Taylor & Francis.

Kromrey, J.D., & La Rocca, M.A. (1995). Power and Type I error rates of new pairwise multiple comparison procedures under heterogeneous variances. *Journal of Experimental Education,* 63, 343-362.

Lewis J.R., & Sauro, J. (2009). The Factor Structure of the System Usability Scale. *In Proceedings of the 1st International Conference on Human Centered Design: Held as Part of HCI International 2009* (HCD 09), Masaaki Kurosu (Ed.). Springer-Verlag, Berlin, Heidelberg, 94-103.

Mattila, K.V.V & Wäljas, M. (2011). Towards user-centered mashups: exploring user needs for composite web services. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems* (CHI EA '11). ACM, New York, NY, USA, 1327-1332.

Nielsen, J., and Mack, R. L. (Eds.) (1994). Usability Inspection Methods, John Wiley & Sons, New York.

Ng, J.W. (2015) Task as a Service Extending the Cloud From an App Development Platform into a Tasking Platform, *Proceedings 2015 IEEE 11th World Congress on Services*.

Ng, J.W. (2014) Web tasking: a position paper. *In Proceedings of 24th Annual International Conference on Computer Science and Software Engineering (CASCON '14)*. IBM Corp., Riverton, NJ, USA, 306-313.

Ng, J. W., & Lau, D. H. (2013). Social ontology and semantic actions: Enabling social networking services for distributed web tasking. Proceedings - *2013 IEEE 9th World Congress on Services*, SERVICES 2013, 131-135.

Norman, D. A. (1988). *The psychology of everyday things*. New York: New York : Basic Books.

Prabhakararao S., Cook C., Ruthruff, J, Creswick E., Main M., Durham M., and Burnett. M. (2013). Strategies and Behaviors of End-User Programmers with Interactive Fault Localization. Proceedings - *IEEE Symposium on Human-Centric Computing Languages and* Environments, 5-22.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, Millner, A., Rosenbaum, E., Silver, J., Silverman, B., and Kafai, Y.(2009). Scratch: programming for all. *Commun. ACM 52*, 11 (November 2009), 60-67.

Robinson, L. (2009). A summary of diffusion of innovations. Retrieved 15 April 2015 from http://www.enablingchange.com.au/Summary_Diffusion_Theory.pdf

Shepherd, A. (2001). Hierarchical task analysis. London; New York: Taylor & Francis.

Statistics Canada. (2012). Canadian Internet Use Survey, 2012. Retrieved April 27, 2016 from Statistics Canada: http://www.statcan.gc.ca/daily-quotidien/131126/dq131126d-eng.pdf.

Stanton, N. (2013). *Human factors methods: a practical guide for engineering and design (*Second edition. ed.). Farnham, Surrey; Burlington, VT: Ashgate Publishing Limited.

Stanton, N. (1996). Human factors in nuclear safety. London; Bristol, PA: Taylor & Francis.

Stolee, K.T., Elbaum, S., Rothermel, G. (2009). Revealing the copy and paste habits of end users. Proceedings 2009 *IEEE Symposium on Visual Languages and Human-Centric Computing,* pp.59-66.

Trochim, W. M. K. (2005). *Research methods: the concise knowledge base.* Cincinnati, OH: Atomic Dog Publishing.

Ur, B., McManus, E., Ho, M. P. Y., & Littman, M. L. (2014). Practical trigger-action programming in the smart home. *Conference on Human Factors in Computing Systems - Proceedings,* 803-812.

Wilson, J., & Corlett, E. (1995). *Evaluation of human work: A practical ergonomics methodology* (2nd ed.). Bristol, Pa.: Taylor & Francis.

Appendix A

Usability Study Material

A.1 List of tasks used in usability study

IFTTT Create Tasks

[User will begin on https://ifttt.com/myrecipes/personal/new page]

Task 1

If daily step goal is achieved in Fitbit, then send a new email from Gmail.

- Enter condition: select Fitbit app, if a daily step goal achieved in Fitbit
- Enter action: select Gmail then send a new email, enter in To address "ekittel@uwaterloo.ca"
- Enter Recipe Title: "test1" and Create Recipe <end task>

Task 2

If iPad price changes at BestBuy then post a tweet.

- Enter condition: select BestBuy app, if product prices changes, use SKU 3315023
- Enter action: select twitter then post a tweet
- Enter Recipe Title "test2" and Create Recipe <end task>

Task 3

If Facebook new status message is posted by you then create a text post in Tumblr

- Enter condition: select Facebook app, if a new status message by you
- Enter action: select Tumblr app, then create a text post
- Enter Recipe Title: "test3" and Create Recipe <end task>

Task 4

If the IBM stock prices rises above $160, then send an email from Gmail

- Enter condition: select Stocks app, if price rises above, user ticker symbol IBM, enter price $160
- Enter action: select Gmail, then send an email, enter in To address "ekittel@uwaterloo.ca"
- Enter Recipe Title: "test4" and Create Recipe <end task>

IFTTT Search Tasks

[User will begin on https://ifttt.com/recipes page]

1. Find: Tweet when you achieve your daily step goal in Fitbit

2. Find: If it's going to rain tomorrow, send me an email from Gmail. Enter email address "ekittel@uwaterloo.ca"

3. Find: Share new links you post on Facebook to Twitter.

4. Find: If google stock price drops, send an email reminder to purchase more shares. Enter To address "ekittel@uwaterloo.ca"

Zapier Create Tasks

[User will begin on https://zapier.com/app/editor-original/5988019 page]

Task 1

When you post a new tweet post it to your Facebook timeline

This is what your task must contain. You do NOT necessarily have to follow this order:

- Enter condition/trigger: select twitter app, select My Tweet- Triggers from you tweet something new
- Enter action: select Facebook app, select Post to Timeline- Create a new post on your timeline.
- Select UWHFstudent@gmail.com Twitter and Facebook account
- Type in message field "This is test1"
- Test twitter trigger
- Name Zap "test1"
- Turn Zap on <end task>

Task 2

Send an email with Gmail for new tweets from @UWHFstudentgirl

This is what your task must contain. You do NOT necessarily have to follow this order:

- Enter action: select Gmail app, select Send Email
- Enter condition/trigger: select twitter app, select User Tweet
- Select UWHFstudent@gmail.com twitter account and UWHFstudent@gmail.com account
- Enter username "UWHFstudentgirl" in 'Only trigger a "User Tweet" from Twitter when…' step

- Enter ekittel@gmail.com in the To field.
- Enter “test 2” in the Subject field.
- Enter “This is a test.” in the Body field.
- Test twitter trigger
- Name zap “test 2”
- Turn Zap on <end task>

Task 3

When you star an email in Gmail post it to your Facebook timeline.

This is what your task must contain. You do NOT necessarily have to follow this order:

- Enter condition: select Gmail app, select New Starred Email
- Enter action: select Facebook app, select Post to Timeline
- Select UWHFstudent@gmail.com Gmail and Facebook account
- Type “test 3” in Message field
- Test Gmail trigger
- Name zap “test 3”
- Turn Zap on <end task>

Task 4

If it is going to rain today, send me an email from Gmail

This is what your task must contain. You do NOT necessarily have to follow this order:

- Enter action: select Gmail app, select Send Email
- Enter condition: select Weather by Zapier app, select Will it Rain Today?
- Select UWHFstudent@gmail.com account
- Enter 43.4 in Latitude field
- Enter -80.5 in Longitude field
- Enter ekittel@uwaterloo.ca in To field
- Enter “test 4” in the subject field
- Enter “This is a test.” in the Body field.
- Test weather trigger
- Name zap “test 4”
- Turn Zap on <end task>

Zapier Search Tasks

[User will begin on https://zapier.com/app/apps-explore page]

1. Email for a User's Twitter Tweets, Get an email via Gmail for new tweets from a specific user.

2. Trigger Weekly Email Reminders, sends an email via Gmail on a weekly basis to remind me to do stuff.

3. Send an Email via Gmail at the same time every day.

4. Post My Tweets to Facebook Page

Scribble Create Tasks

[User will begin on

http://taskasaservice.canlab.ibm.com:10080/ScribbleProject/apps/services/www/ScribbleApp/desktopbrowser/default/index.html page]

Task 1

If fit bit step goal is met AND UW GPA is met, then buy me an iPad from Best Buy; run this task every day.

This is what your task must contain. You do NOT necessarily have to follow this order:

- Enter condition:
  - select fit bit app (called "(Demo) User activity"),
  - select "${my user id}" in Information needed for this condition field,
  - select "Get Activity" in What is involved in this condition field,
  - select "User step account", select "matches", and enter "10000" in Specify conditions field
- Enter condition:
  - Select UW app (look for UW logo and it is called "(Demo) Secured University")
  - Select "Read" in What is involved in this condition field
  - Select "GPA of student" , select "greater than", and enter 85 in the Specify conditions field

- Enter action:
  - Select Best Buy app (look for Best Buy logo, called, "(Demo) Product")
  - Select "Order" in What do you want to do with it field
  - Enter "64GB iPad" in Product Name field
  - Enter "Silver" in colour field
  - Enter "1" in quantity field

- Enter action
    - Select notify app (called “Notify Someone”)
    - Select “Me” in Who field
    - Enter “Task 1” in Subject field
    - Enter “This is Task 1” in Message field

- Set schedule
    - Select “Execute Later” in Schedule a time option, and choose today’s date and 1:00 PM
    - Select “Repeat”
    - Select “Daily” and select 1:00 PM

- Name Scribble “Task 1” in Scribble name field
    - Select Save <end task>
    -

Task 2

If IBM stock is >$175 AND exchange rate is met, then notify me; run this task bi-weekly.

This is what your task must contain. You do NOT necessarily have to follow this order:

- Enter action
    - Select notify app (called “Notify Someone”)
    - Select “Me” in Who field
    - Enter “Task 2” in Subject field
    - Enter “This is Task 2” in Message field

- Enter condition:
    - select Stock app (called “Stock Quote from WebServiceX”)
    - select “Get Quote” in What is involved in this condition field
    - Enter “IBM” in the Stock Symbol field
    - Select “Current Price” and “greater than” and enter “175” in Specify conditions field
    - Specify condition frequency check
        - Enter 1 day, 2:00 and 1 MS
- Enter condition:
    - select Currency Exchange app (called “Currency Exchange from WebServiceX”)
    - select “Calculate Rate” in What is involved in this condition field
    - Enter “USD” in To (Currency Symbol) field
    - Enter “CAD” in From (Currency Symbol) field

    - Select “Currency Exchange Rate” and “matches” and enter “1.32” in Specify conditions field
    - Click Specify condition frequency check
        - Enter 1 day, 2:00 and 1 MS

- Set schedule
    - Select “Execute Later” in Schedule a time option, and choose today’s date and 7:15 AM
    - Select “Repeat”
    - Select “Bi-Weekly” and “Monday” and select 7:15 AM

- Name Scribble “Task 2” in Scribble name field
    - Select Save <end task>

Task 3

If my bank account balance is less than $1000 then notify me; run this task every Friday at 10:00 a.m.

This is what your task must contain. You do NOT necessarily have to follow this order:

- Set schedule
    - Select “Execute Later” in Schedule a time option, and choose today’s date
    - Select “Weekly” and “Friday” and select 10:00 AM

- Enter action
    - Select notify app (called “Notify Someone”)
    - Select “Me” in Who field
    - Enter “Task 3” in Subject field
    - Enter “This is Task 3” in Message field

- Enter condition:
    - select CIBC app (called “(Demo) Secured Bank”)
    - select “Read Balance” in What is involved in this condition field
    - Enter “1234567” in the Account ID field
    - Select “Bank account balance” and “less than” and enter “1000” in Specify conditions fields

- Name Scribble “Task 3” in Scribble name field
    - Select Save <end task>

Task 4

If my bank account balance is more than $25, then buy me a movie ticket me AND if I meet my daily step count goal then notify me; run this task on demand.

This is what your task must contain. You do NOT necessarily have to follow this order:

- Enter condition:
  - select RBC app (called "(Demo) Account balance")
  - select "Read Balance" in What is involved in this condition field
  - Enter "1234567" in the Account ID field
  - Select "Bank account balance" and "greater than" and enter "25" in Specify conditions fields
- Enter action
  - Select Cineplex app (called "(Demo) Movie Ticket"
  - Select "Buy Cineplex Ticket"
  - Enter "2" in the Number of Tickets field

- Enter action
  - Select notify app (called "Notify Someone")
  - Select "Me" in Who field
  - Enter "Task 4" in Subject field
  - Enter "This is Task 4" in Message field
- Enter condition:
  - select fit bit app (called "(Demo) User activity"),
  - select "${my user id}" in Information needed for this condition field,
  - select "Get Activity" in What is involved in this condition field,
  - select "User step account", select "matches", and enter "10000" in Specify conditions field

- Set schedule
  - Select "Execute on Demand"

- Name Scribble "Task 4" in Scribble name field
  - Select Save <end task>

Scribble Search Tasks

[User will begin on Community of Scribbles page]

1. Find movie listings

2. Find Take the next bus to the airport if the weather is clear

3. Find Buy Twitter stock check

4. Find Book a vacation.

WebTasker Create Tasks

[User will begin on http://8cdoj0.axshare.com/#p=home at Create New Task page]

WebTasker Task 1

If fit bit weight goal is met AND UW GPA is met, then buy me an iwatch from Best Buy; run this task every day.

- Enter condition:
  - select fit bit app
  - select "Read Weight" in the Information needed for this condition field
  - select "User Weight" in Specify conditions field, and "matches", and enter "150" in Specify conditions field

- Enter condition:
  - Select UW app (look for UW logo and it is called "GPA")
  - Select "GPA of student" , select "greater than", and enter 85 in the Specify conditions field

- Enter action:
  - Select Best Buy app
  - Select "Order" in What do you want to do with it field
  - Enter "64GB iwatch" in Product Name field
  - Enter "Silver" in colour field
  - Enter "1" in quantity field
- Set schedule
  - Choose today's date and 5:00 PM
  - Select "Repeat"
  - Select "Daily" and select 5:00 PM

- Name Task "Task 1" in Task name field
  - Select Save & Submit <end task>

WebTasker Task 2

If IBM stock is >$175 AND exchange rate is met, then notify me; run this task bi-weekly.

- Enter condition:
  - select Stock app
  - select “Get Quote” in What is involved in this condition field
  - Enter “IBM” in the Stock Symbol field
  - Select “Current Price” and “greater than” and enter “175” in Specify conditions field
  - Specify condition frequency check
    - Enter 1 day, 4 hours, and 0 minutes
- Enter condition:
  - select Currency Exchange app (called “Currency Exchange from WebServiceX”)
  - select “Calculate Rate” in What is involved in this condition field
  - Enter “USD” in To (Currency Symbol) field
  - Enter “CAD” in From (Currency Symbol) field
  - Select “Currency Exchange Rate” and “matches” and enter “1.32” in Specify conditions field
  - Click Specify condition frequency check
    - Enter 1 day, 4 hours, and 0 minutes
- Enter action
  - Select Notification app
  - Select “Me” in Who field
  - Enter “Task 2” in Subject field

- Set schedule
  - Choose tomorrow’s date and 7:15 AM
  - Select “Repeat”
  - Select “Bi-Weekly” and select 7:15 AM

- Name Task “Task 2” in Task name field
  - Select Save & Submit <end task>

WebTasker Task 3

If my bank account balance is less than $1000 then notify me; run this task every Friday at 10:00 a.m.

- Enter condition:
    - select RBC app
    - select “Chequing Account”, “less than” and enter “1000” in Specify Conditions field
    - select 2 days, 3 hours, 0 minutes in Frequency of condition check field.
- Enter action
    - Select Notification app
    - Select “Me” in Who field
    - Enter “Task 3” in Subject field
- Set schedule
    - Select “Daily” and select 10:00 AM.

- Name Scribble “Task 3” in Scribble name field
    - Select Save & Submit <end task>

WebTasker Task 4

If my bank account balance is more than $25, then buy me a movie ticket me AND if I meet my daily step count goal then notify my mom; run this task on demand.

- Enter condition:
    - select RBC app
    - select “Chequing Account”, “less than” and enter “25” in Specify Conditions field
    - select 2 days, 3 hours, 0 minutes in Frequency of condition check field.

- Enter action
    - Select Cineplex app
    - Select “Buy Cineplex Ticket”

- Enter NEW set of condition and action
    - Enter condition: select fit bit app
    - select “User step account”, and “is greater than”, and enter “5000” in Specify condition fields

    - Enter action:
    - Select Notification app
    - Select “Mom” in Who field
    - Enter “Task 4” in Subject field

- Set schedule

    - Leave blank to run now.

- Name Task “Task 4” in Task name field
    - Select Save & Submit <end task>

WebTasker Search Tasks

[User will begin on http://56rz68.axshare.com/#p=results_1 page]

1. Find Tweet when you achieve your daily step goal in Fit Bit
    - Search “fit bit”

2. Find Get an email if there is going to be rain in your area tomorrow
    - Search “rain”

3. Find Share links you post on Facebook to Twitter
    - Search “links”

4. Find Notify me if Google Stock price changes
    - Search “Google Stock”

UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Systems
Design Engineering

A.2 Participant information letter, consent form, and briefing script

Information Letter and Consent of Participant

You are invited to participate in a Web Tasking Interface Study examining usability issues with web tasking interfaces. User adoption of web applications (apps) has become widespread, being integrated into everyday life by the majority of computer and smart phone users. Users are finding multiple ways to utilize web apps outside of their typical self-contained purposes, resulting in an increasing need to connect apps together.

An integration across web apps can be achieved with a web tasking platform. This is where web tasks can be created by the end user by connecting several components of different web apps. Web tasking is a new area in Human Computer Interaction (HCI) research and this proposed study aims to gather data on existing web tasking platforms; including a prototype designed by the researcher, to further develop web tasking interfaces to ultimately lead to an increase in user adoption. In this study are interested in gathering data on task timings, errors, and learning involved with different web tasking platforms.

What You Will Be Asked to Do

After your consent, you will be asked to complete short demographic questionnaire. You will then be provided with a list of tasks to complete in different web tasking interfaces. The tasks entail entering conditions and actions on the web tasking interface; for example, if it is going to rain tomorrow send me an email.

At the end of each interface, you will be asked to fill out an 11-question System Usability Scale (SUS) questionnaire.

UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Systems
Design Engineering

Participation and Remuneration

Participation in this study is voluntary, and you will take approximately two hours of your time. You may decline to answer any questions presented by the experimenter. Further, you may decide to withdraw from this study at any time by advising the researcher, and may do so without any penalty or loss. You will be paid $20 per hour for your participation in this study even if you decide to withdraw your consent at any time. The amount received is taxable. It is your responsibility to report this amount for income tax purposes.

Contact Information

Catherine Burns
Phone: (519) 888-4567 ext. 33903
Email: catherine.burns@uwaterloo.ca

Elizabeth Kittel
Email: ekittel@uwaterloo.ca

UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Systems
Design Engineering

Consent

By signing this consent form, you are not waiving your legal rights or releasing the investigator(s) or involved institution(s) from their legal and professional responsibilities.

You agree to no further disclosure of the user interfaces reviewed in this study, since some of the interfaces not commercially available or have not been released to market.

I have read the information presented in the information letter about a study being conducted by Elizabeth Kittel under the supervision of Dr. Catherine Burns of the Department of Systems Design Engineering at the University of Waterloo. I have had the opportunity to ask any questions related to this study, to receive satisfactory answers to my questions, and any additional details I wanted. I am aware that I may withdraw from the study without penalty at any time by advising the researchers of this decision.

This project has been reviewed by, and received ethics clearance through a University of Waterloo Research Ethics Committee. I was informed that if I have any comments or concerns resulting from my participation in this study, I may contact the Director, Office of Research Ethics at 519-888-4567 ext. 36005.

With full knowledge of all foregoing, I agree, of my own free will, to participate in this study.

Name of Participant

_____

Signature of Participant

_____

Witness Name

_____

Witness Signature

_____

Date

_____

Briefing Script

Hello. Thank you for participating in this test. Please fill out this participant questionnaire with me.

This is a web tasking interface study. I will ask you to create several tasks in four different interfaces. Web tasking is the integration of apps to achieve one goal/task. In this day and age, app users are finding more than one purpose for apps and it appears that single apps are no longer meeting their needs. An integration across apps is what would help them in their tasks across their web apps. This is the reason web tasking platforms were created. They are still relatively new and require further development. Your participation in this study will contribute to that.

For each of the four interfaces, there will be a condition and an action app you will enter to complete the task. For example, if the forecast calls for rain tomorrow send me an email today. You will need to select the weather app as the condition and the email app as the action and enter some information in the appropriate fields (for example the date and condition of rain for the weather). Each interface is different in terms of how they look and their functionality.

You will be creating these tasks from scratch or you will be searching for them in the already published tasks (that were created by other users), [tell them their order here, i.e. create or search tasks first]. You will be instructed as to which tasks you must create yourself and the ones you will search for.

The way you interact with the web tasking interface will be observed and recorded by screen capture, keystrokes, and timing. You will have a time limit of about 5 minutes per task for creating tasks and a 2 minute time limit for your search tasks. I will let you know when your time is up. The paper in front of you is your guide. Feel free to use it at any time.

Do you have any questions?

A.3 Demographics Questionnaire

Web Tasking Study Demographics Questionnaire (to be filled out by experimenter)

Participant Code: _____ Age: _____

Gender: MALE FEMALE

Do you have any computer programming experience? YES NO

If YES, what computer languages do you have experience and level do you have?

_____	BASIC	INTERMEDIATE	ADVANCED
_____	BASIC	INTERMEDIATE	ADVANCED
_____	BASIC	INTERMEDIATE	ADVANCED
_____	BASIC	INTERMEDIATE	ADVANCED
_____	BASIC	INTERMEDIATE	ADVANCED

Familiarity with web tasking interfaces:

Please indicate your familiarity with the following interfaces (circle all that apply):

IFTTT NONE KNOW OF IT TRIED IT USE IT FREQUENTLY

Zapier NONE KNOW OF IT TRIED IT USE IT FREQUENTLY

Have you ever used any other web tasking type program before? YES NO

If YES, please indicate which program and the extent of use:

_____ KNOW OF IT TRIED IT USE IT FREQUENTLY

A.4 SUS Questionnaire

The following statements were rated using a 5-point scale:

1. I think that I would like to use this system frequently

2. I found the system unnecessarily complex

3. I thought the system was easy to use

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

5. I found the various functions in this system were well integrated

6. I thought there was too much inconsistency in this system

7. I would imagine that most people would learn to use this system very quickly

8. I found the system very cumbersome to use

9. I felt very confident using the system

10. I needed to learn a lot of things before I could get going with this system

A.5 Debriefing Questionnaire

De-briefing Questions (to be asked by experimenter)

1. Did you have a preference to enter actions or conditions first? If yes, why?

2. What did like about the functionality and features of each web tasking platform? Is there anything else you would like to see in terms of functionality?

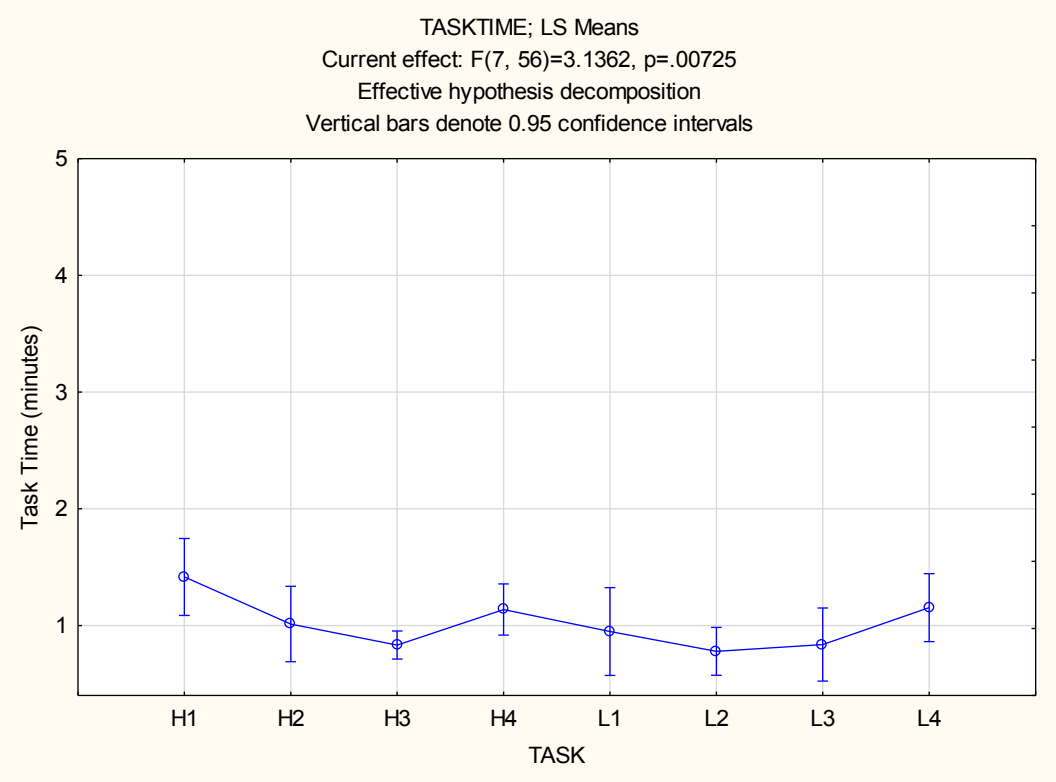
3. How could we improve setting the frequency of condition check and setting the task schedule?

4. Which interface did you like the most and why?

Appendix B

Statistical Analysis

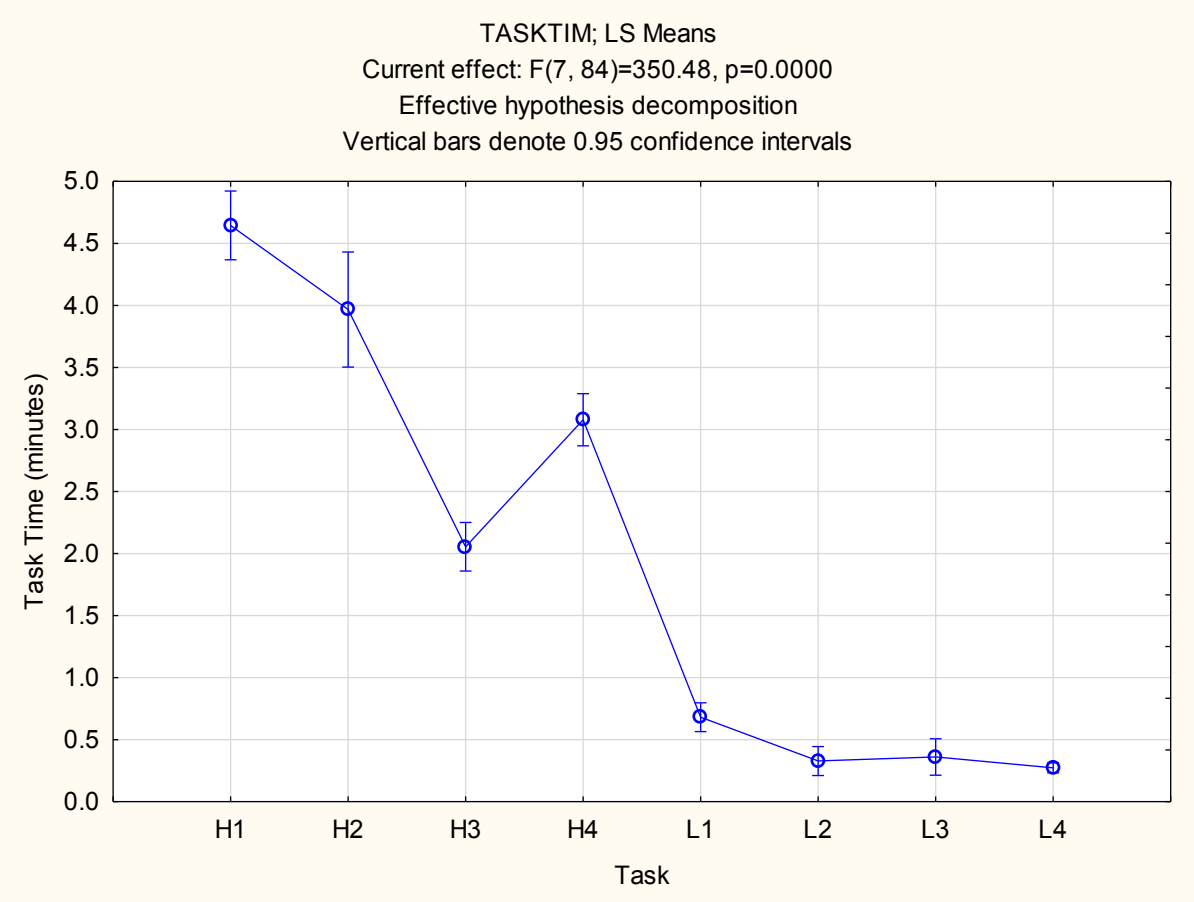
B.1 IFTTT Results



Effect	Repeated Measures Analysis of Variance (IFTTT) Sigma-restricted parameterization Effective hypothesis decomposition SS	Degr. of Freedom	MS	F	p
Intercept	132.6736	1	132.6736	235.8126	0.000000
Gender	0.3779	1	0.3779	0.6717	0.428440
Computer Prog Exp	0.9774	1	0.9774	1.7372	0.212099
Gender*Computer Prog Exp	1.3922	1	1.3922	2.4745	0.141686
Error	6.7515	12	0.5626		
TASKTIME	4.8252	7	0.6893	3.1420	0.005357
TASKTIME*Gender	1.4685	7	0.2098	0.9562	0.468485
TASKTIME*Computer Prog Exp	0.6781	7	0.0969	0.4415	0.873241
TASKTIME*Gender*Computer Prog E	0.8586	7	0.1227	0.5591	0.786954
Error	18.4288	84	0.2194		

Cell No.	Tukey HSD test; variable DV_1 (IFTTT) Approximate Probabilities for Post Hoc Tests Error: Within MS = .21939, df = 84.000 TASK TIME	{1} 1.4604	{2} 1.0823	{3} .85208	{4} 1.2125	{5} .99062	{6} .79583	{7} .91042	{8} 1.1365
1	H1		0.315097	0.009630	0.806940	0.099792	0.003205	0.027790	0.517192
2	H2	0.315097		0.859301	0.993467	0.999331	0.668057	0.967193	0.999981
3	H3	0.009630	0.859301		0.376525	0.990482	0.999975	0.999968	0.676193
4	H4	0.806940	0.993467	0.376525		0.880866	0.203259	0.605769	0.999809
5	L1	0.099792	0.999331	0.990482	0.880866		0.936526	0.999727	0.987101
6	L2	0.003205	0.668057	0.999975	0.203259	0.936526		0.997052	0.451052
7	L3	0.027790	0.967193	0.999968	0.605769	0.999727	0.997052		0.870339
8	L4	0.517192	0.999981	0.676193	0.999809	0.987101	0.451052	0.870339	

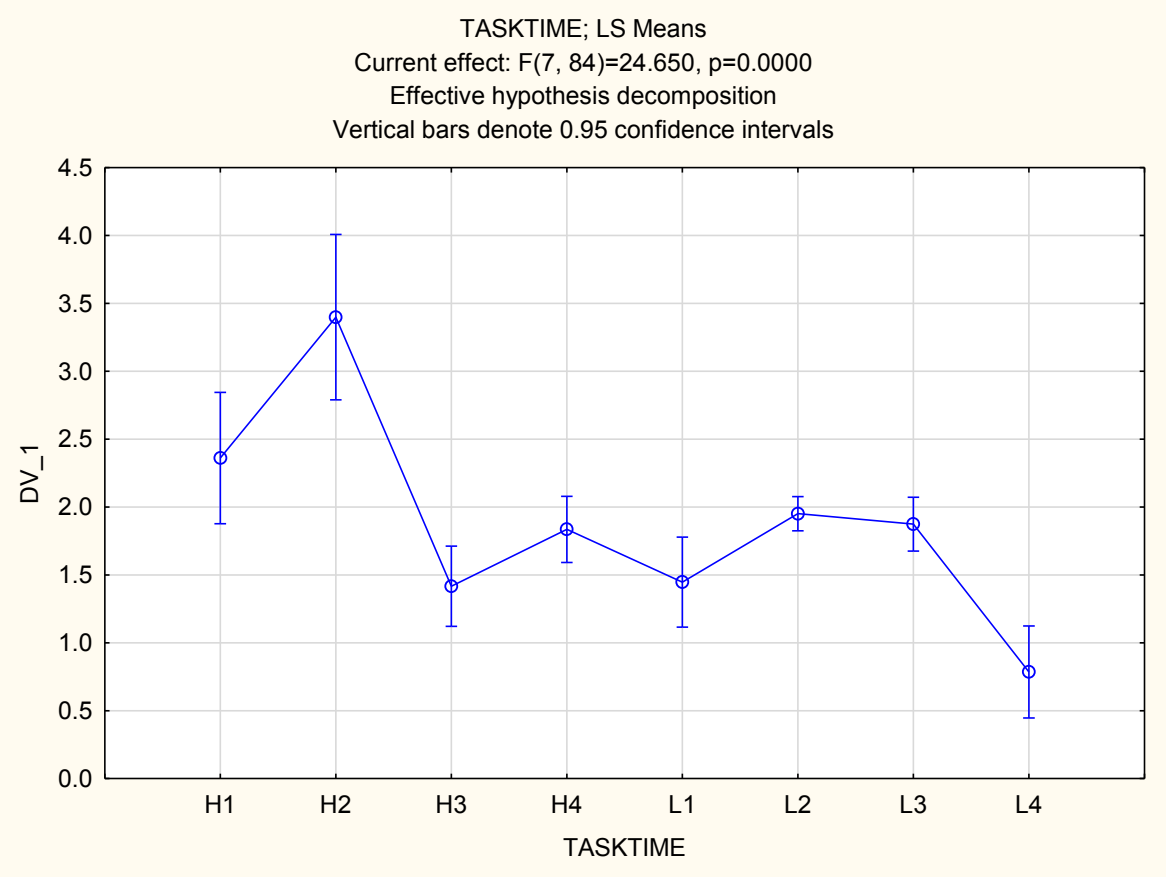
B.2 Scribble Results



Effect	Repeated Measures Analysis of Variance (Scribble in Scribble data) Sigma-restricted parameterization Effective hypothesis decomposition SS	Degr. of Freedom	MS	F	p
Intercept	458.2952	1	458.2952	1130.459	0.000000
Computer Prog Exp	0.0052	1	0.0052	0.013	0.911866
Gender	0.2049	1	0.2049	0.505	0.490745
Computer Prog Exp*Gender	0.8755	1	0.8755	2.160	0.167406
Error	4.8649	12	0.4054		
TASKTIM	343.6684	7	49.0955	350.475	0.000000
TASKTIM*Computer Prog Exp	0.5287	7	0.0755	0.539	0.802491
TASKTIM*Gender	1.4458	7	0.2065	1.474	0.187511
TASKTIM*Computer Prog Exp*Gender	1.6489	7	0.2356	1.682	0.124575
Error	11.7669	84	0.1401		

Cell No.	Tukey HSD test; variable DV_1 (Scribble in Scribble data) Approximate Probabilities for Post Hoc Tests Error: Within MS = .14008, df = 84.000 TASK TIM	{1} 4.6656	{2} 3.9708	{3} 2.0646	{4} 3.1010	{5} .68229	{6} .33333	{7} .33854	{8} .27500
1	H1		0.000143	0.000119	0.000119	0.000119	0.000119	0.000119	0.000119
2	H2	0.000143		0.000119	0.000119	0.000119	0.000119	0.000119	0.000119
3	H3	0.000119	0.000119		0.000119	0.000119	0.000119	0.000119	0.000119
4	H4	0.000119	0.000119	0.000119		0.000119	0.000119	0.000119	0.000119
5	L1	0.000119	0.000119	0.000119	0.000119		0.157573	0.171489	0.054324
6	L2	0.000119	0.000119	0.000119	0.000119	0.157573		1.000000	0.999854
7	L3	0.000119	0.000119	0.000119	0.000119	0.171489	1.000000		0.999742
8	L4	0.000119	0.000119	0.000119	0.000119	0.054324	0.999854	0.999742	

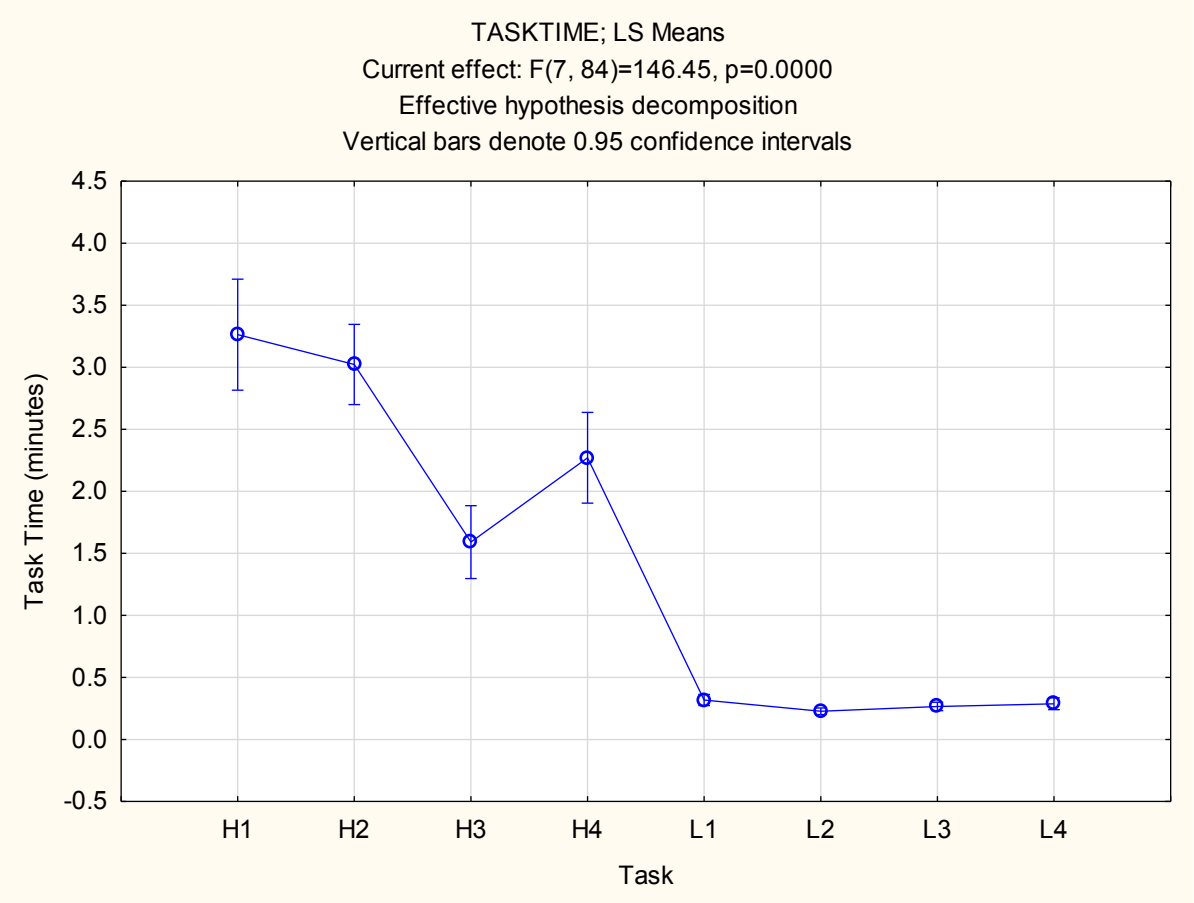
B.3 Zapier Results



Effect	Repeated Measures Analysis of Variance (Zapier dat Sigma-restricted parameterization Effective hypothesis decomposition SS	Degr. of Freedom	MS	F	p
Intercept	454.1972	1	454.1972	577.6865	0.000000
Computer Prog Exp	0.0894	1	0.0894	0.1137	0.741747
Task Complexity	0.0172	1	0.0172	0.0219	0.884907
Computer Prog Exp*Task Complexity	0.0071	1	0.0071	0.0090	0.926117
Error	9.4348	12	0.7862		
TASKTIME	66.3422	7	9.4775	24.6498	0.000000
TASKTIME*Computer Prog Exp	2.6309	7	0.3758	0.9775	0.453092
TASKTIME*Task Complexity	1.2520	7	0.1789	0.4652	0.857071
TASKTIME*Computer Prog Exp*Task Comple›	0.8795	7	0.1256	0.3268	0.939804
Error	32.2966	84	0.3845		

Cell No.	Tukey HSD test; variable DV_1 (Zapier data1) Approximate Probabilities for Post Hoc Tests Error: Within MS = .38448, df = 84.000 TASKTIME	{1} 2.3615	{2} 3.3990	{3} 1.4167	{4} 1.8354	{5} 1.4469	{6} 1.9510	{7} 1.8740	{8} .785
1	H1		0.000331	0.001201	0.255323	0.001896	0.573611	0.348616	0.000
2	H2	0.000331		0.000119	0.000119	0.000119	0.000119	0.000119	0.000
3	H3	0.001201	0.000119		0.548073	1.000000	0.237462	0.432470	0.090
4	H4	0.255323	0.000119	0.548073		0.640226	0.999521	1.000000	0.000
5	L1	0.001896	0.000119	1.000000	0.640226		0.306187	0.522616	0.063
6	L2	0.573611	0.000119	0.237462	0.999521	0.306187		0.999968	0.000
7	L3	0.348616	0.000119	0.432470	1.000000	0.522616	0.999968		0.000
8	L4	0.000119	0.000119	0.090037	0.000288	0.063656	0.000137	0.000201	

B.4 WebTasker Results



Effect	Repeated Measures Analysis of Variance (WebTasker data) Sigma-restricted parameterization Effective hypothesis decomposition SS	Degr. of Freedo	MS	F	p
Intercept	244.7752	1	#######	524.1296	0.000000
Computer Prog Exp	0.3601	1	0.3601	0.7710	0.397146
Gender	0.3718	1	0.3718	0.7960	0.389822
Computer Prog Exp*Gender	3.0914	1	3.0914	6.6196	0.024417
Error	5.6042	12	0.4670		
TASKTIME	185.3075	7	26.4725	146.4519	0.000000
TASKTIME*Computer Prog Exp	0.7637	7	0.1091	0.6036	0.751341
TASKTIME*Gender	2.0728	7	0.2961	1.6382	0.135894
TASKTIME*Computer Prog Exp*Gender	2.8381	7	0.4054	2.2430	0.038477
Error	15.1837	84	0.1808		

Cell N	Tukey HSD test; variable DV_1 (WebTasker repeated measures) Approximate Probabilities for Post Hoc Tests Error: Within MS = .18076, df = 84.000 TASKTIME	{1} 3.3094	{2} 3.0479	{3} 1.6219	{4} 2.2687	{5} .31458	{6} .22708	{7} .27083	{8} .28854
1	H1		0.661824	0.000119	0.000119	0.000119	0.000119	0.000119	0.000119
2	H2	0.661824		0.000119	0.000152	0.000119	0.000119	0.000119	0.000119
3	H3	0.000119	0.000119		0.001233	0.000119	0.000119	0.000119	0.000119
4	H4	0.000119	0.000152	0.001233		0.000119	0.000119	0.000119	0.000119
5	L1	0.000119	0.000119	0.000119	0.000119		0.999055	0.999991	1.000000
6	L2	0.000119	0.000119	0.000119	0.000119	0.999055		0.999991	0.999912
7	L3	0.000119	0.000119	0.000119	0.000119	0.999991	0.999991		1.000000
8	L4	0.000119	0.000119	0.000119	0.000119	1.000000	0.999912	1.000000	

B.5 SUS Score

Effect	Multivariate Tests of Significance (SUS questionnaire results) Sigma-restricted parameterization Effective hypothesis decomposition					
	Test	Value	F	Effect df	Error df	p
Intercept	Wilks	0.039260	599.5444	2	49	0.000000
Interface	Wilks	0.571285	5.2763	6	98	0.000095
Programming experience	Wilks	0.943535	1.4662	2	49	0.240751
Interface Order	Wilks	0.642360	1.5172	16	98	0.108850
Gender	Wilks	0.878669	3.3831	2	49	0.042046

Cell No.	Tukey HSD test; variable SUS Score (SUS questionnaire results) Approximate Probabilities for Post Hoc Tests Error: Between MS = 382.23, df = 50.000				
	Interface	{1} 88.656	{2} 63.125	{3} 85.469	{4} 53.438
1	IFTTT		0.003098	0.967181	0.000190
2	Scribble	0.003098		0.011409	0.504444
3	WebTasker	0.967181	0.011409		0.000297
4	Zapier	0.000190	0.504444	0.000297	