THE USABILITY OF A VISUAL, FLOW-BASED PROGRAMMING ENVIRONMENT FOR NON-PROGRAMMERS

by

Alia H. Shubair

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the Program of

Computer Science

Toronto, Ontario, Canada, 2014

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 authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public for the purpose of scholarly research only.

Abstract

The Usability of A Visual, Flow-Based Programming Environment for Non-Programmers

Alia H. Shubair

Master of Science, Computer Science

Ryerson University, 2014

Living with "Big Data" gives us the advantage of being able to exploit this wealth of data sources and derive useful insights to make better decisions, enhance productivity, and optimize resources. However, this advantage is limited to a small group of professionals, with the rest of the population unable to access this data. Lack of support for non-professionals creates the need for data manipulation tools to support all sectors of society without acquiring complex technical skills.

"Kit" is a visual flow-based programming environment that aims to facilitate manipulation and visualization for all citizens, particularly non-programmers, enabling them to have hands on data in an easy manner.

This study evaluates Kit's usability by having non-programmers involved in various evaluation activities to assess their ability to solve data-related problems using a prototype of the environment. The results provided useful insights to improve the design of data manipulation tools aiming to support non-programmers.

Acknowledgements

I would like to express my deepest gratitude to my supervisor Dr. Dave Mason, who helped through my ups and downs in this research. He has been always by my side guiding, advising, and supporting the completion of this thesis.

I would like to thank my committee chair and our graduate program co-ordinator Dr. Alex Ferworn, who has been always a good friend and a great support for all the graduate students. His positive attitude and care have been an extra motivation for us to give back our best accomplishments.

In addition, I would like to express my appreciation for my committee members Dr. Eric Harley and Dr. Deborah Fels for their guidance and patience through my thesis defence.

I would like to express my appreciation for Prof. Sophie Quigley, who allowed me to be a part of her HCI course experiments. She has been a great help whenever guidance was needed.

I would like to thank Ryerson University and the computer Science Department specifically for giving me this opportunity to pursue my research journey. Furthermore, I would like to express my appreciation to those individuals within the department who will never be forgotten.

A sincere gratitude goes to my best friend Dr. Rasha Aljamal, who has given me the strength and the encouragement through the hard days.

A special and deep gratitude goes to my extraordinary parents Hasan and Eman, who taught me to cherish knowledge and seek for it wherever it may be found.

Last but not least, thank you from the bottom of my heart goes to my husband Khalil who has been always supportive, cooperative, and patient. I would not have done it without his love.

To

To my lovely little angels Abraham & Nada..

I love you.

Table of Contents

List of Figures

Chapter 1

Introduction

> "The reason big data is impacting every one of us is the data oozing out of everything ... It is like electricity flowing throughout an organization – everyone can tap into it on command to answer the individual questions their jobs demand."
>
> Pat Hanrahan – Stanford University

My thesis is that: *Kit can provide an easy to learn and use environment that facilitates data manipulation for non-programmers.*

Living in an information revolution age, gives us the advantage of having everything in a digitized form. Having such wealth of data sources allows us to exploit it into some useful insights. This capability should not be restricted to a technical subset of the population. Having the ability to manipulate data surrounding us should be democratized to include each individual.

My team and I are developing a new interactive, visual flow-based environment that facilitates analyzing data effectively and easily. My role in this project was to run studies with a prototype of the environment, Kit, to evaluate and enhance its usability for this specific audience of non-programmers. My research consisted of four different phases with the engagement of 21 non-programmers who were categorized into three groups. These groups were: the University Planning Office group, the Journalism Grad-

uate Students group, and the General Population group. The phases were designed as following:

The first phase was **"The Introduction."** Through this phase, participants were introduced to the environment by illustrating some previously implemented trivial examples. The environment uses the flow-based paradigm that consists of collections of black boxes that execute a function. These boxes are connected with pipes that allow data flow from one block to another. This phase included assessing non-programmer ability to understand the illustrated data-flow paradigm used in Kit, assessing their ability to replicate the previously illustrated examples, and evaluating their initial impressions of the environment.

The second phase was **"The Design."** This phase was held to collect some design requirements by giving participants some data-related problem scenarios using the technique of *Personas and Scenarios-of-Use* in a focus group session for each participating group. Participants were given four problem scenarios, and they were asked to suggest some models to process these scenarios using the paradigm of blocks and pipes. The main objectives of this phase were to assess participants' ability to suggest some solution models to the given scenarios, assess Kit's capability to implement the suggested models, and create new functionalities based on participants' suggestions. The suggested models were implemented in Kit to be used in the third Phase.

The third phase was **"The Recognition."** This phase was designed to assess participants ability to recognize and comprehend the Kit models implemented by the researcher. Programs that are easy to recognize facilitates code editing and expanding as the developer does not have to spend much time to remember what this code was about. The three sessions started with an introductory example to show participants the newly added blocks, and then a multiple choice test was given to the participants to match three Kit models with their correct problem scenarios.

The fourth phase was **"The Interaction."** This phase included four participants from the General Population group only. Each participant had an individual session with no help offered other than a printed user manual of Kit's blocks. Participants were asked to implement some given tasks using Kit environment interactively. This phase was designed to measure users' ability to use Kit, and hopefully come up with more design recommendations by observing participants.

The main objectives of this research were to:

1. Assess non-programmers' ability to solve data-related problems using a prototype of Kit, and highlight their needs and limitations.
2. Assess how usable the prototype of Kit is when used by non-programmers.
3. Assess how capable Kit is to handle typical scenarios that might occur to ordinary people. Also to assess how easy it is for them to recognize and comprehend the implemented models.

My contribution was that I provided some useful insights about non-programmer' ability to manipulate data in a visual flow-based programming environment. A list of recommendations was derived from these insights to be considered by designers of similar environments. Furthermore, I enhanced Kit's usability by adding new blocks and integrating some help facilities (such as help examples and user manuals) in effort to facilitate data-manipulation for non-programmers.

Chapter 2

Related Work

Living in an information revolution age demands that we be able to process the data that increasingly surrounds us. According to recent studies, data is growing yearly at a 50% rate [47]. Data is no longer considered as static or stale. It has become a raw material that plays a critical role in economics and research[94]. Data manipulation applications have to offer support for those who are not able to manipulate data professionally using off-the shelf, and hard to use tools. Citizens should be able to access data even when lack of programming skills exists. There are many political and social reasons [54] behind the emergence of data democratization trend around the world. Furthermore, offering data manipulation support will facilitate exploiting available data sources to answer anyone's curious questions, and hopefully enhance the ability to solve emergent problems.

Kit was developed to support ordinary citizens and give them the ability to manipulate and analyze data without requiring programming knowledge nor experience. In my research, I am evaluating Kit's usability by measuring non-programmers ability to learn, understand and use the developed environment. Through this literature review, I will be elaborating about six main topics to cover my problem domain:

The 1st section illustrates Kit environment, how does it look, and how to be used.

The 2nd section defines usability and presents different methodologies to evaluate it.

The 3rd section previews the Visual Programming Languages (VPLs), how can it be evaluated, and give some examples of this paradigm.

The 4th section explains the flow-based programming concept.

The 5th section discusses the idea of programming for non-programmers, defines both terms, and provides some examples of this trend.

The 6th section defines the term of "Big Data", and discusses democratizing information technology.

2.1 Kit Environment

Kit is a visual, flow-based programming environment. It is supposed to facilitate extracting data from different sources to make any ordinary citizen able to pose problems from their everyday's observations (more about democratizing data can be found in Section 2.6.2).
The audience for Kit includes non-programmers who are interested in accessing, analyzing, and extracting information from the available sources of the ubiquitous data without the need to learn how to program. Non-programmers population includes any individual who is not professionally a programmer (more details about technical definitions for "programming" and "programmer" terms can be found in section 2.5.1).
Kit is built using "Amber" Integrated Development Environment (IDE). Amber is an implementation of the Smalltalk language that runs on top of the Javascript run-time in a web browser. Kit user interface has a pop-up pie menu for the user to start with. This menu is displayed whenever the cursor is clicked on the screen. The menu shows different options to create multi-purpose pre-defined blocks. The menu is shown in Figure 2.1.

Once the user selects an option from the start menu, a sub-menu will appear with corresponding "sub-choices" to choose from consequently to build the program blocks. Blocks are 2-D rectangular glyph that have inTabs (to receive inputs) at the top and outTabs (to produce outputs) at the bottom. Kit's blocks can be categorized into: **Inputs**, **Outputs**, and **Functionalities**. The functionalities can be found under the options **Data**, **Control**, and **Math**.

Inputs are blocks that have outTabs only. OutTabs are where output values are transferred to the next block after the execution of the block is completed and can be found at the bottom of each block. They are the main sources of data to be manipulated. Kit is planned to support [53] the following inputs formats:

1. Simple data inputs of various types, and tables of such simple values as well.
2. Traditional row-based data sources (e.g. spreadsheets or NoSQL databases).

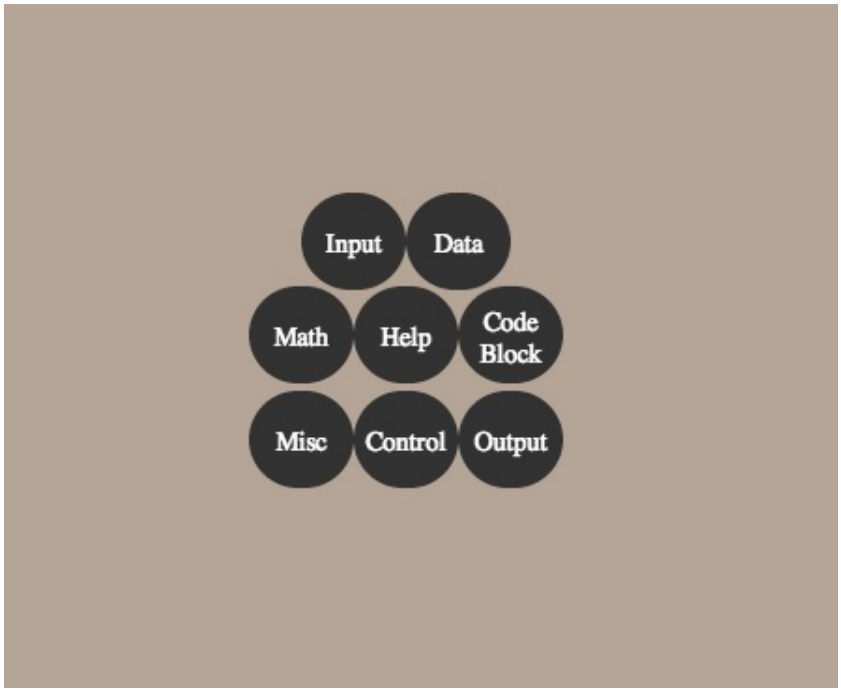


Figure 2.1: A screenshot of Kit's GUI

3. JSON Web API (e.g. Twitter or Facebook).
4. Web page scraping (e.g. a table from web 1.0 website).
5. XML documents.
6. Output values resulting from another block's execution.

At this point[1], Kit supports Simple and Table inputs. It also provides an iterator choice as this block does not require inputs other than the user-defined values. Iterator is a loop representation that has three inTabs representing the user-defined simple inputs of the block. The first inTab contains the loop's index initial value, the middle inTab contains the increment value, and the last inTab contains the terminating value.

Functionalities are some pre-defined interactions between blocks, where data is processed according to the block's role in the code. A block 's execution is completed when all inputs are received and all outputs are produced in a sequentially preserved order. An input value can be changed at any time of the execution. The program will change the output of the affected block and the dependent ones according to the change made

[1] Kit is still under development, and the final version is not completed yet.

dynamically. Kit provides many different functions to be used without having to struggle with the embedded code of the blocks (list of blocks can be found in Appendix C).

Outputs are blocks that have inTabs only. InTabs are where input values are coming from to be processed in the recipient block and can be found at the top of each block. Kit is planned to cover the following output formats:

1. 2-D Graphs.
2. Simple values or tables.
3. Maps.
4. Web applications (e.g. API updates support such as tweeting or posting an update online).

At this point, Kit supports simple values, tables and graphs.

Other than blocks, Kit has pipes that link blocks together to allow data to flow based on the sequence of the piped blocks. Each functionality's output is an input for another block. These pipes represent the natural flow of data visually and provide an easier way to understand data transfer (further details about flow-based paradigm can be found in section 2.4). Pipes are created by clicking on any block's output tab which is located at the bottom side of the block, and drag that pipe to the next block's input tab which is located at the top of the block.

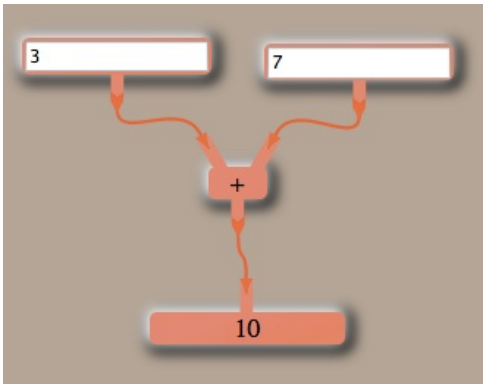


Figure 2.2: Kit Example

Figure 2.2 shows an example on how to add two integers using Kit's blocks and pipes concept. This example shows two simple input blocks that are filled manually

by the user with two numeric values. The input blocks are linked to an addition block (addition can be found under "Math"). The output of the addition block is fed into a simple output block (more Kit examples can be found in Section 3.1).

Using menus and submenus to create multiple blocks and link those blocks with pipes, requires no programming knowledge. All that is required to be able to generate code is the logical analysis of a given problem, and an understanding of the data flow.

Kit is still under evolution, but one of the future planned features is to enable users to create compound blocks that contain group of primitive blocks [53]. This feature enables user to create customized functionalities out of the pre-existent ones. Once blocks are put together, user can minimize the view to hide the contents of the created group of blocks under a newly defined single block. The user should be able to expand the view at any time to edit it or to see the full size of these user-customized compound blocks. This feature is suggested to enable a clean view of the code and a minimal mental confusion that might be caused by the crowded view of multiple blocks.

2.2 Software Usability

The objective of this research is to evaluate the usability of Kit. Hence, this section illustrates the definition of usability and presents some different methodologies to evaluate it.

2.2.1 What is Usability?

Usability is "a quality attribute that evaluates the user interface's ease of use" [68]. Nielsen defined usability with the following attributes:

1. **Learnability:** How easy it is for the user to complete a given task when using the system for the first time.
2. **Efficiency:** How fast fast it is for the user to accomplish a task once he/she learned the design.
3. **Memorability:** How easy it is for the user to remember how to use the system when they return to the design after a period of time.

4. **Errors:** How many error does the user make? how severe? and how easy it is to recover from these errors.

5. **Satisfaction:** How pleasant it is for the user to use the system.

The usability literature strongly asserts that usability of a system is extremely important. Nielsen [68] specified three different reasons to prioritize system's usability. First, the urge to compete with software saturation in the market. If the user was not able to use the system, he or she will look for a more usable one to use; Second, usability is related to users productivity. when employees interact with usable system more work can be done in less time with less mistakes and minimal frustration; Third, a usable system requires less training time and effort. Lastly, when the second and the third reasons are combined together, more profits are to be achieved and more users are expected to use the system. Accordingly, usability concerns have to be identified and considered in parallel with system development [66]. One of the earliest studies about usability has been done by Gould and Lewis [31, p. 300], in which they defined the best principles of a usable software design with these following rules.

- "Early focus on Users and Tasks:" the target audience of a software system should be well understood. Developers should understand their needs, characteristics, behaviours, etc. along with the expected tasks to be performed using the desired software. This understanding enables the developers to incorporate correct requirements into a usable tool.

- "Empirical Measurement:" Software should be tested before reaching the final version in order to facilitate easy updates and modifications. Actual users should have access to a testing version (prototype) of the system, and their performance should be recorded and analyzed.

- "Iterative Design:" Encountered problems from the previous step should be fixed immediately, resulting in many different corrective versions of the same system.

Nielsen extended this model into a more detailed "Usability Engineering Life Cycle" [66]. He strongly believed that usability concerns should be considered before, during, and after software development. Nielsen's usability model is shown in Figure 2.3.

2.2.2 Usability Evaluation

Usability Evaluation is a part of the Usability Engineering process [81]. Usability goals are specified during the requirements collection phase. Then iterative testing is performed during the design/development phase to match the outcome results with the

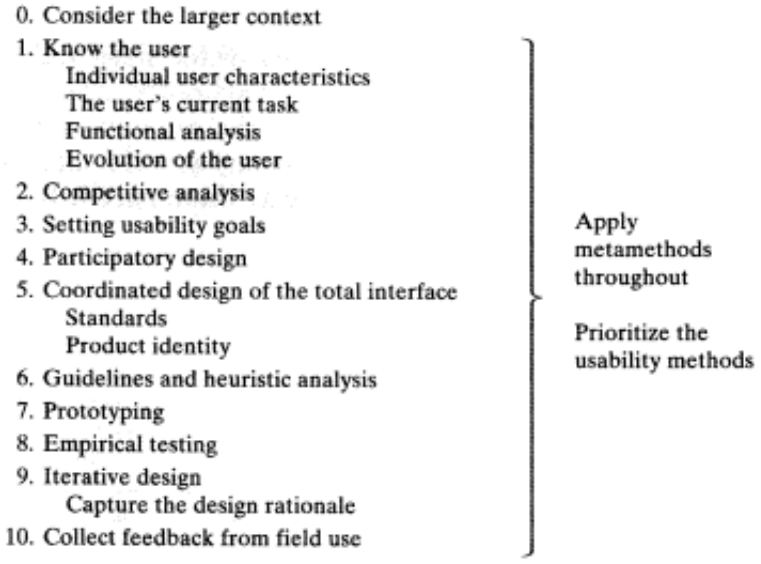


Figure 2.3: Nielsen's Usability Engineering Model (From [66, p. 13])

previously specified goals. Finally, users' feedback are collected once the system is installed and ready for use.

There are three basic methods to evaluate the usability of a system in development. Those methods are categorized based on the evaluation source. The evaluation source can be users, experts, or models. All three methods depend on usability engineers/professionals to design, conduct, analyze, and report the evaluation process.
The following is Scholtz's taxonomy of the usability evaluation methods [81].

1. **User Centred Evaluations**: Kit's usability assessment has mainly been done using different user-centred techniques. In order to perform this type of evaluation, representative users need to be recruited, and representative tasks need to be developed. A well-designed procedure is required to capture potential problems with the evaluated system. This methodology has the advantage of having potential users as a part of the evaluation procedure. However, the drawbacks of this method are that it is expensive and time consuming since subjects need to be recruited successfully, and compensated for their participation. This evaluation method can be categorized into the following sub-categories.

   (a) **Formative Evaluations**: The primary form of data collected in this method is mainly verbal. In formative evaluations, the evaluator tries to improve the

design to identify and diagnose the potential needs or problems that might be encountered by system's users. Formative usability is mainly performed before finalizing the system's design. The earlier the formative evaluation performed, the more useful evaluations are found. This type of evaluation is usually performed on partial prototypes to assess the usability as early as possible. Albert [2, p. 45] used the following metaphor to illustrate the formative usability:

> "When running a formative study, a usability specialist is much like a chef who periodically checks a dish while it is being prepared and makes adjustments to positively impact the end result. The chef might add a little salt, then a few more spices, and finally a dash of chilli pepper right before serving."

Some examples of formative evaluation techniques are: **Focus Groups** (see Section 2.2.2.1), **Scenarios-of-Use** and **Personas** (see Section 2.2.2.2).

(b) **Summative Evaluations**: This is a more formal way of conducting usability evaluations. It is usually performed after the system is completely developed to evaluate the usability characteristics of system. The main objective of running summative evaluation is to determine how well does the system meets its users' goals. Good experimental design is crucial in this evaluation methodology. In addition, directions and materials provided to the users should be well designed and tested. Going back to Albert's metaphor [2, p. 46], he described this technique as follows.

> "Summative Usability is about evaluating the dish after it comes out of the oven. The usability specialist running a summative test is like a food critic who evaluates a few sample dishes at a restaurant or perhaps compares the same meal in multiple restaurants."

Some examples of summative evaluation techniques are: **Questionnaires** (see Section 2.2.2.3), and **Task-Scenarios** tests (see Section 2.2.2.4).

2. **Expert-Based Evaluations**: These methodologies can be categorized under the term of "Usability Inspection." It is performed by usability experts without the need for users' evolvement. These evaluations are less expensive, and less time consuming to be performed. Furthermore, some of these evaluations can be performed at early stages of the system development. However, they do not provide solutions to the detected problems. They also are hard to report since many evaluators are performing them, which means different ways of reporting the same

problem, and different prioritization for the problems' severity. Some examples of these evaluations are:

Cognitive Walkthrough: This methodology focuses on the ease of learning of the system. It evaluates how easy it is for the user to recognize the correct steps of performing a task without rehiring the user to waste the time reading manuals [75] (see Section 2.2.2.5).

Heuristic Evaluation: This methodology tests some specific categories of usability by professional evaluators to detect the User Interface (UI) design's problems [69] (see Section 2.2.2.6).

3. **Model-Based Evaluations**: These methodologies relies on psychological research of humans' perception, cognition, and memory to determine the measurements of cognitive processing and motor movements. These evaluations are relatively inexpensive which means more iterations of the design can be easily tested. However, it is time consuming to produce a model description. Furthermore, models, once generated, need to be validated by testing actual users performing tasks to predict the accuracy of the generated model [81]. An example is:

   GOMS Models: The GOMS model consists of Goals, Operators, Methods, and Selection rules [41]. Goals are the tasks that users are required to complete. they might be categorized into subgoals. These goals and sub-goals are ordered in a hierarchy to be achieved upon their priorities. Operators are actions that are performed in pursuit of achieving goals. Operators can be perceptual, cognitive, or motor acts, or a composite of these. Methods are sequences of required operators to accomplish goals. Selection rules select one of the different possible methods to achieve a goal based on the context. This methodology was not used in this research, because of the tight timeline and the lack of expertise in this area.

The following subsections illustrate with further details some of the techniques presented above.

2.2.2.1 Focus Groups

Recently, focus groups has been an important tool to collect qualitative data among professionals of various disciplines.

In this research, focus groups were used for the following reasons:

"Introduction Phase" (1st phase) (refer to section 3.1)

1. to observe participants ability to understand Kit.
2. to observe participants ability to learn Kit.

"Design Phase" (2nd phase) (refer to section 3.2)

1. to collect more design requirements.
2. to observe participants' different approaches of solving problems.
3. to assess participants ability to map their suggested models into a program.

Based on Kontio et al. [44], the focus group method is a fast and cost effective qualitative method to obtain experiences from experts and (potential) users . Each focus group has 3-12 participants, and the discussion is guided by a moderator who is responsible to keep the discussion in the planned direction. The moderator may be accompanied by an observer who records the session by hand, audio, video, or keyboard. Focus group sessions are highly recommended in risk management studies, requirements prioritization studies, or usability evaluation study.
Kontio [44] specified the following strengths for holding focus group sessions:

1. **"Discovery of new insights:"** having different participants within the same focus group, encourages them to interactively react to the points discussed. The reactions will be somehow reflected and built on others' experiences, which gives a different direction than the personal interview.

2. **"Aided recall:"** When a participant shares a piece of information, most of the participants would agree even if they were not aware of such information to say it themselves.

3. **"Cost efficiency:"** Several users can be interviewed at the same time, which saves the practitioner time, and effort.

4. **"Depth of interview:"** Focus group sessions reveal the concerns along with the reasons justifying those concerns.

5. **"Business benefits to participants:"** Focus groups gives a good opportunity to be introduced to others' benchmarking experiences and practices. Furthermore, focus groups increase participants' networking contacts and cooperations with people of similar work's responsibilities.

Despite the listed advantages, Kontio specified the weaknesses of focus group sessions as following:

1. **"Group dynamics:"** It is harder for the moderator to keep the session on topic compared to the interviews since it is largely controlled by the participants communication styles. This weakness can be overcome by structuring the session, and balancing participation.

2. **"Social acceptability:"** An example would be if someone shared incorrect information, that was disagreed with by the rest of participants. A situation like this will discourage him/her to give any (useful) insights later because of the embarrassment. This weakness can be overcome by moderator taking control in conducting discussion in such scenarios.

3. **"Hidden agenda:"** Some participants may have some hidden agendas due to a business relationship between them, motivation to appear light because of the potential publication of the results, or the internal policy of their company. This weakness can be overcome by setting some explicit privacy rules, and emphasizing the importance of sharing information openly.

4. **"Secrecy:"** Similar to the previous point, participants may hold insights back due to business reasons. This can be overcome the same way as the previous point.

5. **"Limited comprehension:"** When complicated issues are verbally discussed during the session, some participants (and researchers) may be unable to process all the discussed information. This weakness can be overcome by selecting participants of the same experience level, and provide some briefings frequently during the session.

2.2.2.2 Scenario-of-Use and Persona

This methodology was developed by Alan Cooper [21], who introduced it as a design tool for interactive systems. It is "an archetype of a user that is given a name and a face, and it is carefully described in terms of needs, goals, and tasks" [8].
Problem scenarios were used in the "Design Phase" (refer to section 3.2) of this research. Participants were given four problem scenarios with four different personas (can be found in section 3.2.1) to encourage communication and to come up with specific design ideas. Based on Pruitt et al. [77, 93], persona can be used for the following benefits.

1. When the designer has a vague vision about the potential users of the system, persona helps the designer focus on a well defined user's needs, goals, and activities.
2. It provides a shared medium for communication and collaboration among people engaged in the design process, such as designers, developers, testers, writers, managers, marketers, and others.
3. When you have a target user, it is easier to make assumptions and add features since the target users are explicitly defined.
4. Minimal resources are required to create personas, which makes it considerably a cheap design tool.
5. It can be used at a very early stage of the design process.

However, personas can be misused. Some examples of misusing the methodology are described below:

1. When a persona is created, it is designed to have some specific needs and attributes. It is better to create multiple personas than generalizing or stretching one to fit a general description.
2. When designing a persona, people who are engaged to deal with it should be considered. For example, marketing people have different goals than development team. This would require considering each team's goals differently.
3. Some organizations can overuse personas. This might affect the usage of other user-centred methods, data collection, or product evaluation.
4. Personas are created by designers, which means they may have wrong assumptions that may lead to wrong findings.

2.2.2.3 Questionnaire/Surveys

Questionnaires are the most frequently used tools for usability evaluation. It is an inexpensive tool to measure usability and it does not require any specialized or expensive equipments to be completed [49]. In this method, potential users are asked to answer some rating questions. It gives the developer/designers an indication of the users impressions about the developed prototype or software.

In my research, I surveyed participants twice:

By the end of the "Introduction Phase" (See 3.1) to evaluate participants' impressions of the following usability aspects:

1. Understandability: to assess how easy it is for non programmers to understand the presented examples.
2. Learnability : to assess how easy it is to learn the structure of the language and the generation of the code.
3. Effectiveness: to assess how optimistic participants are about Kit's ability to accommodate real problems solutions.
4. Importance: to assess how important it is to provide non-programmers with a usable data manipulation tool.
5. Interest of Learning:to assess participants interest in learning Kit .
6. Lack of Programming effect: to assess our claim that using Kit doesn't require any programming knowledge.

By the end of the "Interaction Phase" (See 3.4) to evaluate participants' impression of the following usability aspects:

1. Memorability: to assess how easy it is to remember the programming language.
2. Understandability: to assess how easy it is to understand Kit.
3. Learnability: to assess how easy it is to learn how to learn Kit.
4. Ease of Use: to assess how easy it is to use the environment to come up with the solutions.

The questionnaires used were not derived from any of the standard questionnaires listed below. The aspects were derived by browsing various generic usability questionnaires available online that relates to the purpose of this study[2].

Questionnaire is a very important tool to evaluate usability [78] for the following reasons:

1. It provides an important feedback from users perspective.
2. The results of questionnaires are comparable across multiple dimensions (e.g. systems, users, or tasks).

[2]Refer to the thesis statement.

3. A questionnaire is a cost and time effective evaluation tool.

4. Provide an evaluation measurement for aspects that are hard to measure such as satisfaction, and pleasure to use.

Some of the standard questionnaires are [92]:

SUMI: This is a commercial questionnaire that comes with complete software that generates scores and reports. It is designed and sold by Human Factor Research Group at University College Cork. This evaluation measures the Global Usability, Efficiency, Affect, Helpfulness, Controllability, and Learnability.

WAMMI: This questionnaire was developed by Jurek Kirakowski and Nigel Claridge to evaluate the the web sites' quality. It measures the Attractiveness, Controllability, Efficiency, Helpfulness, Learnability, and Global Usability.

SUS: This questionnaire was developed by John Brooke in 1986. It is a mature public-domain questionnaire that has been used extensively and recommended strongly for its robustness[92].

QUIS: This questionnaire was developed by Kent Norman in 1987. Ben Schneiderman draw users' attention to this questionnaire when he complemented it in his book "Designing the User Interface."

USE: This questionnaire was developed by Arnold Lund. He specified three dimensions to be evaluated in questionnaires [49], Usefulness, Satisfaction, and Ease of Use/Learning.

CSUQ: This questionnaire was developed by Jim Lewis. This questionnaire asks the user to evaluate 19 different aspects, then to specify the most negative and positive aspects when using the evaluated system.

IsoMetrics: This questionnaire was developed by Guenter Gediga and his team. It works for summative and formative assessments. It was built to help measuring ISO 9241 part 10[3].

[3]Part 10 (1996, withdrawn) "Dialogue principles": Gives ergonomic principles formulated in general terms; they are presented without reference to situations of use, application, environment or technology. These principles are intended to be used in specifications, design and evaluation of dialogues for office work with visual display terminals (VDTs) [39].

2.2.2.4 Task Scenarios

This technique tests the usability by having a real user with some specific list of tasks to be performed using the system. The key in this technique is to make the tasks simple with clear steps. Task Scenarios were used in the "Interaction Phase" (refer to section 3.4) to test users ability to create provided tasks using Kit. The tasks were very focused and simple. They were not dependent to evaluate them independently. The success of the task scenario technique can be achieved by following these guidelines by Jeff Sauro [80]:

1. **"Be specific."** The users need some specific tasks to be performed. "No generalizations" is recommended in this technique because users tend to make more mistakes when the required task is vague.

2. **"Don't tell the user where to click or what to do."** When evaluating usability, user can not be led to the correct implementation. Evaluator needs to be specific, yet he or she can not walk the users through the implementation's steps.

3. **"Use the user's language."** If the user did not understand the terms used in the scenarios, this can lead to false positive[4] results. User's language needs to be considered to avoid confusions.

4. **"Have the correct solution."** This will make it easier to the evaluator to know whether the task was successfully completed or not.

5. **"Don't make the task dependent."** If the user failed in completing one task, dependent tasks will encounter failure too. In that case, it is hard to detect the problems in each different task because of the error propagation.

6. **"Provide context but keep the scenario short."** This will draw the user's attention and keep him/her focused on the required task.

2.2.2.5 Cognitive Walkthroughs

A cognitive walk through is "a precisely specified procedure for simulating a user's cognitive processes as the user interacts with an interface in an effort to accomplish a specific task" [75, p. 748]. The cognitive walkthrough methodology consists of the following two phases.

[4] A false positive is a test result that is incorrect because the test indicated a condition or finding that does not exist.

Preparation Phase: During this stage, evaluators need to select a suite of representative tasks for the the potential users of the system. The selected tasks should be composed of sequences of basic tasks. Each task should be accompanied with a full description. This description is written from the user's point of view, which means it uses general language without any system-specific terms. Next, list the sequence of actions required to perform the specified task, with the fewest number of obstacles to users. If there are multiple sequences of performing the same tasks, all should be listed. If the interface is not fully designed, the designer needs to include more detailed specifications for the interface's appearance before each action, and of the interface's behaviour when the action is correctly performed. Lastly, anticipated users and their initial goals need to be thoroughly specified and described.

Evaluation Phase: This phase analyzes the interaction between the user and the interface. The evaluator needs to find answers for the following questions [97, p. 9]:

1. **"Will the user try to achieve the right effect?"** This measure the user's ability to figure out the required subtasks for achieving a required goal.
2. **"Will the user notice that the correct action is available?"** This measures the visibility of system's elements.
3. **"Will the user associate the correct action with the effect they are trying to achieve?"** This measures the user's ability to use the visible elements correctly to achieve the desired goal.
4. **"If the correct action is performed, will the user see the progress is being made toward solution of their task?"** This measure the user's ability to understand whether his/her actions were correct or not.

2.2.2.6 Heurestic Evaluation

Heuristic evaluation is "an informal method of usability analysis where a number if evaluators are presented with an interface design and asked to comment on it" [69]. Nielsen suggested ten usability heuristics to be considered for User Interface (UI) design [67].

1. **"Visibility of system status:"** Users should be always informed with what is going on through the use of appropriate feedback.

2. **"Match between system and the real world:"** The system should not use any system-oriented terms. Instead, it should speak a general-term language that is understandable by users.

3. **"User control and freedom:"** Users should have an easy access to undo or redo their actions.

4. **"Consistency and standards:"** Users should understand different words, situations, and actions clearly without any ambiguity by following platform conventions.

5. **"Error prevention:"** This can be achieved by eliminating the error-prone conditions, or adding a confirmation option before executing the action.

6. **"Recognition rather than recall:"** Have each possible option visible for the user to minimize its memory load.

7. **"Flexibility and efficiency of use:"** Some options can be added for experts (not necessarily seen by non-programmers) to accelerate the interactions, and allow users to implement frequent actions easily.

8. **"Aesthetic and minimalist design:"** Dialogues should be precise, with the minimal amount of required information.

9. **"Help users recognize, diagnose, and recover from errors:"** Error messages should be precise and easy to understand. They can include some brief suggestion to fix the detected issue.

10. **"Help and documentation:"** Documentation should be available in an easy to understand language, and easy to find location.

2.3 Visual Programming Languages

Kit is a visual programming language (VPL) where users implementation is made out of graphics and visualization instead of a textual code. This section aims to define the visual programming languages, and explains the HCI (Human-Computer Interaction) aspects related to it to support that this approach is suitable for novice users. At the end of this section some visual programming environments are presented to illustrate the representation of this paradigm.

2.3.1 Overview of VPLs

Based on Shu [84, p. 9], "Visual Programming is the use of meaningful graphic representation in the process of programming". Nickerson [65] suggested that the previous definition is too general. Therefore, he shortened it to: "The use of *diagrams* in the process of programming."

In 1990, Myers [63] emphasized the importance of visual programming as it facilitates programming for those users who are not necessarily equipped with the knowledge of understanding nor writing codes. He suggested that further investigation of the use of graphics and visualizations would increase the accessibility of programming to users. He surveyed many advantages of using visual languages and graphics which are listed as below.

1. The multidimensional presentation of visual systems optimizes the use of brain power in a better way compared to the one-dimensional presentation used in the conventional languages with a textual representation.
2. Images and graphs enhances the understandability of the program. Many studies showed that representing data structures by images were helpful to enhance users ability to understand.
3. The graphical representation resembles the mental way of presenting problems and objects in the real world.
4. It makes it easier for non-programmers to understand and generate programs without having to struggle with the textual syntax and mistyping errors.
5. It encapsulates more information about the desired actions with less attention to the required syntax.
6. Visual programming can be a good alternative to describe complex systems (e.g. real-time systems or concurrent processes) that are hard to be described using conventional textual languages.

In contrast to the previously listed advantages of VPLs, many opponents found out that these claims are overstated and not necessarily true. Green and Petre [32] ran empirical studies to compare the comprehensibility of textual programs versus the visual ones. The study targeted expert programmers, and their finding was that visual programs were harder to comprehend compared to the textual ones. They suggested that

the reason lies in the structure of the graphics used in the visual programs which is claimed to be harder to scan by experts who are used to implement with textual languages. This opinion was supported with another study [74] done later by the same team. The study was performed on two different groups of participants, experts and novices. This study investigated the factors that might affect a visual programming language readability. The claim was that the quality of the graphical diagrams, the level of programmers' experience, and even the programming background that programmer is coming from are all important factors effecting user's ability to read and understand the graphical program. They said that "What a reader see is largely a matter of what he or she has learned to look for" [74, p. 69]. This claim explains novice programmers inability to exploit the graphical cues of the visual programming languages.

The performance variation between experts and novices are explainable. The limitation of non-programmers' ability to map code is found previously in textual programming. A study by Fix et al. [28] showed that mental representations of programs differ between experts and novices. The study claims that expert programers were able to develop better mental model than those created by novices. Experts were able to create hierarchal and multi-layered mental model. These layers were explicitly connected to desired goals. Furthermore, experts were able to recognize recurring patterns and connect these patterns correctly. Finally, experts were able to produce well-grounded textual programs. Novice programmers lack such skills which explains the most-likely struggle in mapping their ideas to generate codes.

Boshernitsan and Downes [10] surveyed multiple aspects of the VPLs. They discussed visual languages as a more productive alternative. In their study, they stated that VPLs are accessible to a wider range of users. This claim was derived based on the fact that people store and retrieve information in visual representation which makes it easier to relate the visual program to the real world and eliminate the necessity of translating users visual ideas into an artificial textual code. They claimed that the visual programming trend mitigates the steep learning curve of learning programming. In their study they surveyed the following different classifications of visual programming languages [10, p. 2].

1. **"Purely visual language:"** this category has a pure graphical presentation. To create a code, you need to create different graphical objects and relate/link them in a certain way to perform some specific tasks. No textual code is required at any point of manipulating, debugging, or executing the program. Examples of this

category include languages like VIPR [20] (see section 2.3.3.3) and Prograph [22] (see section 2.3.3.1).

2. **"Hybrid text and visual systems:"** these systems combine both graphical representations and textual underlying code such as Rehearsal World [29].

3. **"Programming by-example systems:"** in these systems, programmers are required to load graphical objects and teach the system how to react with these loaded examples. Pygmalion [85] is an example of this VPL classification.

4. **"Constraint-oriented systems:"** in these systems, visual objects are ruled by predetermined constraints to mimic some natural rules. Such systems are useful in developing simulations such as Thinglab [9], and ARK [86].

5. **Form-based systems:** these systems shared the metaphors used in spreadsheets. Program is represented as a group of interacting elements that change states over the program execution. An example of such systems is Forms /3 [15].

Kit falls under the first category. It is a purely visual programming language, where user do not need to include any textual code.

2.3.2 Usability of VPLs

Green and Petre [35] introduced a new evaluation approach to analyze the usability of visual programming environments specifically. They claimed that their approach is task-specific and it focuses on the processes rather than the final product. They emphasized the importance of considering characteristics of anticipated users and their potential tasks and needs. In their literature review, they surveyed lots of important aspects that explains the psychology of programming. These aspects are associated with the transition between problem domain and program domain. Some of these aspects are summarized below.

1. Programming notation can not highlight all the underlying aspects of information all at once. Some aspects are exposed at the expense of others [33].

2. The presentation of the used programming language needs to accommodate the mental representation of information.

3. Translating mental representation into programming plans can be very stressing and may lead to programming mistakes due to statements dislocation and/or code composition failure [87, 82].

4. The process of programming should offer a flexible order of creating components based on user's preference [95, 4]. Components do not have to be created sequentially.

5. The environment should provide readable code with large window of access for easier editing and expanding to the code [34, 23].

6. The environment should support user's needs of primitives to provide a natural mapping between problem domain and program domain [46].

Green and petre surveyed the HCI (Human-Computer Interaction) of programming as well, in which the main focus is on how programmer interacts with the code itself. They emphasized the importance of having an effective way to measure the success of code management (in terms of layout, and text control) in visual environments. Another important HCI area to explore was searching and browsing in visual environments. Programmer should be able to reuse pieces of code and expand it easily. Furthermore, an environment should support programmers needs in "finding the way and finding the way back" when it comes to keeping track of development trails.

In their research they suggested new cognitive dimensions to assess the usability of visual programming environments. These dimensions are listed below .

"Abstraction[5] Gradient" Users should be able to map problem concepts onto programming entities with appropriate abstraction. This abstraction can be a challenge for novice programmers since programming abstract terms such as variables, conditionals, loops, etc. may be far from their mental models. To overcome this challenge, abstraction need to be incremental, and abstracting groups of components should be offered to make easier to implement. In spite of its problematic learnability, once abstraction is well-chosen, it increases the code comprehensibility and decreases error-proneness.

"Closeness of Mapping" Users should not struggle with overwhelming lexical constraints, or large number of primitives. The representation of the program should be as simple and as clean as possible to enhance end-users ability to use such environments. Meanwhile, visual environments should provide programming metaphors that are similar to the user's domain. This should provide task transparency to end-users.

[5] "Abstraction is a grouping of elements to be treated as one entity, whether just for convenience or to change the conceptual structure" [35, p. 144].

"Consistency" Users will find a language easy to learn when it maintains its consistency syntactically and structurally. When an environment is consistent it is easier to guess commands based on previous related knowledge.

"Diffuseness/Terseness" An environment should have the least possible number of lexemes (different programming entities) without sacrificing the understandability of the code by overusing abbreviations.

"Error-proneness" An environment should minimize the possibility of slips occurrences[6]. In textual programmers one of the most common slips is mistyping the call of variables. VPLs reduced the occurrences of slips since it has different syntactic structure, and more explicit dependencies.

"Hard Mental Operations" An environment should avoid "Brain twisters." These mentally complicated operations decrease the comprehensibility of a programming language. Green and Petre defined hard mental operations by two properties. The first one is when the issue lies on the notational level rather than the semantic one (such as conditionals and negations). The second is when such offendings are conjuncted (e.g. nested negations).

"Hidden Dependencies" User should be able to trace entities without having a difficulty to detect the dependencies between the components. All dependencies should be visible and traceable. In conventional languages, this can be solved by using cross references and call graphs, whereas data flow languages use antecedents and dependents trees. VPLs that use the box and line representation provide an easy representation to detect dependencies at a local level.

"Premature Commitment" An environment should avoid requiring a user to make decisions when pieces of related information are still unavailable. This situation usually occurs under the following circumstances: "the notation contains many internal dependencies; the medium or working environment constrains the order of doing things; and the order is inappropriate" [35, p. 155]. However, users can overcome this issue by preparing ahead some approximate values manually, or by creating some place holders to be filled in a later stage of the development process. VPLs representation allows users to develop the program in an order-free way giving them the freedom to expand in any direction.

[6]Slip is "doing something you didn't mean to do, where you knew what to do all along but somehow did it wrong" [35, p. 149].

"Progressive Evaluation" An environment should supports users ability to evaluate their programming progress gradually and more frequently before the program is completed.

"Role-Expressiveness" Users should be able to understand the role of each segment of code in an easy to read and understand notation. This feature can be enhanced by using meaningful identities, well-structured modularity, use of secondary notation, and use of explicit description level.

"Secondary Notation and Escape from Formalism" An environment should provide users with extra source of informal data to make the code more readable such as indentation, user comments, statements grouping, etc. Such sources of information provide a communication mean between the developer and the reader of the code. In VPLs, users can pay extra attention to organize the layout of blocks and lines to have a cleaner data flow diagrams to enhance readability and highlight parallelism.

"Viscosity: Resistance to Local Changes" An environment should require a minimal effort when users need to make small changes. This can be achieved by providing global update techniques/tools, and introducing abstractions. The small changes need to be applied easily no matter how big or complicated the code is. In VPLs this should be done without the need of rewiring blocks of a visual program.

"Visibility and Juxtaposability" *Visibility* is measured by how easy it is to access a required material (with no hidden dependencies) whether this material is ready to be accessed and visible, or if it needs to be identified first in order to be accessed. *Juxtaposability* is the ability to compare two different pieces of the same code at the same time side by side. A programming environment should provide users with both features to enhance its usability. Otherwise, programmers have to exhaust their working memory, or can use a hard copy of the required part of code for the reason of code comparison or have a doubled view to create a new identical environment.

The authors suggested that these cognitive dimension should be combined with other usability evaluation techniques to obtain the best results. More usability evaluation techniques are listed in section 2.2.2.

2.3.3 Examples of VPLs

In this section, many examples of visual programming languages are illustrated (In an alphabetical order). Most of these languages aimed to facilitate data manipulation for non-expert programmers. The representation of each language is briefly explained to get a better understanding of how visual programming environments can be designed.

2.3.3.1 Prograph

Prograph [56] is another visual object-oriented language. Prograph was influenced by a mix of different languages and paradigms such as APL, LISP, and FP. The underlying database system can be traced similar to FQL (Functional Query Language), DAPLEX, and LISP. Control and iteration logic is following the ALGOL (Algorithmic Language) logic. The language has lots of commons with the GPL (Graphical Programming Language) even though designers were not aware of the existence of such a paradigm.

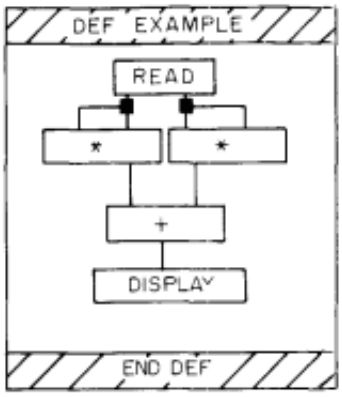


Figure 2.4: A Model Example of Prograph Program (From [56, p. 93])

Prograph program looks like Figure 2.4. Every prograph program contains a top bar, a body, and a bottom bar. The top bar indicates the user-defined name of the program. The body contains all the operation boxes required to process the data. Similar to Kit, operation boxes are connected using wires to represent the data flow from a box to another. Inputs of each box are connected to the top of the box, whereas outputs exit from the bottom.

In addition to the pre-defined primitive boxes, users are allowed to define as many compound boxes as required by putting primitives together. Prograph offers implementation of many programming entities such as conditionals shown in figure 2.5, and

loops shown in figure 2.6. What is different in Prograph is that identifiers are not used at all in the representation of the code.

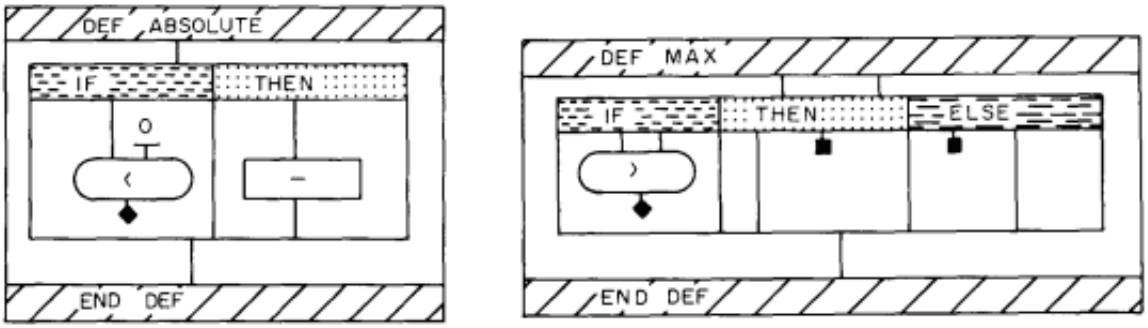


Figure 2.5: Prograph Representation of Conditional Statements (From [56, p. 94])

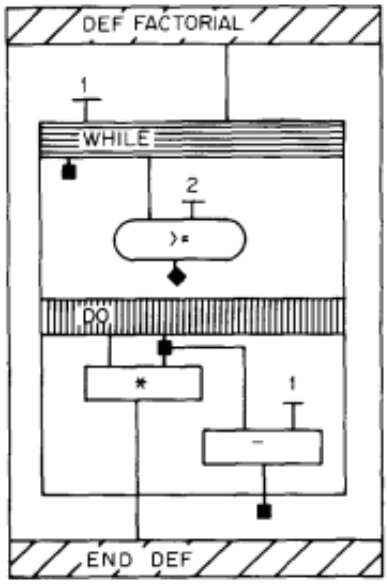


Figure 2.6: Prograph Representation of a `while` loop (From [56, p. 98])

2.3.3.2 Raptor

RAPTOR [16] is a visual programming language that helps students to visualize their algorithms and trace the execution visually. It is a combination of Ada, C#, and C++. The main purpose of developing Raptor is to overcome students' challenge of understanding the programming process in an introductory programming course. It is sometime overwhelming to understand the structure of the language and remember

the small syntactical details associated with it which annoys and distracts their attention. The objective was to allow users to focus on programming activity rather than worrying about the syntax and the structure of the language itself. The environments emphasizes the ordered reasoning of the algorithm and enable the users to run the task continuously or in a step-by-step sequence according to their preference and needs to gain a better understanding of the programming terminology.
Raptor supports lots of programming entities such as conditionals, loops, function calls, assignments, inputs and outputs. Figure 2.7 shows how the environment looks like, and what kind of entities can the user implement.

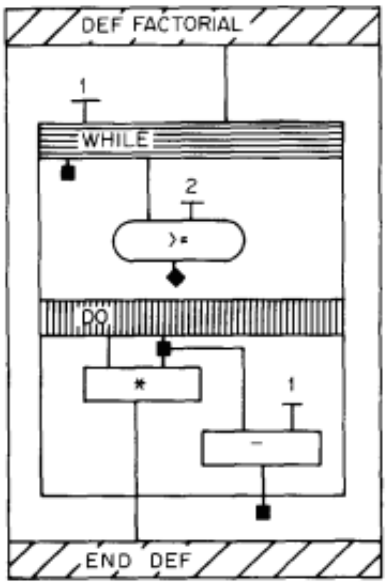


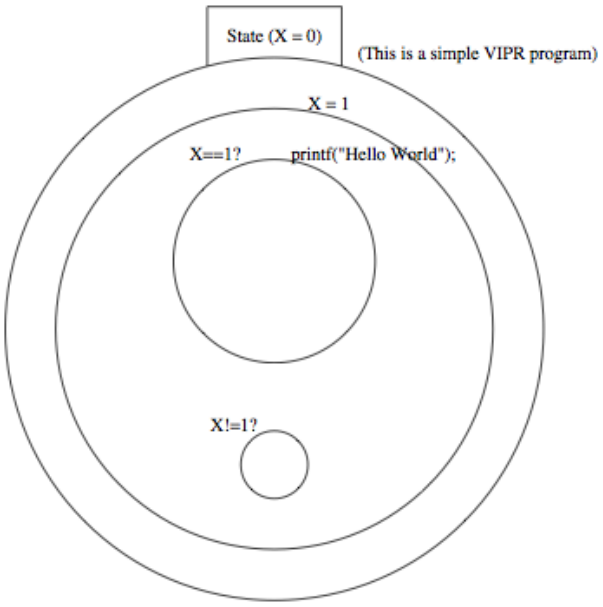
Figure 2.7: A Screenshot of Raptors Environment (From [16])

RAPTOR enforces syntax checking at the editing stage to prevent syntactically invalid programs. Secondary notations are available by right clicking on the desired entity, and it will show as a talking bubble next to the selected entity. Both features should considerably enhance tool's usability for novice programmers and make the programming task easier and less stressful by allowing users to pay more attention on the problem itself rather than the syntax of the program.

2.3.3.3 VIPR

VIPR is a visual programming language that it developed by Citrin et al. [19, 20]. It has been developed to facilitate OOP (Object-Oriented Programming) understanding

by developers. The OOP paradigm offers lots of useful features that can enhance the quality of the generated code. Features like polymorphism, inheritance, and dynamic dispatch are, unfortunately, hard to write, understand and modify. Hence, VIPR offered a visual object oriented environment that facilitates the use of the OOP paradigm without having to struggle with its code understandability. VIPR is a purely graphical tool that uses simple graphical rules.

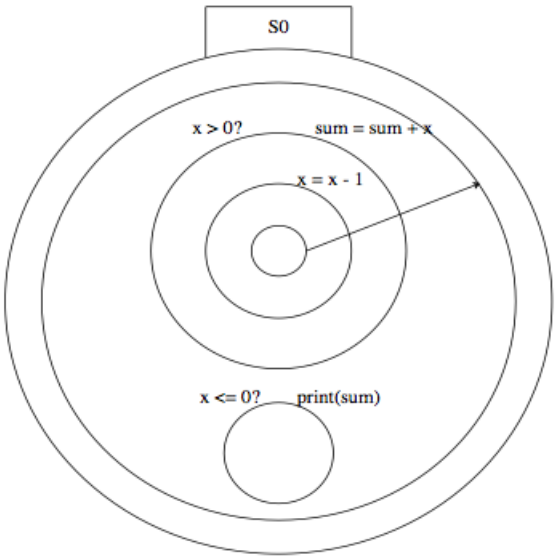



```
/* This is a simple VIPR program */
int x;
void main ()
{
    x = 1;
    if (x == 1) {
        printf("Hello World");
    }
}
```

Figure 2.8: VIPR Hello World and C Equivalent (From [19, p. 5])

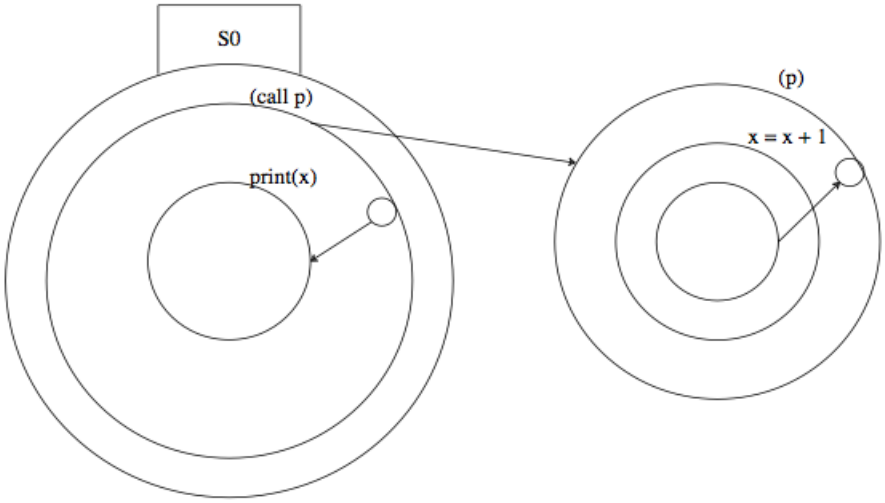
Figure 2.8 presents the implementation of Hello program. The program consists of nested circles. Each circle represents a code statement. The if-statement is represented as two graphical choices, one to be executed according to the evaluation of the corresponding conditional statements. The code starts from the outer circle, and user creates as many inner circles as required based on the sequence of statements to be executed. If two statements are to be used within the same level, their corresponding circles are to be adjacent sharing the same parent circle.

VIPR provides a visual representation for: identifiers that are defined at the top of the circle that matches their scope, loops which are presented in Figure 2.9 , and functions calls and returns as presented in Figure 2.10. As can be seen in the figures, arrows are used to present the switch of control among different circles.




```
while (x > 0) {
   sum = sum + x;
   x = x - 1;
}
print(sum);
```

Figure 2.9: VIPR Representation of a `while` loop (From [19, p. 9])




```
void main()
{
    call p;
    print(x);
}

void p ()
{
   x = x + 1
   return;
}
```

Figure 2.10: VIPR Representation of Function Call and Return (From [19, p. 10])

2.4 Flow Based Programming

Kit is a Flow-Based Programming (FBP) environment. It falls under the visual programming languages paradigm, yet it is specialized in manipulated data in a flow-based representation. In this section, FBP is explained based on its inventor's publications [58][59][61] except as specified. This literature review is completely cited from the specified references, without making unjustified (personal) claims.

Flow-Based Programming (FBP) is a programming paradigm invented in the early 1970s by J. Paul Morrison. He defined it as a network of connected black boxes, each box represents a process. The main idea of the FBP paradigm is having an application that does not run in a linear order executing one process at a time sequentially during its execution. Instead, the application works as "a network of asynchronous processes communicating by means of streams of structured data chunks called, *Information packets*" [61]. This paradigm was designed to focus on data and its transformations in order to obtain the desired output. The need for such a paradigm emerged because of the increasing complication of developing business applications and maintaining them, added to that the increasing programming blacklog when using conventional programming methodologies.

Developers needed a new methodology to offer solutions in a natural way and with a higher productivity rate. The FBP concept offers processes as black boxes that hide the implementation inside. Those blocks can be reused as many as required either by the developer who implemented them or by other developers who might need them. A new process is easily implemented, when it is not previously created and its cost is justified. Such modularity can achieve better productivity, maintainability, and reliability. The FBP concept works like a car assembly line, where blocks are implemented individually and then assembled to form what looks like a UML representation [62] as the one shown in Figure 2.11.

Further, FBP has a consistent view from the high level design, down to the implementation. It has a natural layout, that can be used as documentation to the code since it is very understandable even for non-programmers. It makes computers conform like humans instead of enforcing people to think like computers. The idea of FBP emulates the natural flow of data, and it lessens the gap of the way of thinking between developers and users, which enables more non-programmers to be engaged during the different

stages of building software [89].

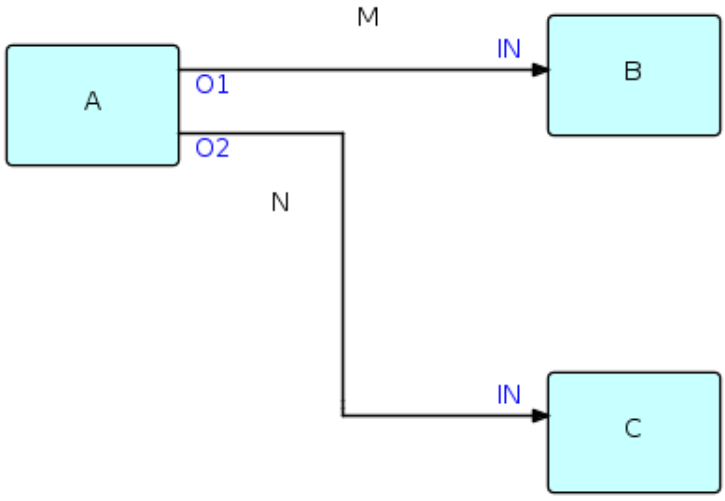


Figure 2.11: A Simple FBP Diagram (From [61])

As shown in Figure 2.11, in FBP an application's network consists of processes (A,B, and C). Each process has input(s) (IN) and output(s) (O1 and O2) ports. The processes are connected with connectors or *bounded buffers* (M and N).
When the process receives an Information Packet (IP) through its input port, the IP is processed according to the process role indicated in the hidden code assigned to this process. Once the manipulation is performed on the IP, the process output the IP through its output port to get it transferred to the next destination. Each connection can be listed as the following quadruplet: process name, output port, input port, process name.
Throughout the previous sequence, components are guided as follows:

- Connections have a fixed capacity with a predetermined number of allowed IPs at a time.
- Ports can be simple or array type.
- If a connection is full, the feeding process will be suspended.
- If a connection is empty, the receiving connection will be suspended.
- Each process has its own storage space, control blocks, parameters, etc. that are not shared with other processes.

- Each IP is owned by one process at a time, or is in the process of being transferred from one to another.
- IP can contain data or signals (will be called bracket IP in that case) to group data into a sequential pattern to form a *substream* which has the capability to be nested.
- IPs can be chained as well into a *tree stream* which moves through the network as an object.

As can be seen from the previous patterns, the relation between processes in FBP is cooperative rather than hierarchical. FBP is based on delegating tasks from one block to another to get the job done easily and quickly. A simple illustration to show the efficiency of such approach would be [62]:

> "Let us say I run a restaurant. Bob is a cook and Dale is a waiter. Dale's task is to wait on every customer that comes through the door. In order to get that done, Dale will seat the customers at a table and take their order, he then delegates the task to make the food for Bob. Bob cooks the food and when it is done he gives a task to Dale to deliver the food to the customers."

As can be noticed, Bob, and Dale are processes. The delegation here from one process to the other is the connection. To show the real benefit of FBP concept, the previous restaurant's example can be extended as following:

> "Now say you want to be able to start serving Mexican food at the restaurant. We would need a cook who knows how to cook it! Think how we would design our connections and components to allow for this new cook?"

Can you see the difference here? Bob can not cook Mexican. He can not be connected to Dale anymore. Instead, Dale needs to be connected to the kitchen to look for the right cook and notify him/her with customer's order.

Using the previous methodology, FBP is able to form countless patterns by changing connections of blocks without the need of changing the code. This characteristic of *Configurable Modularity* adds a great advantage to engineer the code [60]. FBP has

been implemented successfully using Java (JavaFBP), C# (C#FBP), and C++ (CppFBP). The popularity of FBP is gradually increasing for its simplicity and efficiency in dealing with data. Following is a list of some projects which have been built to introduce FBP concepts to the public:

1. Marten: `http://www.andescotia.com/products/marten/`
2. NoFlo: `http://noflojs.org/`
3. meemoo: `http://meemoo.org/`
4. quadrigram: `http://www.quadrigram.com/`
5. Quartz Composer: `https://developer.apple.com/library/mac/documentation/graphicsimaging/conceptual/quartzcomposeruserguide/qc_concepts/qc_concepts.html#//apple_ref/doc/uid/TP40005381-CH212-SW9`
6. VUO `http://vuo.org/`
7. Vvvv `http://vvvv.org/documentation/documentation`
8. LabVIEW: `http://sine.ni.com/psp/app/doc/p/id/psp-357`

2.5 Programming for Non-Programmers

In this section, I illustrated the technical definitions of "programming" and "programmer" terms, showed some previous studies about non-programmers way of thinking and how can they be supported, and finally present a list of various projects aimed to facilitate programming for the audience of non-programmers.

2.5.1 What Is Programming and Who Is A Programmer?

Blackwell [7] did a detailed research on what is programming. In his study he was able to collect these early definitions of programming:

1. "Programming ... is basically a process of translating from the language convenient to human beings to the language convenient to the computer." by McCracken[7] – 1957 [7, p. 1]

[7] "Daniel D. McCracken (July 23, 1930 – July 30, 2011) was a computer scientist in the United States. He was a Professor of Computer Sciences at the City College of New York, and the author of over two

2. Programming is the process of translating the mathematical formulation into the language of the computing machine. The previous definition was written by Booth[8] – 1958

3. "This sequence [of basic operations] is called the program, and the process of preparing it is called programming" [102, p. 4], by Marshal Henry Wrubel – 1959

Based on the previous definitions, programming is the process of converting a way of solving a problem into a special language that is compatible with the computer in order to develop what is called a program to achieve a specific task. Programming activities have evolved through the years from doing specific mathematical operations into a wider range of general data processing tasks.
During our research, we did not provide any technical definition for the programming activity. When I referred to programming, I gave the participants the choice to interpret the term based on their understanding of the programming process.

In the same study, Blackwell discussed the evolution of definitions of "programmer" as follows:

1. In his book "Psychology of Computer Programming", Weinberg [96] used the *programmer* term to indicate any computer's user, including the level-entry users.

2. There was some clear distinction between the terms *analyst* and *programmer*, until the emergence of 4GLs in the 1980s, which offered a decent level of programming abstraction.

3. In the 1990 *analysts*, and *programmers* were categorized as programmers.

4. A new term of *end-user programming* emerges, where the user who is not specialized in programming can perform some specific-domain implementations. Those users can still be categorized under the big category of non-programmers as their programming knowledge is limited to the tool they are using to complete the required task. An example of end-user programming would be customizing spreadsheets.

dozen textbooks on computer programming. His A Guide to Fortran Programming (Wiley, 1961) and its successors were the standard textbooks on that language for over two decades. His books have been translated into fourteen languages" [98].

[8]"Kathleen Booth (née Britten) is credited with writing the first assembly language and the design of the assembler and autocode (ARC and APE(X)C) for the first computer systems at Birkbeck College, University of London" [99].

Throughout this study, non programmers are those who are not professionally programmers. They can be end-users, novices, beginners, etc.

2.5.2 Research About Non-Programmers

Programming by non-programmers has been a research topic for a long time. In 1974, Miller [57] performed some experimental studies to understand how non-programmers think by reorganizing natural English commands into programs to solve some given problems. Non-programmers were mainly challenged with the concepts of disjunction, and negation. He emphasized the research of programming as a problem solving approach. He believed that it is important to have a procedure specification that lists a sequence of the required actions to be performed by subjects. Doing so, will give the ability to measure the individuals' behavioural variation to be investigated and analyzed. He stated that programming as a problem solving activity is ruled by many external factors such as: creativity, planning, and inductive reasoning. Which means variations are existent even within the unified domain. Finally, he mentioned the possibility of measuring programmers' behaviour with the use of transaction-recording techniques.

In 1977, Keppel and Kropp [42] developed a user-friendly environment for non-programmers to design, execute, file retrieve, and modify forms. In this study they emphasized the importance of visualizing the data interactively for non programmers as a key to facilitate learning and editing. They claimed that the ease of use of their tool did not limit its efficacy. Professional programmers would still be able to access and manipulate the invisible code to customize the tool according to their needs.

A similar study was performed by Dunsmore in 1980 [24]. He used an empirical study of how to design an interactive problem solving computer to be used by non-programmers. In his study, he investigated user's ability to do tasks like: sign-on, sign-off, communicate with the system, and match his /her needs with the correct packages to be used. He highlighted in his work the difference between non-programmers who want to learn the language, and those who want to performs specific tasks without the need to build a code from scratch. Based on this, the power of software is not measured solely by its ability of performing tasks, but also by its ability to adapt the user's specific needs without requiring him/her to pay the expense of having to learn a new language.

Mental representation of programs was an important search field for Fix et al. [28]

who defined five different characteristics of representing a computer program mentally. These characteristics are: "Hierarchal structure, explicit mapping of code to goals, foundation on recognition of recurring patterns, connection of knowledge, and grounding in the program text." Their study showed that experts' mental models contained all these characteristics. Per contra, novice users were unable to cover these areas in their mental models.

More recent studies in the 21st century continued to investigate non-programmers' issues, and how to create a friendly environment for them.
In 2001, Pane et al. [72] performed a two stage empirical study, The main objective was to understand the natural language structure in order to improve the understandability of the computer programs by non-programmers. The first stage was to examine children's suggested commands to a set of tasks. The second stage was derived from the first stage's results. It was designed to have adults and children to answer 11 problem scenarios that cover some essential programming concepts. The study concluded that non-programmers spend lots of their mental energy trying to translate their ideas into a program-like command. Researchers' insight was that the difficulty of learning a programming languages is derived from its formality. Programming languages have to specify commands in a simple and precise format, and that is what makes them different than the natural human structure of language.

In 2004 Myers, Pane, and Ko [64] extended the work described in the previous paragraph, and investigated the reasons why the conventional structure of programming languages is resilient to change even after thirty years of research proving that it is far from how humans think. They found out that the research in the areas of Human Computer Interaction (HCI), and Empirical Studies of Programmers (EPS) have set many useful guidelines for designing systems, but lack some answers for questions like, what is the closest programming paradigm to natural human thinking? They tried to dig deeper for an answer by conducting further studies with children and adults to understand their natural way of thinking when solving problems. They observed the following: (a) Users preferred to design with event-based structures. (b) Users preferred aggregate operators (using a set of instruction as one complicated block) more than iteration as in: "Move everyone below the fifth place down by one."(c) Boolean expressions were rarely used and exhibited a high error rate when used. (d) Users sketched pictures to design the layout of the system, and texts to describe actions and behaviour.

Eckerdal, and Berglund [25] tried to understand how non-programmers think about programming. For that purpose, they interviewed some first year students from the computer science department who were, at the time, taking an object oriented programming course. The students were asked to describe their thinking. Students were unable to give a precise definition. Some of them talked about programming as a different way of thinking, or added that it is written with some magic characters. Meanwhile, some of them were able to give some phrases as: learning to program is a way of thinking, which enables problem solving, and which is experienced as a "method" thinking. This research highlighted the importance of understanding the abstract concepts in order to be able to learn them.

The United States Military Academy added a mandatory programming course to introduce all the first year's students to programming foundations. In 2008, Ring decided to investigate what might motivate non-programmers to learn programming concepts. He monitored the performance of the students which improved considerably when using certain methodologies: (a) a Visual Design Tool, which facilitated drawing flowcharts. This methodology gave students the capability of learning the concepts of sequence, iteration, and selection without the need to code. The code was embedded within the blocks of the flowchart; (b) introducing Visual and Graphics Problems, which means adding graphics problems to the curriculum to help understanding the underlying programming concepts better; (c) immediate Feedback During Testing, in which the user's performance is evaluated with a feed back instantly wether it was positive or negative in which they added colour to the messages of successful tests, and red colour to the failed ones.

2.5.3 Projects for Non-Programmers

There have been many efforts to make it possible for non-programmers to learn programming's basics as an increasingly essential skill. Some examples of such efforts can be found in the following links:

1. Khan Academy: `https://www.khanacademy.org/computing/cs`
2. Code Academy: `http://www.codecademy.com/`
3. Learning Labs: `http://learninglabs.org/`
   it includes: Ladies Learning Code, Girls Learning Code, Kids Learning Code and other organizations.

4. Programming Without Coding: `http://doublesvsoop.sourceforge.net/`

5. "Anybody can learn" campaign on: `http://code.org/`

Meanwhile, many research projects focus on offering some programming facilities that enable non-programmers to perform programmers task without the need to master the programming skills. Following are some of these projects:

EAZYTAGZ [79] is one example of a tool that targets non-programmers. It was built to help educators develop their teaching portals. As an alternative to a commercial system, which might be usable but not flexible for these non-programmers, EAZYTAGZ offers a set of building blocks that contains custom HTML code hidden under a new tag name. Those tags make it easier for the user to read and maintain because it only requires the tag name without the need to show the embedded code inside each tag. This makes the code simpler and more easily modified by non-programmers. The tool was provided with a library to support users in four modules: publishing online material, discussion forums, electronic submission for students' work, and online testing.

"Pseudo Tutor" tool for building intelligent tutoring systems has also been suggested for non programmers. Koedinger et al. [43] suggested a model that enabled non-programmers to author a system that showed intelligence behaviour without requiring any AI programming. The educator was provided with a Tutor GUI Builder to facilitate creating the desired UI, and a Behaviour Recorder in which the educator provides a variety of alternative ways of solving the same problem. The Behaviour Recorder, when switched into "pseudo tutor" mode, was able trace students' performance step by step during solving problems, provide hints and feedback that were previously designed by the educator. The educator had full control of designing the models of answers, hints, feedbacks, and knowledge labels. Knowledge labels were used in a skills matrix that is used to show the knowledge elements used in each designed problem. The developers of this system claimed that it can be used in various disciplines such as Economics, Math, LSAT, and Language Learning tutoring.

Another idea was to facilitate the use of an intelligent agent-based system called POSH by integrating a toolkit that allows entry-level development for non programmers to create their own animated characters [13]. Such systems were highly recommended in gaming development, where the story writer can create the same exact character he imagined when he/she designed the narrative. Such a system kept program-

ming as simple as it could be, and facilitated complex programming in a doable and easy way for non-programmers, meanwhile, it did not affect experts' ability to do more complex tasks and extensions. It provided the user with rapid prototyping, to create a character with its associated goals and scenes in a reasonably short period of time.

MIST is another project aiming at non-programmers (end-users) who are reusing others' user interface to facilitate the development of their projects' interface [6]. This tool was designed and developed to make it possible for them to reshape a previously developed dynamically-configureable, interactive user interface to match their needs in creating a new one for their environments without the need of building it from scratch. This project was extended to include the facility of integrating new pieces of code to evolve the user interface in order to match the users' evolving requirements and needs without the need of understanding the code [5]. Their goal was to solve the struggle the non-programmer experienced understanding the underlying code to extract the required pieces. They developed a View Model Controller (VMC) to decouple the logical part away from the user interface components.

One more example was Dinah [38], which is a project to facilitate selection from code with graphical output, that is similar to the user's interest, by segmenting it for the user to pick the piece he/she needs. When the user finds the desired example, Dinah will show the output result in parallel with its matching piece of code which is highlighted synchronously to enable the user to pause whenever he/she sees the desired output part to have the code reused as required. Dinah's performance was evaluated later with more empirical studies [37]. The developers found out that it was hard for the user when they searched on the web to make sure that they are selecting the correct example. Furthermore, it was hard for the user to pick the correct piece of code when the program has multiple tasks.

2.6 Big Data

In this section, importance of having data-manipulation and analysis skills are justified by explaining the term of Big Data. It has been so common lately to hear about this term as data growth is uncontrollably increasing around the world. The second main idea in this section is justifying the need to support non-programmers with data manipulation tools specifically.

2.6.1 What is Big Data?

Based on Fisher et al. [27] Big Data is defined as "data that can not be handled and processed in [a] straight forward manner" [27, p. 53]. They stated that Big Data has been an interesting topic not only for developers, but also in various humanistic fields. Having personal digital documentation for each individual throughout the realms of social media, makes Big Data an interesting research source for those who are working in the "human behaviour" discipline. Big Data has been an interesting source of data for HCI (Human Computer Interaction) researchers, and UI (User Interface) designers. One way of using Big data in the HCI field would be to collect and analyze A/B testing results. In A/B testing methodology, users are asked to try multiple user interfaces for the same tool. Users' different reactions are used to choose the most usable version. Another example of using Big Data in the HCI area would be when researchers look for the most frequently asked questions by users for certain applications or websites to determine what usability issues need to be fixed. Another example is online gaming websites, when they collect and analyze the data related to the way users are playing these games. Such data will help update these websites' offerings based on analyzed insights.
The need for utilizing Big Data created the discipline of "Big Data Analytics" which transfers the humongous size of raw data down to a smaller size and higher quality data. This can overcome the limitations of the sampling techniques, that are "prone to potential error and bias" [27, p. 51], and makes it possible to process the data as a whole, no matter how big, dynamic, multi-sourced, or multi-formatted it is.

Boyd and Crawford [11] suggested that the "Big Data" term is poorly named. They claimed that Big Data does not necessarily have to be "big." Instead, they suggested that Big Data has been highlighted lately because of its relationship to other data. They stated that the value of Big Data is gained because of the ability of eliciting patterns and relations out of the data sources to develop useful insights in the academic and the industrial fields rather than how big this data really is. However, the limited access and skills to use Big Data caused the research findings to be stumped for those people who are unable to use it. This definition of Big Data is the closest sense to Kit's form of source data ("Ubiquitous Data").

Jacobs [40] defined Big Data as "data whose size forces us to look beyond the tried-and-true methods that are prevalent at that time" [40, p. 44]. In his experiments he tried to reach the minimum quantity of Big Data by generating fake datasets and ana-

lyze the machine performance manipulating them as they grow. He suggested that Big Data is mainly caused by the replication of data generated to improve the quality of the analytic insights. He suggested that what makes data big is the repeated observations over time and space.

Manyika [50] listed reasons on why Big Data is increasingly valuable. Big Data provides an important information source at a higher transparency and availability level. Meanwhile, the digitally collected information can provide the industry with a complete performance history to make better future plans and management policies. The efficient use of Big Data can save governments millions of dollars in many different domains. By 2018, the increasing need of data analysis will encounter a labour-market shortage of around 140,000 to 190,000 analysts in the United States alone, which means having skilled data analysts is going to be a huge issue in the few coming years.

Lohr[47] highlighted the need of not only powerful search engines, but also intelligent ones. He listed Google[9] and Siri [10] as successful examples of using Big Data for general-purposes. An intelligent system like Siri, is able to perform a high level of knowledge search, and the knowledge base is extended by the millions of question asked by users around the world. Such systems are increasingly needed to use the available data resources.

2.6.2 Who Can Use Big Data? And Why?

The short answer is: *Anyone*. Having a wealth of data available to almost everyone is only possible if we support people with ways to access it. Even simple daily activities can be related to some implicit computations of raw data when the facility of having it cleaned and well-organized is available. An example of such approach would be if someone noticed a correlation between two different factors' occurrences in such a way: "I wonder if there is a correlation between street lighting and the occurring of potholes."[52]. Such findings will not only answer people's questions, but also will affect the quality of life when every citizen has the ability to be a researcher. People's observations and ideas can be shared to get further investigations by the responsible officials or the interested organizations.

[9] The famous search engine[30].

[10] Siri is a smart knowledge navigator that works on Apples's iOS, and it lets you use your voice to give commands [3].

It is strongly believed that programming needs to be democratized for various political and social reasons [54]. Based on Wing's research [100, 101], "Computational Thinking" will be one of the fundamental skills, such as reading and writing, that everyone should have by the middle of the 21st century. The computational thinking is derived by a combination of the three factors of science, technology, and society. Out of her extensive research in the area, she had the vision of having computational thinking as an important instrument to new discoveries in nearly all - if not all - disciplines. Meanwhile, her second vision was to integrate computational thinking as a part of childhood education, to embrace the importance of computational thinking and make it a growing skill from the early years of life.

Chapter 3

Experimental Design

This research is a part of the "Kit" Project. Kit is a visual flow-based environment built to support non-programmers (who are not trained programmers) in the field of data manipulation [53] (refer to 2.1).

The objective of developing Kit was to facilitate data-manipulation activities for the audience of non programmers. Throughout this study, a prototype of Kit was used to evaluate the environment's ability to support its target audience. The prototype included many fully-functioning blocks, yet the output visualizations such as the maps and the coloration blocks were not complete at the time of the study. This exploratory study consisted of four different phases, assessing different aspects of non- programmers' ability to learn and use the environment. These evaluation activities included the engagement of 21 different participants who were categorized into three different groups (see Section 3.5 for more specifications about participants). The diversity of the participants' backgrounds was an important issue to give a better understanding on how different people might have different approaches, needs, and limitations when solving the same problems. Consequently, lots of useful insights and recommendations will contribute towards enhancing the design of Kit to cover a larger scope of users' needs.

This study had three objectives:

1. Assess non-programmers ability to provide flow-based models to some given problem scenarios. The assessment will provide useful insights to understand potential users' needs and to suggest further design recommendations.

2. Assess the usability of the prototype of Kit when it is used by non-programmers.

3. Assess how capable the prototype of Kit is to handle inquiries similar to those in real-life, and how recognizable the generated models are. Kit should provide a comprehensible presentation for the solution models to minimize users mental effort when referring to the model later for reusing or editing.

The following sub-sections demonstrate the different phases of this research which are: Introduction Phase, Design Phase, Recognition Phase, and Interaction Phase.

3.1 Introduction Phase: Introducing Kit and Measuring Impressions

In this phase, the main research question was: *How easy is it for non-programmers to understand the flow-based paradigm used in Kit?*
The objectives of this phase were to:

1. introduce the flow-based paradigm of Kit to the participants by elaborating some trivial examples

2. assess users ability to replicate the presented examples

3. survey the users to evaluate their initial impressions of Kit.

The Introduction Phase included three focus group sessions, one for each participating group. Each session was completed in about 15-30 minutes.

At the beginning of the session, participants were introduced to the project's website:

`http://programmingfortherestofus.com/`

The website includes Kit's IDE (Integrated Development Environment), along with some related topics and examples to support people who might be interested in the "Programming for the rest of us" topic.

Participants were introduced to the environment by demonstrating the help examples provided by Kit. The used examples were previously implemented and integrated as a part of Kit's environment. They can be found by selecting the following sequence

of choices from the menu: **Help**, then **Trivial Application**.

During the sessions, participants were encouraged to ask questions or share concerns whenever they had any. Once blocks of examples were explained, participants were asked to implement these examples again using Kit environment indirectly. They were encouraged to navigate the menu and predict the correct selections to replicate the presented models (examples). This procedure was done by me using a working prototype of Kit[1], following their suggestions to implement the code on my laptop that was connected to a projector.

The demonstrated programs are described below:

"Hello World" Example:

The first example was the the traditional "Hello world" program, which is illustrated in Figure 3.1.

During this example, the first step was to show participants how to create an input's textfield by choosing the following series of selections: **Inputs** then **Simple**. Using the same logic, participants should be able to create the output text field using the following sequence: **Outputs** then **Result**. The last step was to connect the input block by dragging a pipe that creates a data flow connection between input and output blocks.

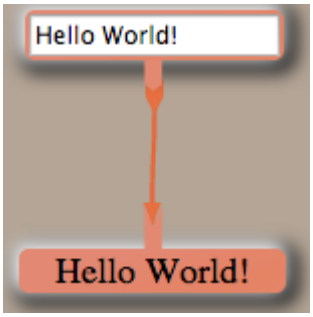


Figure 3.1: Introduction Phase: Hello World Program

[1] All the presented examples were fully-developed in the version of Kit used in this phase.

"Graphing" Example:

The second example was plotting a data table on a graph, which is illustrated in Figure 3.2[2].

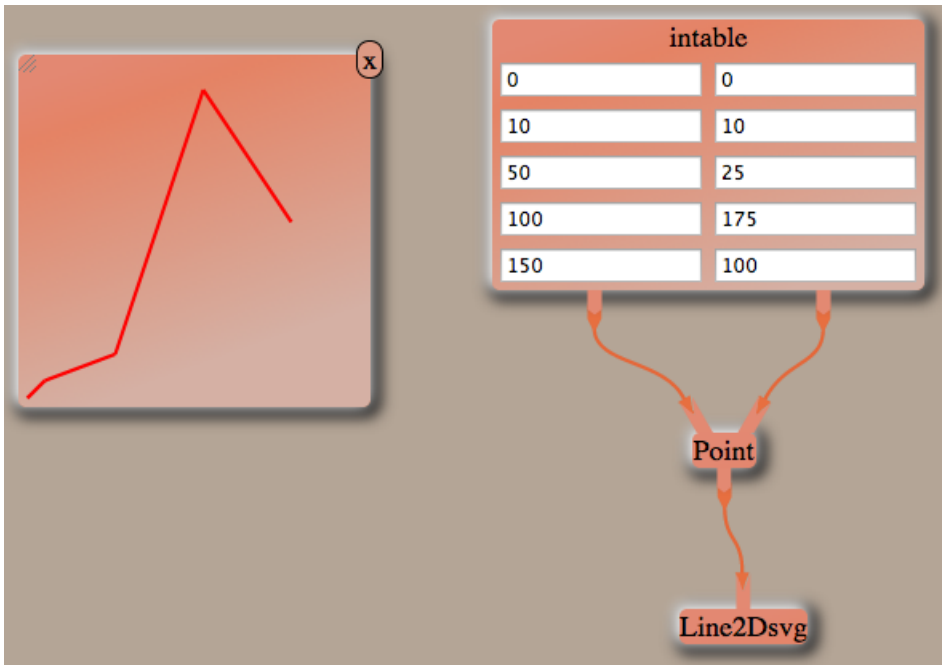


Figure 3.2: Introduction Phase: Graphing Program

During this example, participants were asked to create the correct sequence to create the upper part (Input) of the program which was: **Inputs** then **InTable**. The created table was always initialized with a single row and a single column. To modify the size of the table, it should be clicked to have a popped up sub-menu that shows the possible actions to be performed on this table. Accordingly, the user had to select: **Add Row**, then **Add Column** options as many times as required to obtain the correct table size without restricting the order of these two selections. At this point, the table was created, and each point contained two different values representing the x-axis and the y-axis respectively. The next step was to create an entity to hold these two values in a single point. That was performed following this sequence: **Data**, then **Point**. The last step, was to create the correct output layout by following this sequence: **Output**, then **Line**. After

[2]Kit is still under development, which explains why the graph is missing labels and legends. These features are to be added soon.

blocks were created, they were connected by using pipes to start the data flow.

"Calculation into table" Example:

The third example was to create a program that performs calculations into tables. The example is illustrated in figure 3.3. At this point, I am assuming that participants would able to create this input table easily based on their experience with the previous example. To create the addition block, they were required to click: **Math**, then choose **"+"**. The **OutTable** had been created in the previous example as well, so it should be relatively easy for them to make the correct sequence of choices to build the required output block. After dragging the pipes to connect the blocks, the example was completely done.

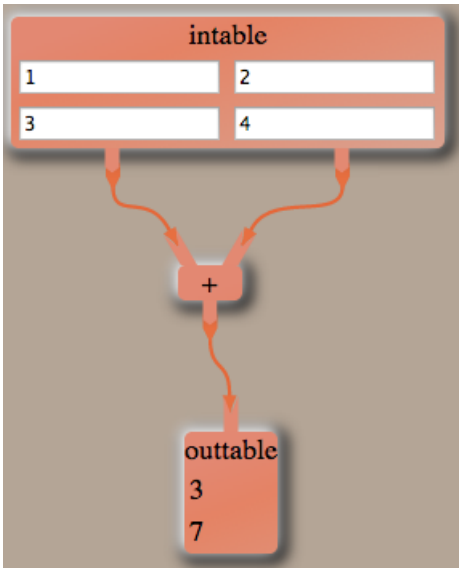


Figure 3.3: Introduction Phase: Table Calculation Program

"Take the first part of a sequence" Example:

The last and the most complicated example is the iteration program as shown in Figure 3.4.

The hardest part of this example was to introduce the "loop" concept to naïve programmers. The term "loop" was replaced with **Iterator**, and this block can be found under the **Input** submenu. The **Iterator** block is always initialized with three inTabs. The first inTab receives the start value of the sequence, the last inTab receives the last value of the sequence, whereas the middle inTab receives the increment base of the sequence. Each input of these inTabs is a **simple Input**,

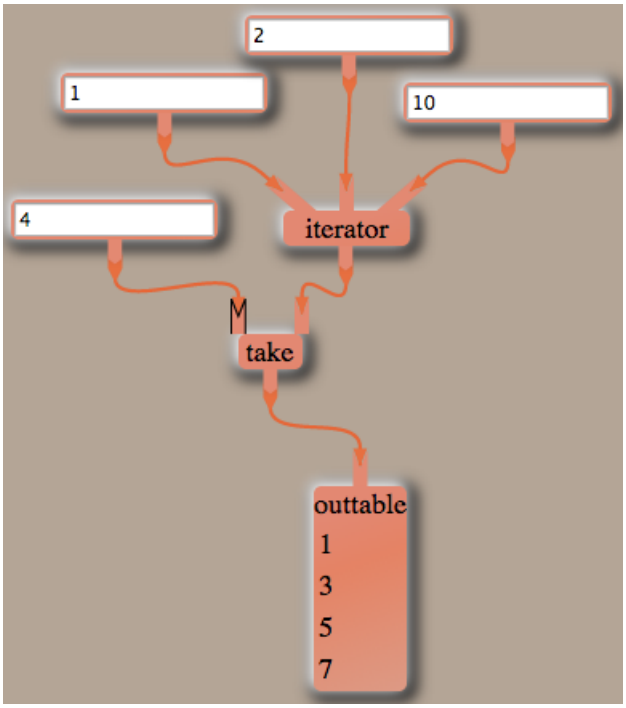


Figure 3.4: Introduction Phase: Iteration Program

which is a single numeric value. Once the sequence was generated, the participants were asked to take the first four values of the sequence only. To obtain that subsequence, the following steps needed to be performed: **Control**, then **Take**. When **take** was created, users should be able to notice that **Take** block has two inTabs by default, one of them is framed in black, which indicates that this is a control tab that controls the number of values to be printed out as an output. control tabs are user-defined values to constraint some processing of the corresponding block. The last part will be the **OutTable**, which was created in the same way as the previous two examples.

At the end of this session, participants were given an initial survey (can be found in Appendix B) to measure participants' initial impression about Kit's usability after the orientation phase. The survey consisted of two main parts. The first part collected participants' general information[3], such as:

- Age

[3]without being linked to the names of the participants

- Gender
- Level of Education
- Programming Knowledge Level (They were not given a specific definition of programming, it was up to their understanding of programming knowledge in general.)

The second part collected participants' initial impressions for the following usability's measurements:

- Kit's Understandability
- Kit's Learnability
- Kit's Effectiveness
- Their interest in having a new data-manipulation tool that is easy and efficient
- Their interest in learning how to use Kit
- Difficulties encountered for lacking programming knowledge.

3.2 Design Phase: Collect Design Requirements by Problem Scenarios

In this phase, the main research question was: *How can Kit facilitate processing real data inquiries for non-programmers?*

The objectives of this phase were:

1. to assess non-programmers ability to provide solution models for problem scenarios
2. detect some of non-programmers needs and limitations (whether these needs were related to the programming environment or associated to the users' approach of solving problems).
3. to provide some design recommendation to accommodate these needs.
4. to suggest more functionalities to be added to make Kit capable of handling complicated inquiries.

During each design session[4], participants were given four different problem scenarios (which are described later in this section), and they were asked to suggest some solution models to those scenarios using the flow-based paradigm in the 60 minutes session. Participants were asked to share ideas, recommendations, and suggestions regarding data analysis and manipulation. Creating the solution model would require finding answers to some data-analysis related question, such as:

- What data fields might be required to perform such analysis?
- What data source might be used to find the required data?
- How can data be reshaped (by using functions similar to filter and merge)?
- How can data be analyzed/processed (by suggesting correlations and calculations)?
- How can the output be visualized to achieve the best outcome?

Collecting such information will help to gain better understanding of how non-programmers approach data and what obstacles they encounter. These findings will hopefully provide some general guidelines to improve the design of Kit, to make it more effective for the target audience.

The scenarios (refer to Section 2.2.2.2 to read more about scenarios and personas) are described in the following subsections:

3.2.1 Problem Scenarios

The problem scenarios were designed to be presented in the main discussion sessions for participants to encourage their creativity and come up with suggested solutions. Each scenario consisted of two parts to emulate real world problems (More useful readings about personas can be found in Carroll's work of "scenario-based designs"[17]):

Persona

A persona is a person who adds emotional weight to the scenario and adds motivation for the participants to think in the right direction. Persona encourages participants to live the scenario as if the problem is theirs. This encourages realistic and optimized solution models to accommodate the persona presented in the scenario.

[4]Each participating group has its own design session.

Task

The task will be a general open-ended problem that can be solved in various ways to encourage participants to think creatively in many different directions. Meanwhile, the task should be specific and clear to the audience. It should be more like regular situations that might be encountered by the persona.

The importance of creating problem scenarios was to explore the different approaches of participants to solve the problems, and to detect the design issues to be fixed. The key while discussing the solutions to the problem scenarios was not to provide the participants with any specific features provided by Kit. Ideas and strategies should be freely presented without any design limitations to maintain the flexibility of their solutions.
Based on the previous guidelines of creating useful problem scenarios, four scenarios were designed and presented in the discussion session with each different group:

Nutritionist Problem Scenario

"Emily is a Nutrition student. She is working on a study to find the link between the obesity rate in different countries with the following factors: average household income, average educational level, and ethnicity".

Real-estate Agent Scenario

"Jacob is a real-estate agent. He wants to study the most profitable region in a given city, to invest in a property. He must study the history of sale price and other factors".

Teacher Scenario

"Lily is a teacher who wants to compare students' performance in two different semesters to compare their progress in three different participants".

Traffic Engineer Scenario "William is a traffic engineer who works on road congestion. William is trying to find out the relationship between each congested intersection and the count of each different type of vehicles (i.e cars, buses, trucks, etc.) during different periods of time".

3.3 Recognition Phase: Matching Kit Models with Scenarios

The main research question here was: *How recognizable/readable are the generated Kit models?* The main objective of this phase was to assess the created model's readability.

The programmer should be able to understand any previously created code to facilitate an easy reuse of the code [36]. Users do not like spending too much time and mental effort trying to understand what each part of the code does. Whenever an edit is required, minimal effort of code recognition should be required.
This phase consisted of introductory part at the beginning of the session, followed by a multiple choice test. Total of 15 participants were involved in a 30-40 minutes session. This phase was done about four months after the discussion sessions were completed as Kit environment was not fully evolved at the time of having the first two phases. Considering this long duration of development, it was hard to predict participants' ability to remember their suggested models. The challenge of designing the test was to create some similar but different problem scenarios to match the same built model.

The introductory model is shown in figure 3.5. Out of the four suggested solution models (can be found in Section 4.2), the real-estate agent scenario was picked as it had the most number of blocks including these that were newly suggested. The model was modified to cover larger scope of blocks.

Following are the models used in the Recognition Phase:

Introductory Model

Compared to the original suggested model in section 4.2, I added the radio buttons to explain how they work. I have also added a blue coloured dot to be placed on the map to indicate the average income.
After the introductory example was discussed, participants were encouraged to ask questions or share concerns. Lastly, participants were asked to take the multiple choice recognition test (found in Appendix E). They were provided with a "Kit Manual" (found in Appendix C) to help them to understand the functionality of the blocks.
The test consisted of three Kit models with three problem scenarios assigned to each model. For each model, we provided a correct answer, a close answer, and a wrong answer. The participants were asked to pick what they think is the correct answer for the provided Kit model.

The Nutritionist Model

The first model is the shown in Figure 3.6, and participants were given the following question:

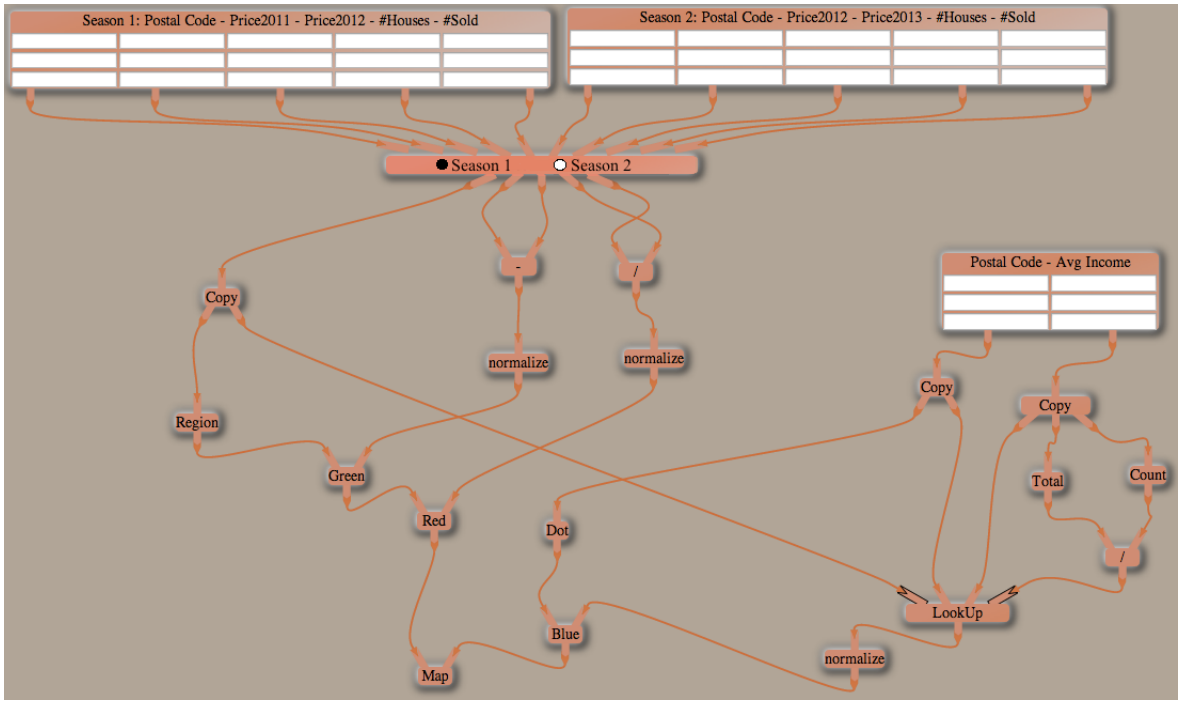


Figure 3.5: Recognition Phase: An Introductory Example Representing the Real-Estate Problem Scenario

> Emily is a Nutrition student. She is working on a study to find the link between the obesity rate in different countries with the following factors: GDP, average educational level, and ethnicity.
> In your opinion, which one of the following scenarios matches that diagram:
>
> - Scenario 1: She decides to plot lines showing the ratio of each of the factors divided by obesity level on a graph by country.
> - Scenario 2: She decides to plot lines showing the obesity level and each of the factors on a graph by country.
> - Scenario 3: She decides to produce a scatter-plot with obesity by country, with each datapoint coloured by the combination of factors.

The green scenario is the correct selection, The orange scenario is the close selection, and the red scenario is the wrong selection.
Among the previous scenarios, participants were expected to exclude the first scenario as it was the obvious incorrect option since the model did not include any division operator. Next, the subject needed to choose one of the remaining two options. The trick

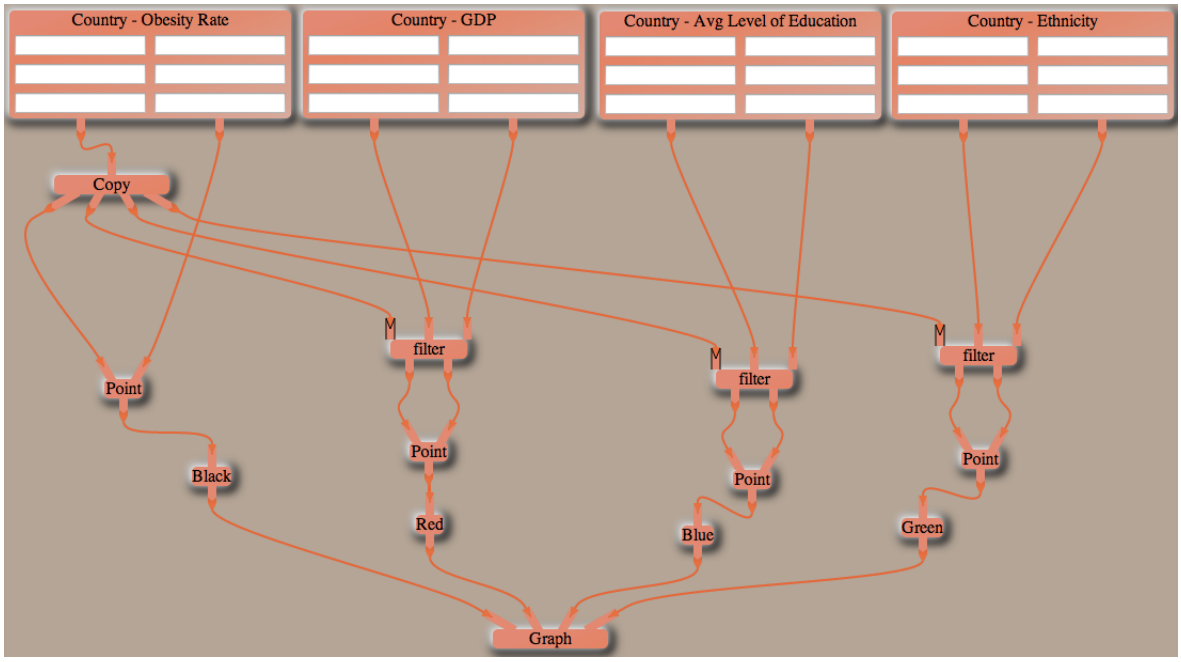


Figure 3.6: Recognition Phase: The Test's First Model

here was the *"the combination of factors"* phrase used in the third scenario. The shown model plots each factor individually, which makes this phrase incorrect.

The Teacher Model

The second model is shown in Figure 3.7, and participants were given the following question:

> Lily is a teacher who wants to compare the students' performance in two different semesters to compare their progress in three different subjects.
> In your opinion, which one of the following scenarios matches that diagram:
>
> - Scenario 1: She decides to plot the average for all students by subject, for each term.
> - Scenario 2: She decides to produce a scatter-plot with average grade in first term by average grade in second term, per student.
> - Scenario 3: She decides to plot the differences between the two semesters by student, showing the differences in each subject and the average of the three.

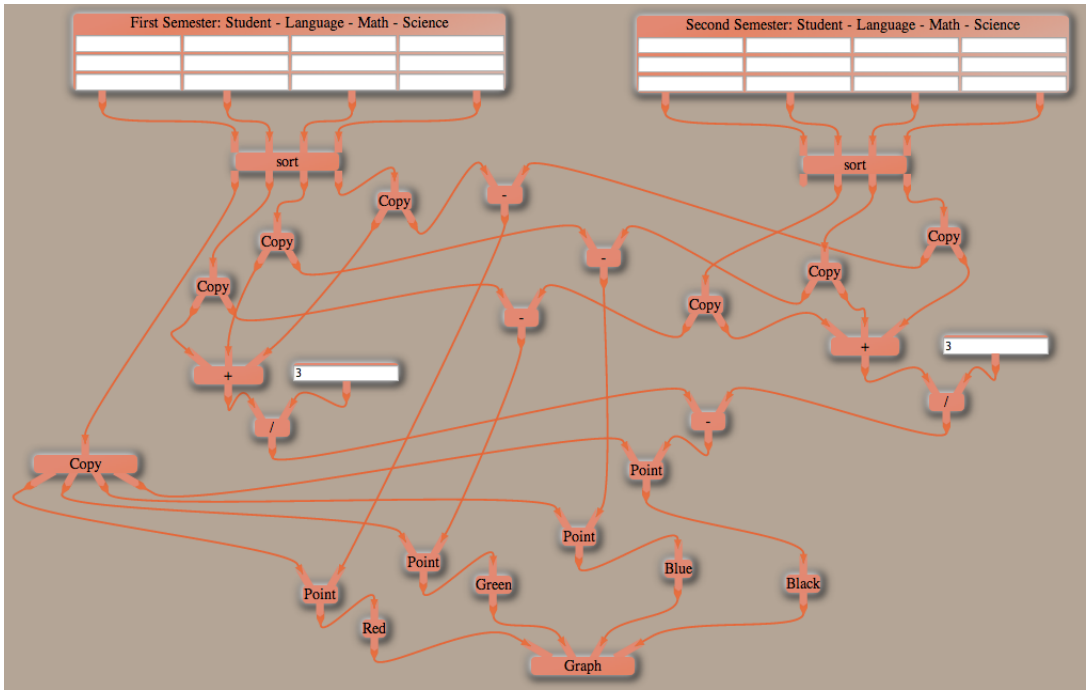


Figure 3.7: Recognition Phase: The Test's Second Model

The green scenario is the correct selection, The orange scenario is the close selection, and the red scenario is the wrong selection.
The confusing answers in this model are the first and the third scenarios. The difference here was obvious because the first scenario lacked lots of the difference values calculated in the model. Furthermore, the model is producing one graph not two as indicated in the first scenario.

The Traffic Engineer Model

The third and last model is shown in Figure 3.8, and participants were given the following question:

> William is a traffic engineer who works on road congestion. William is trying to find out the relationship between each congested intersection and different types of vehicles (i.e cars, buses, trucks, etc.) during different periods of time.
> In your opinion, which one of the following scenarios matches that diagram:

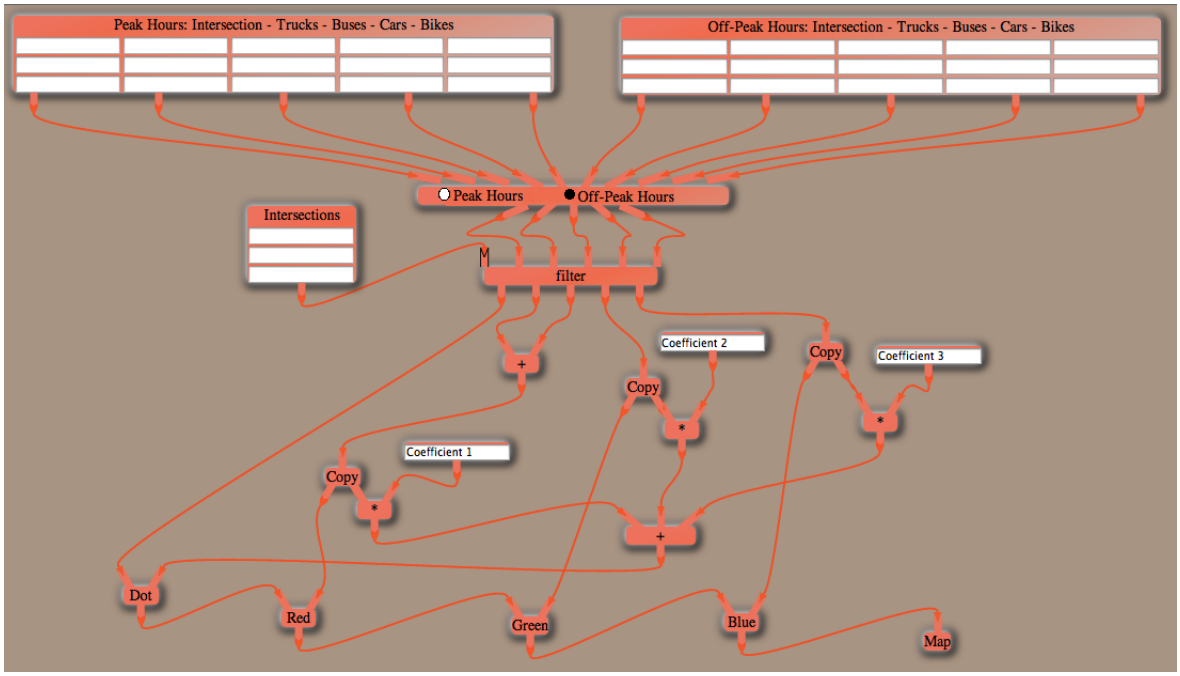


Figure 3.8: Recognition Phase: The Test's Third Model

- Scenario 1: He decides to put a toggle so he can switch between the peak-hours/non-peak-hours data, and for each graph plot a dot on a map for each intersection that is sized by the weighted traffic volume, and coloured by the truck+bus / cars / bikes ratio.
- Scenario 2: He decides to put a toggle so he can switch between the peak-hours/non-peak-hours data, and for each graph plot a dot on a map for each intersection that is sized by the weighted traffic volume, and coloured by the truck+bus / cars / bikes ratio. He also plots an extra point that is sized by the overall traffic weight.
- Scenario 3: He decides to plot a graph for each of peak-hours and non-peak-hours data, and for each graph plot a dot on a map for each intersection that is sized by the weighted traffic volume, and coloured by the truck+bus / cars / bikes ratio.

The green scenario is the correct selection, The orange scenario is the close selection, and the red scenario is the wrong selection.
Among the previous scenarios, participants were expected to exclude the third scenario because the model produces one map only. Next, the subject needed to choose one of

the remaining two options. The trick here was the "He also plots an extra point that is sized by the overall traffic weight" sentence used in the third scenario. The plot had one dot only, and no extra dot was created.

3.4 Interaction Phase: Hands-On Kit

The research question of this phase was: *How easy is it to use Kit interactively to perform a specific task?*
The objectives of this phase were to assess non-programmers ability to interact with Kit by observing them while completing list of tasks, and detect problems encountered by these users. In the interaction test, I asked a subgroup (4 participants) of the participants to complete some given tasks using Kit. Four participants from the GP group completed this test[5].

Participants were tested individually, and each session duration was 45-50 minutes. Participants were asked to complete the following tasks while being observed:

1. The first problem scenario:
   You have a 4*4 input stream. You need to calculate the sum of the first three values of each row, and the average of the third and fourth value. The results from both calculations will be output in a 2*4 output stream.

2. The second problem scenario:
   John is a sale manager who works in a department store. He wants to investigate the sales' performance throughout the past five years. Assuming that he has the sales history table for the last 5 years that includes: Year, Dept1, Dept2,Dept3, and Dept4 how would he retrieve the following information:

   (a) Avg. of sales for all the departments each year.

   (b) Avg. of sales for the last three years for each department.

   (c) Output the last year's sales only.

For the first problem, participants were asked to create a code similar to the one shown in Figure 3.9 , and they were expected to implement it in the following order:

1. Create a table and fill it with arbitrary values.

[5]Participants of this phase were selected based on their availability.

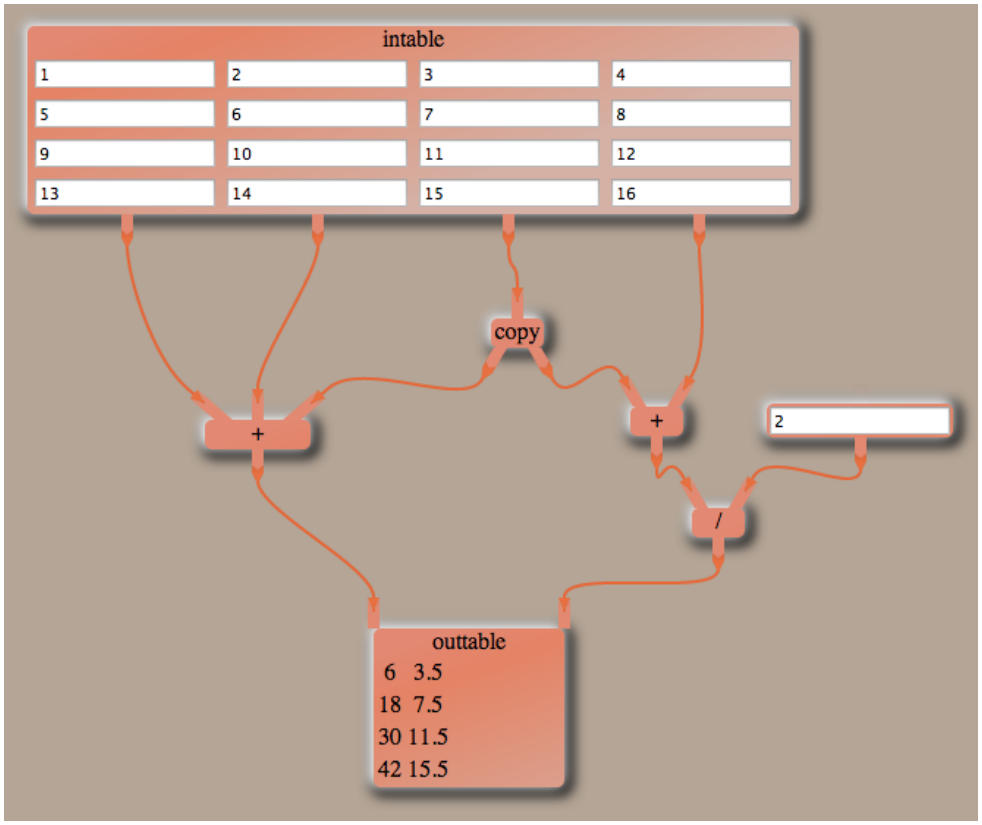


Figure 3.9: Testing Phase: First Problem's Implementation

2. Create a "+" block and add an inTab to it to add the first three values.
3. Create another "+" block to add the last two values. A "copy" block is needed to duplicate the third value since it is used twice.
4. Create a "/" block to calculate the average and link the first inTab with the output of the previous step.
5. Create a simple input box and fill it with 2, then link it to the second inTab of the "/" block.
6. Create an output table and add a column to it.
7. Finally, link the outputs of the second step and the fifth step to the output table.

For the second problem scenario, the users were asked to create a table of sales for four departments for the last five years. Hence, a 5 by 5 table needed to be created. The

following steps needed to be performed in order to get the correct model of the first part which is shown in Figure 3.10

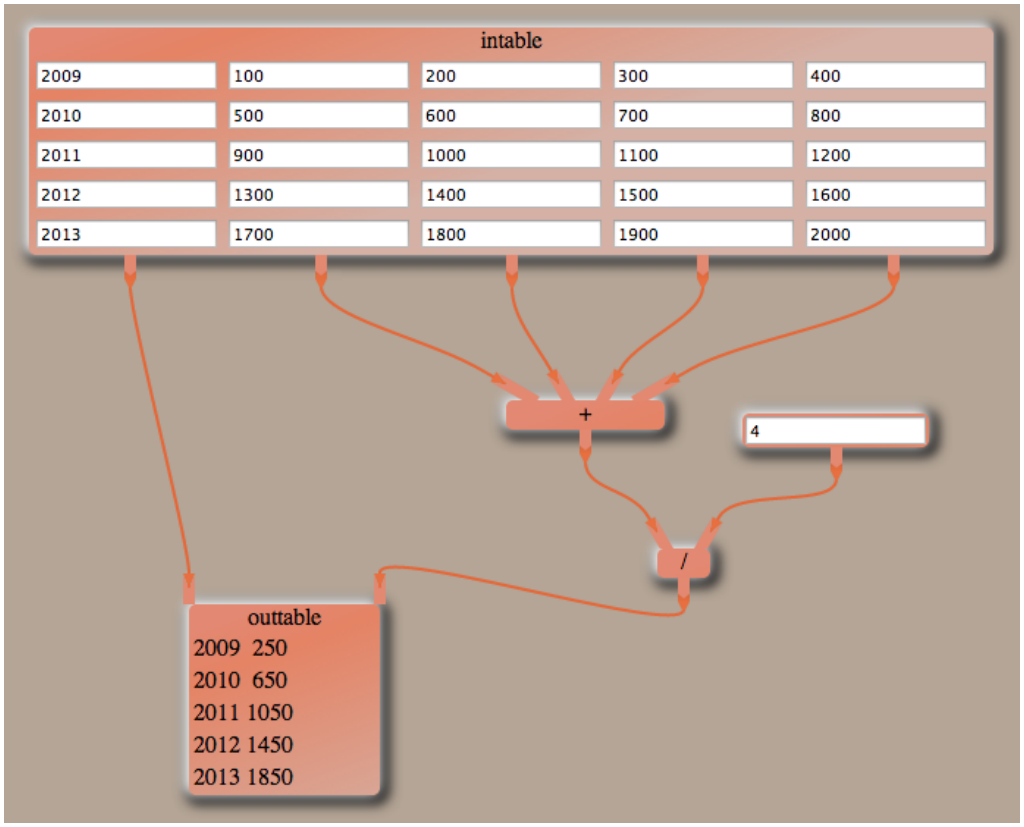


Figure 3.10: Testing Phase: Second Problem's Implementation - Part (a)

1. Create a "+" block with four inTabs to sum up all the departments' sales.
2. Create a "/" block, and link the first inTab to the output of the first step.
3. Create a simple output and fill it with 4, and link it to the second inTab of the "/" block.
4. Create an output table with two columns and link the inTabs with the year's tab, and the output of the 3third step.

To complete the model of the second part in the second problem (shown in Figure 3.11) correctly, participants needed to follow these steps:

1. Create a **Filter** block with 6 inTabs in order to get the last three years only. The first inTab is the control tab that includes the filter table, the second inTab is

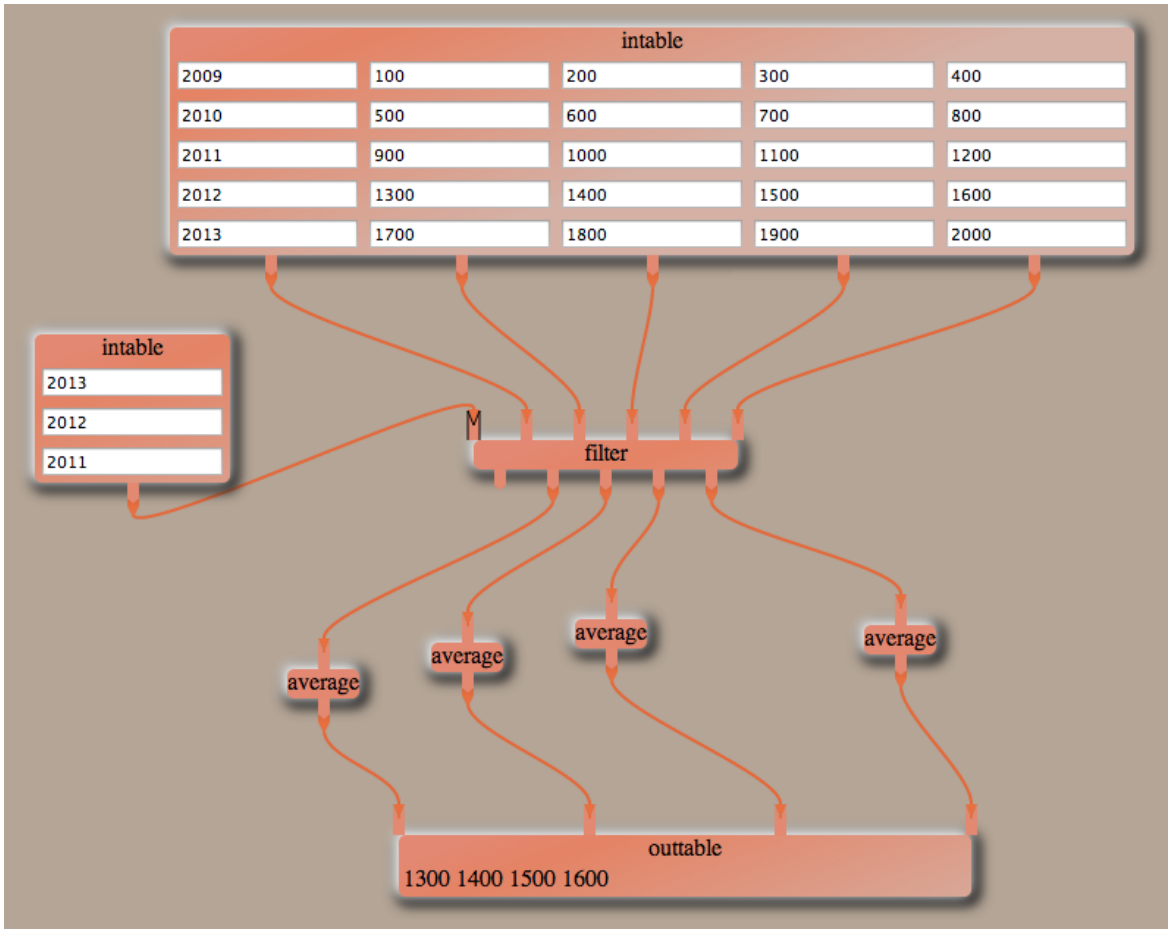


Figure 3.11: Testing Phase: Second Problem's Implementation - Part (b)

linked to the year's pipe, and the last four inTabs will be linked to the departments' sales.

2. Create a filter table, and fill it with the required years.

3. Create four **Average** blocks and link each one with the corresponding departments's pipe. This average calculates the average of the column's values, while the average in the previous example calculates the average per row.

4. Create an output table with four inTabs, and link each one with the outTab of every **Average** block.

The last model for the second problem is shown in Figure 3.12, and can be implemented by following these steps:

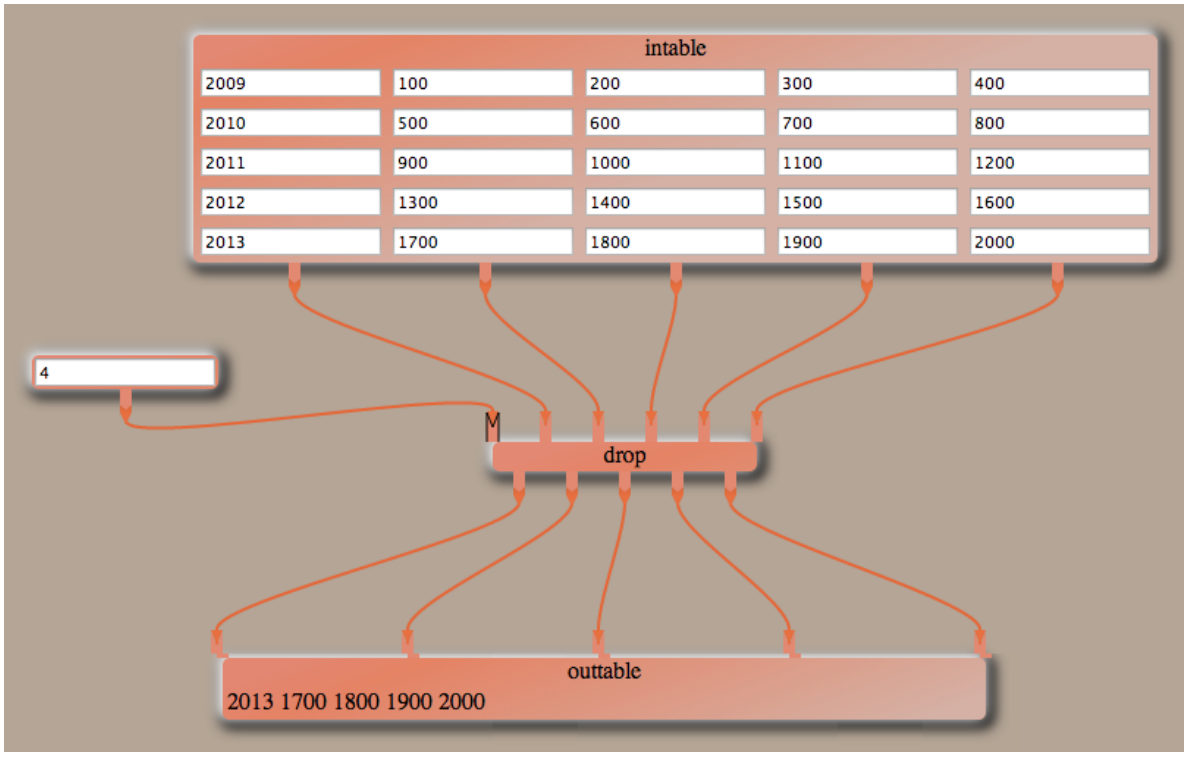


Figure 3.12: Testing Phase: Second Problem's Implementation - Part (c)

1. As the table is already sorted by year, the last year is going to be located at the last row. Participants can use the **Drop** block to get rid of the first four tuples, by indicating feeding the value 4 in the **Drop** block. To do so, participants need to create **Drop** with six inTabs. The first inTab is a simple input with the value 4. The second inTab is linked to the year attribute. The rest are linked to the departments' attributes respectively.

2. Create an output table with 5 inTabs and link it to the outputs of the first step.

3.5 Participants

Based on the golden standards of Gould and Lewis [31], system's potential users need to be involved in the early stages of designing a system. Through the different stages of this research, a total of 21 participants were recruited to participate in the different phases of the study. Participants were of different genders, careers, ages, and levels of education. The participants were categorized into three main groups, and they were asked to participate in the different phases of the study. The objective of having a wide

range of participants was to observe different ways of approaching problems. The diversity of the participants exposed more strategies and methodologies naïve programmers may use when they analyze data.
Throughout the sessions, volunteers were free to participate in discussions and ask questions. All participants were asked to sign a consent form that contained a formal description of their participation (see Appendix A).
Following are the three groups of participants:

3.5.1 University Planning Office

The first participating group is the University Planning Office (UPO). The main goal of having this group was their experience of manipulating data on a daily basis. They are professional data manipulators who are not necessarily programmers. Based on the UPO website [91]:

> "Every year, UPO manages the development and implementation of activities related to the University's academic plan, enrolment planning, and academic budget planning. The UPO also undertakes policy research, data analysis and institutional research project oversight for the university."

Having the domain experts of data-manipulation field is very important since they could provide good solution models based on their analysis skills. Furthermore, they could come up with more specific requirements and recommendations either for blocks to be added or general features to be considered. We successfully recruited seven UPO participants, who were four females and three males. There were three participants in the 36-45 age group, two in the 46-55 group, one in the 25-35 group and one in the >65 group. Two participants stated that they had little programming knowledge, four had programming knowledge and one had no programming knowledge at all.

Participants who picked either "Little" or "Yes" as an answer for their programming knowledge were asked to list the language(s) they know. These are the results:

- One participant indicated that he has a little programming experience with SPSS[6] language.

[6]SPSS Statistics is an IBM software package that is used for statistical and analytical purposes [88].

- Another participant indicated that she knows: MATLAB [7], Stata [8], and VBA[9].
- A third participant indicated that she knows: HTML[10], JavaScript[11], and SQL[12].

The rest of the Participants either didn't have any previous knowledge or didn't specify it.

3.5.2 Journalism Graduate Students

Participants from this group were graduate students in the Journalism Masters program of Ryerson University. Journalists were another valuable participating group as they might have some analytical skills. Looking at newspapers or watching television reports that are filled with statistics and visualizations makes it clear how data is an important part of the media. In order to come up with useful insights, journalists need to investigate, manipulate and analyze the data sets related to the story of interest. They have to understand patterns and study attributes carefully. Every successful journalist should be able to filter and visualize his or her insights to be able to present useful information to the public out of raw data. Figure 3.13 shows the Data-Driven Process for journalists which describes the steps of processing data to come up with the story's insights.

The Journalism Graduate Students (JGS) group consisted of five females and a male. They were all in the category (25-35) years old which may give an indication that they have not yet been extensively practicing their knowledge in the real world. Their level of education is the same since they are all graduate students. Most of them did not have any programming experience except one female who specified a little knowledge in: HTML, JavaScript, and CSS[13].

[7] MATLAB is a high-level language and interactive environment for numerical computation, visualization, and programming [55].

[8] Stata is a data analysis program. It has a command driven environment with the option of having dialog boxes to generate the commands for new users [1, p. 3].

[9] Visual Basic for Applications (VBA) is a part of the Visual Basic development tools. It provides language support and an interface for forms, controls, modules, and data-access technologies [48].

[10] Hyper Text Markup Language (HTML) is the foundation of all content appearing on the World Wide Web (WWW). It is a set of instructions to describe how to display a content more than being an actual programming language [14, p. 1].

[11] JavaScript (JS) is a scripting language that enables enhancing static web application by providing dynamic, personalized, and interactive content [73, introduction].

[12] Structured Query Language (SQL) is a special-purpose programming language that was designed to manage data held in a relational database management system (RDBMS)[45].

[13] Cascading Style Sheets (CSS) is a language designed for describing the appearance of documents written in a markup language such as:HTML[76].

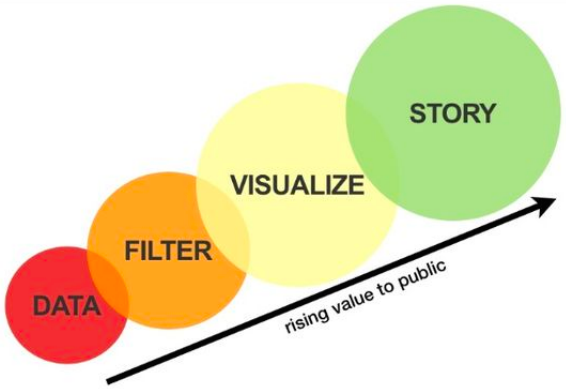


Figure 3.13: The data-driven journalism process (From [51])

3.5.3 General Population

The final group is the General Population (GP). Any individual can fit the criteria of this group except those who are professional programmers. This group had the most diversity among its individuals. It consisted of three males and five females, and they were in four different age categories. Three Participants were under 25 years old, three were (26-35) years old, and two were (36-45) years old. Three of them had some college/associate degree, while four of them had a Bachelor's degree, and the last one had a Graduate degree. Half of them didn't have any programming experience or knowledge, whereas the other half had a little knowledge of programming concepts. The importance of this population is that it covers different classes of society, which meant introducing different approaches to solve our problem scenarios.

Chapter 4

Results

This chapter presents the results of the previously discussed phases of the study. The results are illustrated in four sections. The first section discusses the results of the Introduction Phase and illustrates participants' evaluation of Kit. The second section describes the results of the Design Phase, where participants were asked to suggest solution models for some given problem scenarios. The third section describes the results of the Recognition Phase, where participants were asked to match Kit models with its correct problem scenario. The last section describes the results of the Interaction Phase, where participants were asked to implement some given tasks using Kit interactively.

4.1 Introduction Phase Results

In this phase, the main research question was: *How easy is it for non-programmers ti understand the flow-based paradigm used in Kit*. As mentioned before a prototype of Kit was introduced to participants by using previously implemented trivial examples that were provided by Kit environment (described in Section 3.1). Each example was illustrated to participants to introduce them to the flow-based paradigm. Once the examples were demonstrated, participants were asked to replicate them by trying to figure out the a correct sequence of actions using Kit's menu. Participants were not using Kit directly. Instead, I was using Kit in a projector and they were suggesting options to be clicked from the menu. By the end of the session, participants were given a survey (see Appendix B) to evaluate some of Kit's usability aspects based on their

initial impressions.
The objectives of Introduction phase were:

1. introduce the flow-based paradigm of Kit to the participants by elaborating some trivial examples.

2. assess users ability to replicate the presented examples.

3. survey the users to evaluate their initial impressions of Kit.

The following subsections illustrate how each focus group reacted to Kit's examples. Results are illustrated using "Box and whiskers" plot which is a graphical device that can visually present the range, the in quartile range, the median, the lowest (minimum) score, the highest (maximum) score. In a box plot, maximum and minimum values are represented by upper and lower whiskers respectively. The data range is divided into four quartiles. The first quartile starts from the lower whisker and ends at the beginning of the second quartile. the second and the third quartiles are indicated in the legend of the graph, the fourth quartile starts at the end of the third quartile and ends at the lower whisker. The median is the centre value and it is indicated by the value that separates the second and the third quartiles. Box plots provide a visual way to examine the centre, the variation, and the shape of distribution of interval ratio variables[18].

4.1.1 Focus Group Sessions

4.1.1.1 UPO's Usability Survey

The UPO group has the highest level of data-manipulation experience as they manipulate large amount of data on a daily basis (more details about UPO can be found in 3.5.1). Having domain experts in this research was very important because they would be able to analyze problem scenarios and create a good solution models based on their experience. Furthermore, domain experts would have good capabilities to highlight some usability issues or provide useful recommendations to enhance Kit's design. Some issues were addressed when participants asked questions like: "Would Kit be able to import tables from HTML pages? and how would the headers be saved?" or "Is the tool compatible with any browser?" such question can address some important aspects to be considered by the team of Kit.

By the end of the Introduction Phase, participants were surveyed to evaluate their impressions after being introduced to Kit. Figure 4.1 shows a box plot that represents participants' suggested scores. As can be seen, *importance* had the best evaluation among the other measurements, while *effectiveness* scored the lowest.

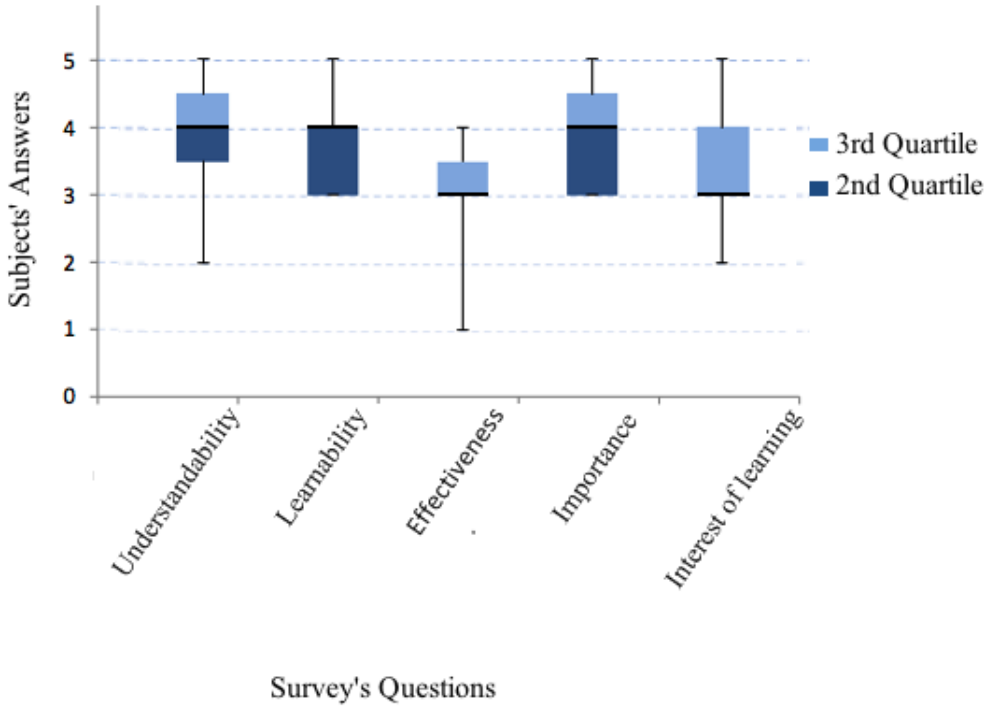


Figure 4.1: Design Phase: UPO Survey Results
(1 = Least and 5 = Most)

4.1.1.2 JGS's Usability Survey

The second group of participants was the JGS. This group was limited to a specific age group and a single level of education (more details about this group can be found in Section 3.5.2). This group had a combined session that included introduction and design phases together on the same day (the duration of this part was about 15 minutes). Hence, the usability survey was answered based on their impressions after the completion of both phases.

During the introduction session, participants were able to understand the presented examples. However, they showed some hesitancy and lacked confidence when they

tried to rebuild the illustrated examples. They were able, after making several mistakes, to figure out a valid sequence to replicate the shown examples. When they were asked for any question or concerns, they did not share any.

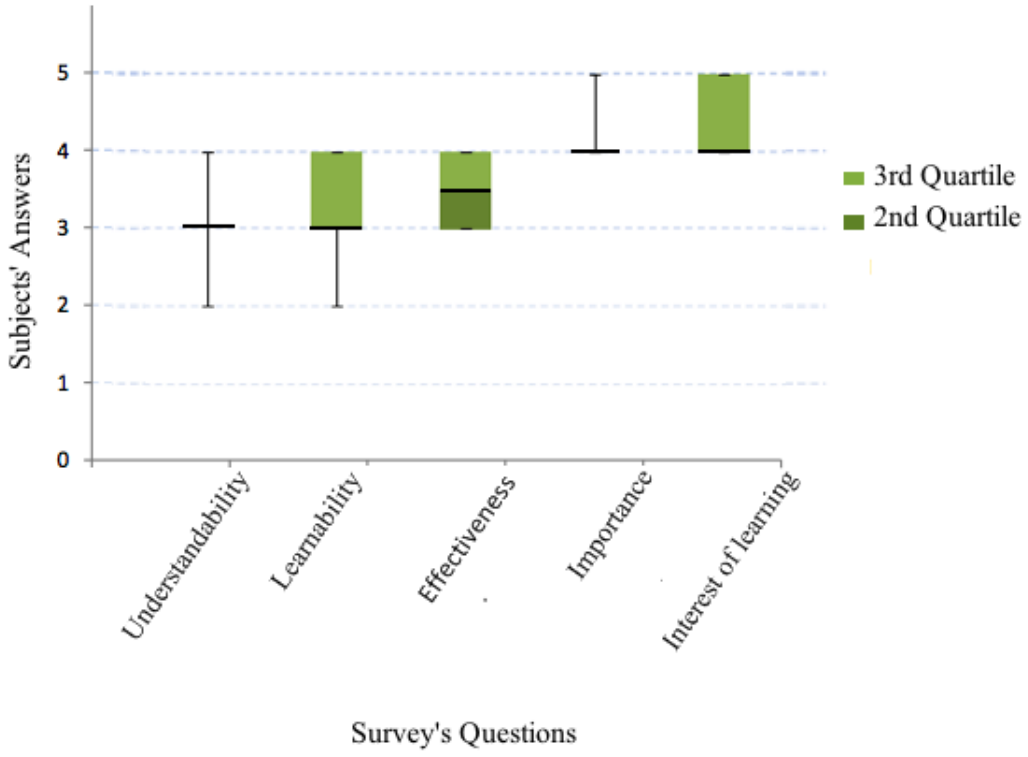


Figure 4.2: Design Phase: JGS Survey Results

(1 = Least and 5 = Most)

As can be seen in the Box plot in Figure 4.2, the lowest minimum value of 2 was assigned to both *understandability* and *learnability* measurements. These scores are the least for these two measurement among the participating groups. However, *Interest of learning* had the highest scores as all the value ranged between 4 and 5, with half of these values to be greater than 4. Four out of six participants in this groups agreed that lack of programming experience affected their ability to comprehend the examples negatively. Keeping in mind that this session was combined with the "Design" session, their scores can be derived based on both sessions' contents rather than the introductory examples only. Aside from the numbers, three participants added the following comments:

- JGS1: "Came late! Wish I could offer more comments.".

- JGS2: "I feel that my lack of math/data analysis knowledge made me a weaker test's [sic] subject. Perhaps someone who understands data management better would be a better test's [sic] subject. Very interesting though!"
- JGS3: "As journalists we look for basic/simple numbers. I find the data created by Kit sometimes makes it overly complicated."

From these comments, I drew two conclusions:

1. Despite participants' ability to interpret the simple examples (help examples are shown in Section 3.1), they were overwhelmed with the real-life examples. When participants from this group were introduced to the real-life problem scenarios in the same session, they struggled to understand the data analysis rather than how to implement this analysis. This struggle influenced their initial impressions of Kit's usability. Their lack of data-manipulation experience, which has been shown in their comments, affected their ability to comprehend the solution models suggested in the design session (refer to Section 4.2). That might explain the high percentage of 67% of participants who answered with a "Yes" for the survey's sixth question that asks about the significance of programming knowledge. Despite their struggle, this group had high scores for the third question which measures *effectiveness*. This means that they believed in Kit's ability to handle the real-life problems. They showed high level of interest to learn Kit as can be seen in the fifth question's scores.

2. JGS1 had the lowest evaluation's values for the "Understandability" and the "effectiveness" measurements, and the second lowest evaluation for the "Learnability" measurement. The participant's late arrival might be the reason with his low scores. He missed more than half of the session, and that would easily affect his ability to give some reliable scores when evaluating the tool. Unfortunately, this assumption can not be confirmed as this participant was not a part of the next phases, but it seems a reasonable conclusion.

4.1.1.3 GP's Usability Survey

The third group of participants was the GP. For more details about this group refer to Section 3.5.3. The duration of this session was about 15 minutes. It was impressive how participants from this group, who had no to little programming knowledge and did not

necessarily have any data manipulation experience, reacted to Kit with lots of enthusiasm. Participants from this group were able to comprehend the presented examples, and they were able to suggest most of the correct sequences of actions to replicate those examples. However, they showed limited knowledge of some programming terms such as input, output, and iterator. These terms were explained to participants to give them a better understanding of the corresponding blocks in Kit. When participants were asked to share comments or concerns, they had none.

Figure 4.3 shows relatively similar scores for the different measured aspects.

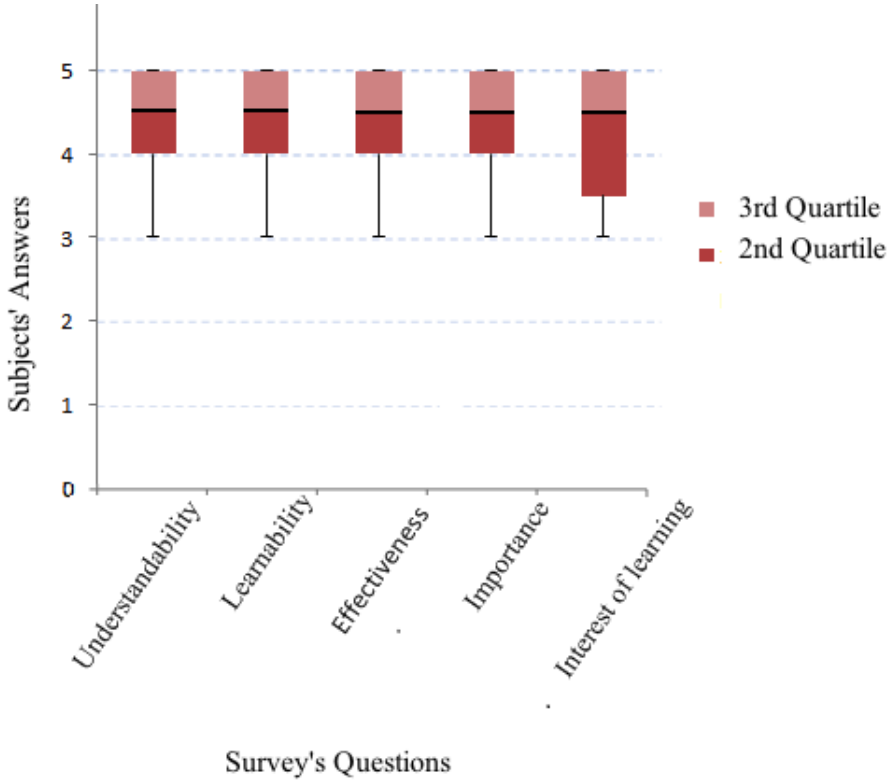


Figure 4.3: Design Phase: GP Survey Results

(1 = Least and 5 = Most)

There was a single feedback comment from this group. One participant suggested using the blue colour instead of the orange as it is more appealing and comfortable to be used.

Looking at figure 4.3, all measurements were coincidentally the same, except the *interest of learning* which was slightly lower than the rest.

4.1.2 The Main Findings

In total, this survey was answered by 21 different participants that were categorized into three different groups (participants specifications can be found in Section 3.5). The sessions showed that most of the participants comprehended the illustrated examples well. They were able to deal with Kit's menu and make correct sequence of actions most of the time, except for those in the JGS group. The concept of flow-based programming was quite understandable, and did not require any previous knowledge to comprehend the implemented examples. However, when they were trying to implement the examples 8 out of 21 participants found it problematic (see Figure 4.4)[1]. Half of those who suggested that it was challenging to follow the session's contents without having a basic programming knowledge were from the JGS group, which is a consistent with their low evaluation scores.

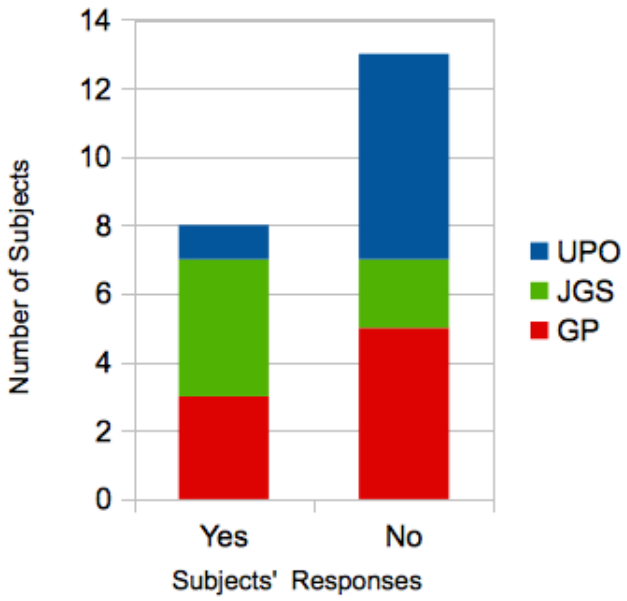


Figure 4.4: Design Phase: Participants Responses for the Lack of Programming Effect Question

[1]The colours in this figure are used consistently through this chapter to indicate the different groups of subjects. Blue colour indicates the UPO group, green colour indicates the JGS group, and red colour indicates the GP group.

The UPO group shared some useful recommendations regarding data importation and browser compatibility as discussed before. They found it easy to learn and understand the environment. However, their evaluation for *effectiveness* implies their doubt of Kit's ability to handle more complicated inquiries. Their highest evaluation was given to the *importance* of having an effective and easy-to-use data manipulation tool, which suggests the importance of having such tools. It might also reflect their desire to not offend the researcher. The JGS were the least active group. Their scores were the lowest among all the groups for the *understandability* and *learnability* measurements. Despite their relatively high score for *interest of learning aspect*, they did not show their interest while in the session[2]. The GP group showed lots of enthusiasm and cooperation. Participants from this group assigned high evaluations to the surveyed aspects of Kit.

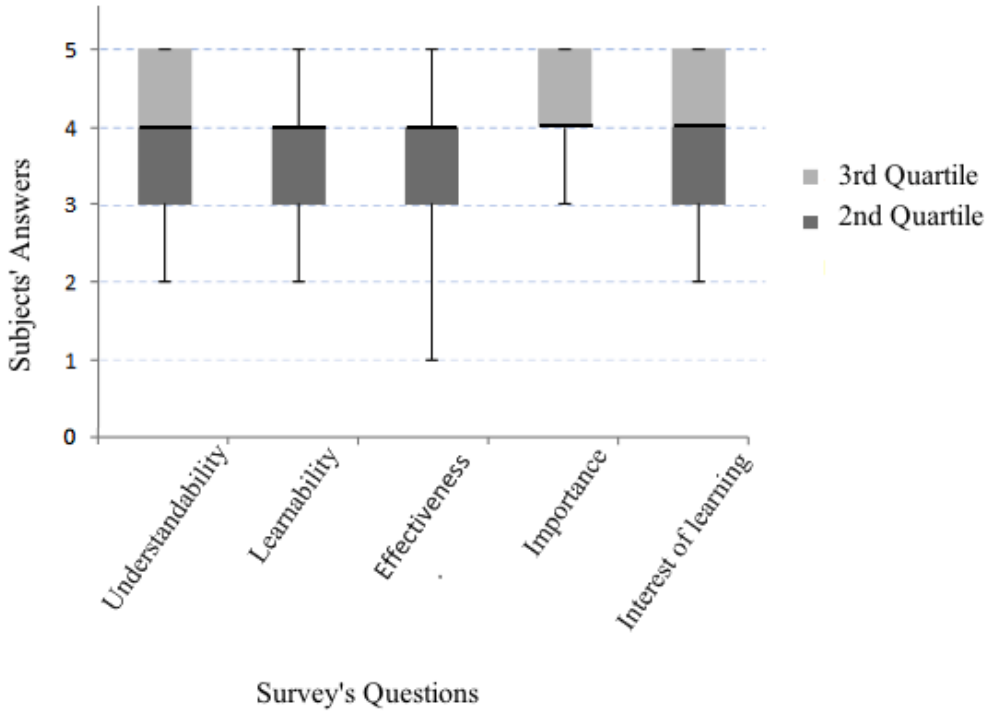


Figure 4.5: Design Phase: Survey Results - All Participants

(1 = Least and 5 = Most)

As can be seen in Figure 4.5, the lowest minimum score was assigned to *effectiveness* with a value of 1, while the highest minimum value was assigned to *importance*. All the

[2]I had to ask some of them to kindly stop working on side unrelated topics.

surveyed measurements had the same median value of 4, and they shared the maximum value of 5.

4.2 Design Phase Results

In this phase, the main research question was: *how can Kit facilitate processing real data inquiries for non-programmers?* During this phase, participants were given four different problem scenarios (the Problem Scenarios were previously explained in Section 3.2.1). Participants of this phase were the same as those who had participated previously in the introduction phase. They were encouraged to discuss their approach to solve problems in terms of suggesting some data transformations (to be translated into new blocks). The session discussed data-related activities such as: what data sources might be used?, what data fields might be imported[3]?, how can data be customized to the desired structure?, how can data be manipulated?, and finally how would it be visualized?. The duration of this session was about the same for all the groups and it was around 60 minutes with a total of 21 participants.
The objectives of this phase were:

1. to assess non-programmers ability to provide solution models for the given problem scenarios.

2. detect some of non-programmers needs and limitations (whether these needs were related to the programming environment or associated to the users' approach of solving problems).

3. to provide some design recommendation to accommodate these needs.

4. to suggest more functionalities to be added to make Kit capable of handling complicated inquiries.

Unsurprisingly, the UPO group had provided the most professional data analysis models. Therefore, these solution models were used to analyze the functional requirements of Kit. Many functionalities were derived from these models and were successfully integrated into Kit environment as new blocks. The JGS group was overwhelmed with the data analysis. They couldn't come up with additional ideas, and it took them a while to understand the blocks suggested by the other groups most of the time. The GP

[3]Despite encouraging participants to come up with new data attributes to be manipulated during the session, participants used pre-determined ones because of lack of the time during the session.

group was very active and suggested many ideas, members were able to use the visualization of the output correctly most of the time . However, they were lost in the middle and were considerably challenged on how to put ideas together sometimes (further details can be found in Section 4.2.2).

Through this section, the first subsection presents the UPO suggested solution models, the evolution process of the models to elicit new functionalities and design ideas, and the implementation of Kit after developing the required blocks. The second subsection demonstrates a thematic analysis for the focus groups sessions. The third subsection lists some findings and derive some recommendations to enhance the design of Kit.

Following are the suggested solutions for the problem scenarios. For further details related to any of the mentioned Kit's blocks refer to Appendix C: Kit Manual, which illustrates the functionality of each block by giving an example of it, and Appendix D:Kit Code, which includes the source code of the newly added blocks.

4.2.1 Problem Scenarios

4.2.1.1 The Obesity Study Scenario

The problem Scenario:

> "Emily is a Nutrition's student. She is working on a study to find the link between the obesity rate in different countries with the following factors: GDP (Gross Domestic Product), average educational level, and ethnicity. "

I assumed that I have the following tables from the United Nation's databases[4]:

1. The obesity rate for the world's countries. presented by "Country" and "Obesity Rate" attributes:

Country	Obesity Rate

2. The GDP of each country, presented by "Country" and "GDP" attributes:

Country	GDP

[4]UNdata is a good source of data related to the UN organizations[90]:
the website can be found in:`http://data.un.org/DataMartInfo.aspx`

3. The average educational Level for each country, presented by "Country" and "Average Education" attributes:

Country	Avg. Education

4. The major ethnicity of each country, presented by "Country" and "Ethnicity" attributes:

Country	Ethnicity

The participants suggested that it would be ideal to have the different factors plotted in a graph with the country variable as an x-axis. The rest of the factors were visualized as multi coloured y-axes. When participants were asked to translate that into blocks and pipes, they came up with the solution model presented in Figure 4.6. They suggested to merge the tables with the country key being matched to produce three-tuples of data, each tuple contained country, obesity rate, and a third attribute. Next, create three different data points of three dimensions to hold these tuples for each single country. The points were coloured according to the third attribute added to that tuple. In the shown example, **Red** was used for the tuple that contains "Average Income" value, **Blue** was used for the tuple contains "Education" value, and **Green** was used for the tuple contains "Ethnicity" value.

This model was modified later by adding a **Copy** block, and editing the **Merge**[5] block which was renamed later to **Filter**. **Filter** filters the tables by matching the first two inputs. The first input was the first table's key, whereas the second input was the second table's key. If a match was detected, the block would output the key (once) along with the rest of the corresponding input values. Each pair of tables were filtered separately to create a 2D point instead of the 3D point suggested before. The use of **Filter** block will ensure to match those countries that are in the main data table of with available obesity rate value only. The modified version is illustrated in Figure 4.7.

In Figure 4.7, participants have suggested some already existing blocks, and they were able to come up with new ideas as well. The previously-existent functionalities are:

1. **Copy**: it duplicates the input value as many as required.

[5]Kit had a **Merge** block which merged multiple values into a single column

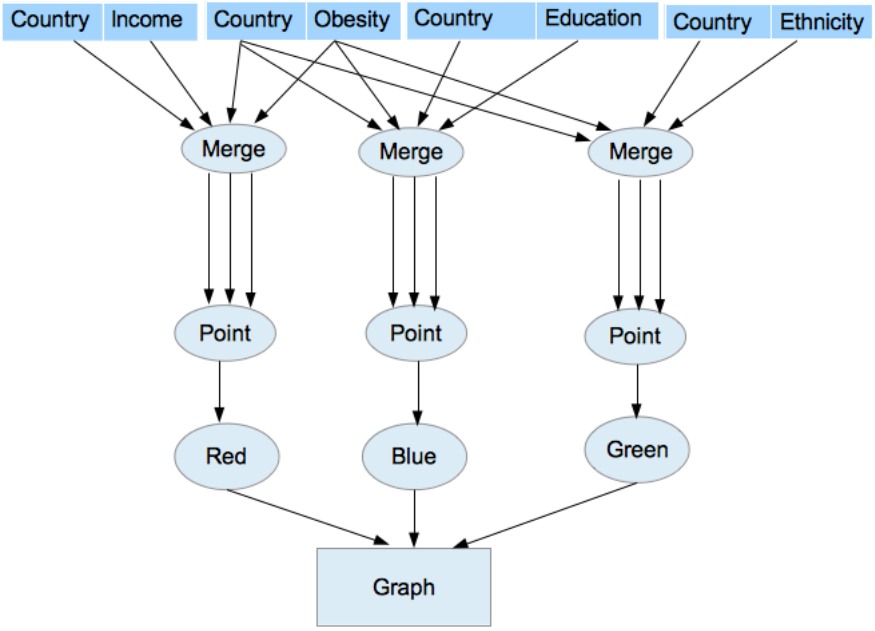


Figure 4.6: Design Phase: The First Problem Scenario - Suggested Model

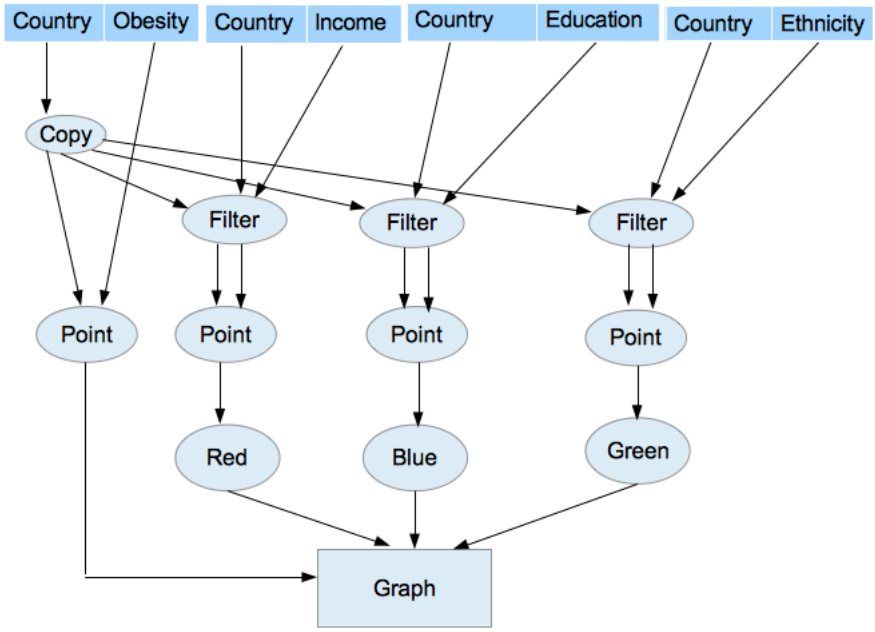


Figure 4.7: Design Phase: The First Problem Scenario - Modified Model

2. **Point**: it combines the x and y axes to have a single point.

The newly added functionalities are:

1. **Filter**: which is responsible to match any two table's keys (the first two input pipes), once the match is found the key's value is outputted along with all the correlated input values.

2. Colouring blocks such as: Red, Blue, Green, and Yellow which are categorized as visualization functionalities[6].

After implementing the suggested functionalities, a corresponding Kit model was built as required. The model is shown in Figure 4.8

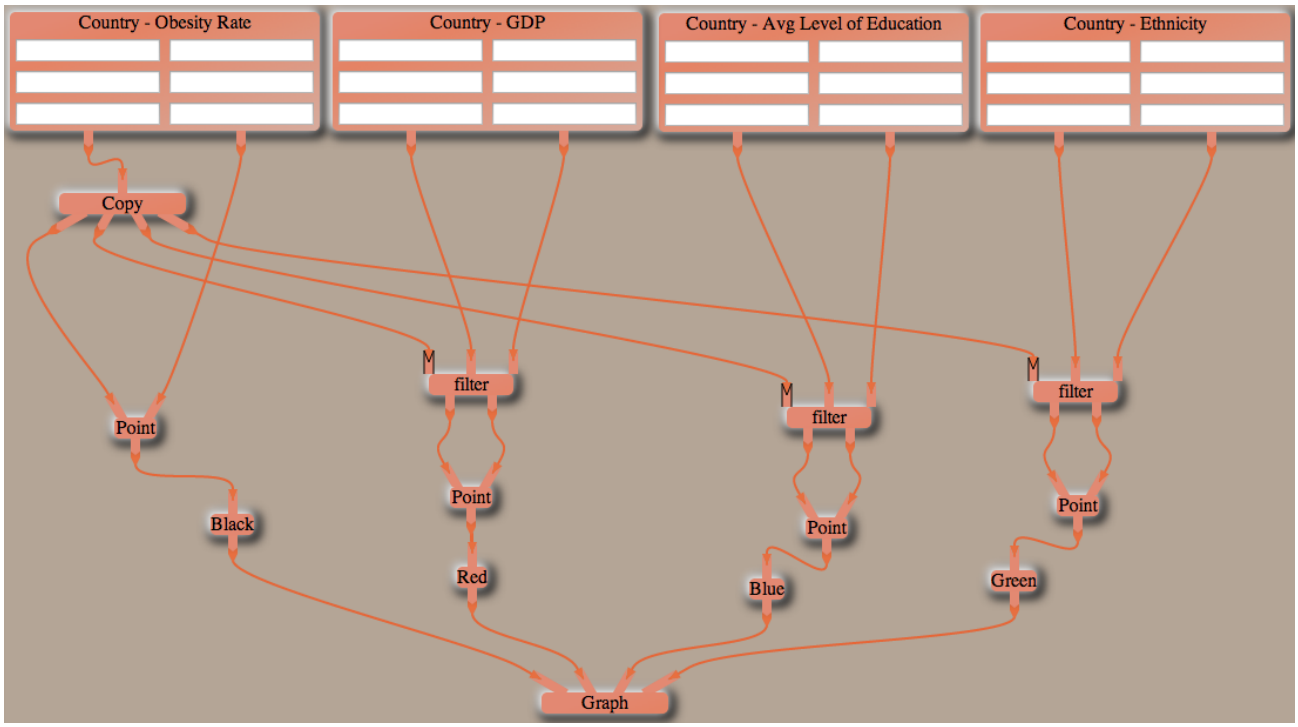


Figure 4.8: Design Phase: First Kit Model

[6]Another team member has been working on the visualization's implementation.

4.2.1.2 The Real-estate Agent Scenario

The problem Scenario:

> "Jacob is a real-estate agent. He wants to study the most profitable regions in a given city to invest in a property. He must study the history of sale price and some other factors."

I assumed having the following tables:

1. Some facts related to the previous sales history for a list of postal codes, which can be imported from a real-estate agency in the following format:

Postal Code	Price1	Price2	Total	Sold

   This table includes: the "Postal Code" value as a key, "Price1" as an average sale price for the first interval of time, "Price2" as an average sale price for a second interval of time, "Total" as the total number of houses offered for sale in the area, and "Sold" as the number of houses actually sold in the area.

2. Some facts about the residents of the region, which can be imported from Statistics Canada[7]:

Postal Code	Average Income

   This table includes each "Postal code" with its matching "Average Household Income".

The suggested Analysis was to find some variables that could be used to predict the change of a property's price based on its location. These factors are visualized by colouring the map according to the value of each causative factor differently. The suggested model is illustrated in Figure 4.9.

The suggested factors are:

- The difference between "Price1" and "Price2", as it shows whether the sales is going higher or lower within the region. They also suggested to identify whether the difference was an increment or a decrement.
- The ratio of "Total" and "Sold" as it shows how easy it is to (re)sell a property in that region.
- The "Average Household" income" as it shows the financial situation of the region's residents.

[7] http://www.statcan.gc.ca/

- If the "Average Household Income" for a certain postal code is not available, assign the average of the "Average Household Income" value to that postal code.

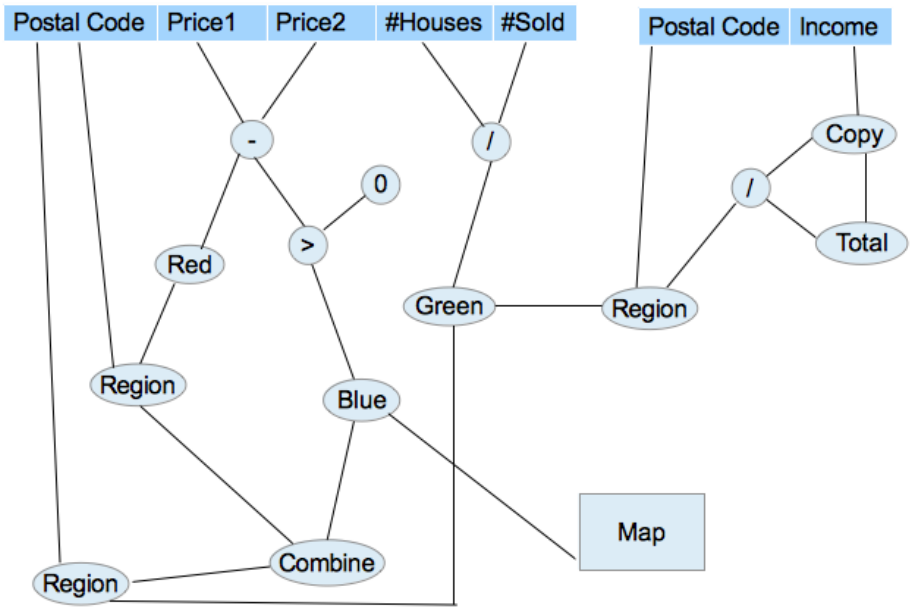


Figure 4.9: DesignDesign Phase: The Second Problem Scenario - Suggested Model

As can be seen in the suggested model, the analysis was not as clear as the one from the previous problem scenario. The scenario here was more complicated and required more functionalities to be used.
The challenge in this problem scenario was to calculate the average of "Average Household Income" attribute. In Kit, a block could not receive a new tuple of data unless the previous tuple was fully consumed and outputted. In this model, "Avg. Income" attribute was the only consumed value in the tuple."Postal Code" attribute has not been inputted in any functionality to be consumed at the same pace. This problem caused the processing to get stuck right after consuming the first value of 'Avg. Income" attribute. To resolve this issue, a **Synchronize** block was suggested, to consume the unused values of the postal code attribute.
The model was modified in Figure 4.10, and the newly added functionalities were as follows:

1. A **synchronize** block to consume the unused attributes and keep the appropriate flow of the data.

2. A **LookUp** block to match the two tables was added.

3. A **Normalize** block to scale the values from 0 to 1 to determine the density of the colour assigned to the region with 1 as the highest density, and 0 is the lowest, and that was a better replacement for the ">0" part.

4. Some visualization blocks such as colours, **Region**, and **Map** blocks. Which are part of another team member's work.

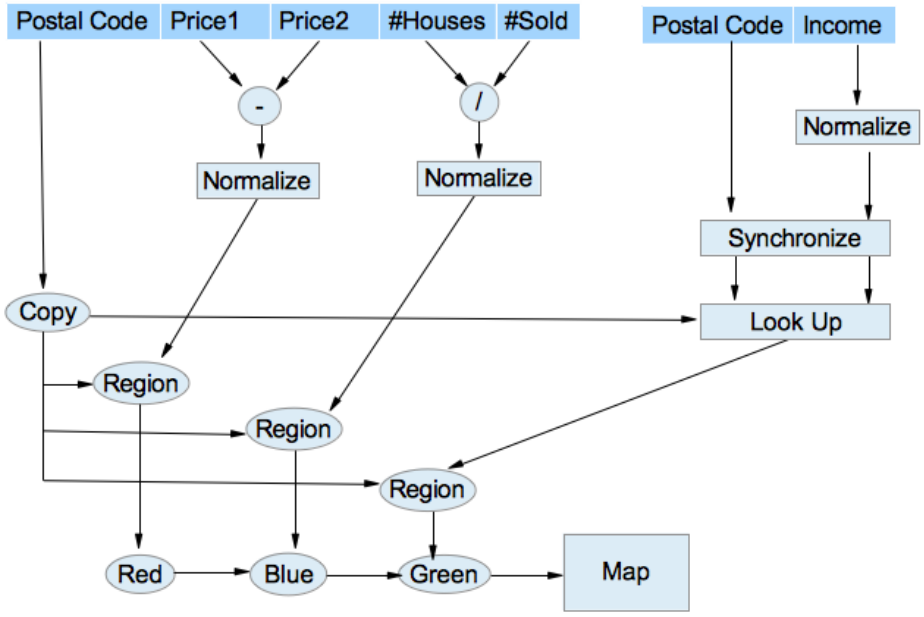


Figure 4.10: Design Phase: The Second Problem Scenario - First Modified Model

Once all the previous blocks were created, the normalized values were to be fed into the colouring blocks to decide the density of the colours as an indication for the value of the visualized variables. The visualization functionalities were reorganized as well to match Kit's syntax.

Keeping in mind that the main objective of developing Kit was to have a simple and understandable tool for naïve programmers, **Synchronize** block was not an easy concept for the audience to understand. In order to understand how **Synchronize** works, participants needed to understand the flow's mechanism and how the data is consumed. This block was a challenging concept and it would be better to come up with a more natural way of solving the flow of data problem. The alternative was to replace **Synchronize** with a buffer area where values can be consumed and saved to be used later within the block's scope. This way, users do not need to worry about the balance of

attributes consumption of any table.

The **LookUp** block has been changed to match the two tables by comparing their keys. If a match is found, the corresponding "Avg. Income" is outputted with the matching "Postal Code" value. If not, the average of "Avg. Income" is assigned to the "Postal Code" of the first table.

Next, **Normalize** and other visualization's functionalities were reordered to be located in the correct place to emulate a more natural way of user's way of thinking.

Finally, more functionalities were added such as:

1. A **Total** block that would calculate the sum of a single attribute's values directly.
2. A **Count** block that counts the number of values available for a single attribute.

The final model after applying the previous modifications is illustrated by Figure 4.11.

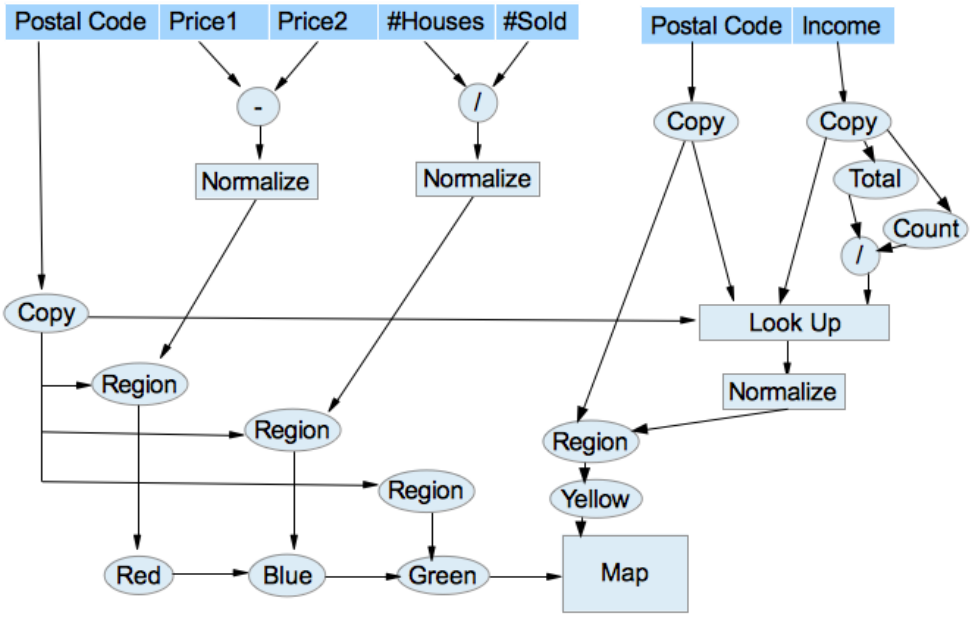


Figure 4.11: Design Phase: The Second Problem Scenario - Second Modified Model

After implementing the model using Kit, Figure 4.12 shows the final Kit Model for this problem scenario.

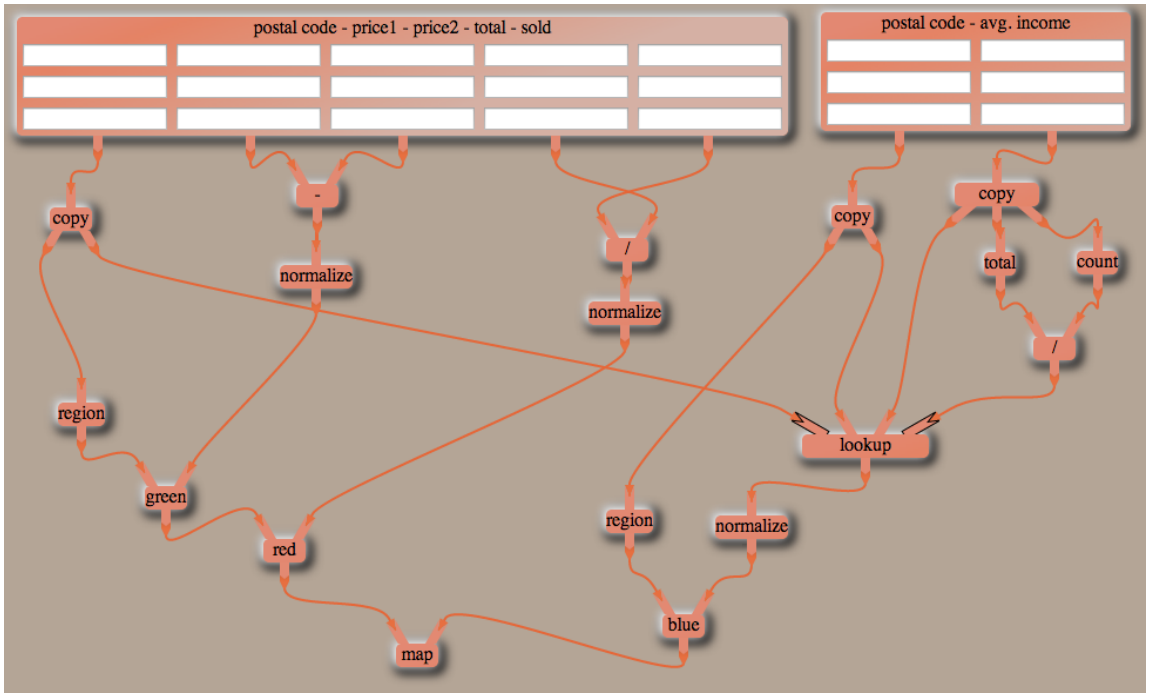


Figure 4.12: Design Phase: Second Kit Model

4.2.1.3 The Teacher's Scenario

The problem Scenario:

> "Lily is a teacher who wants to compare the students' performance in two different semesters to compare their progress in three different subjects."

Assuming we have the following tables:

1. The students' performance in Language, Mathematics, and Science in the first semester:

First Semester			
Student	Language	Math	Science

2. The students' performance in Language, Mathematics, and Science in the second semester:

Second Semester			
Student	Language	Math	Science

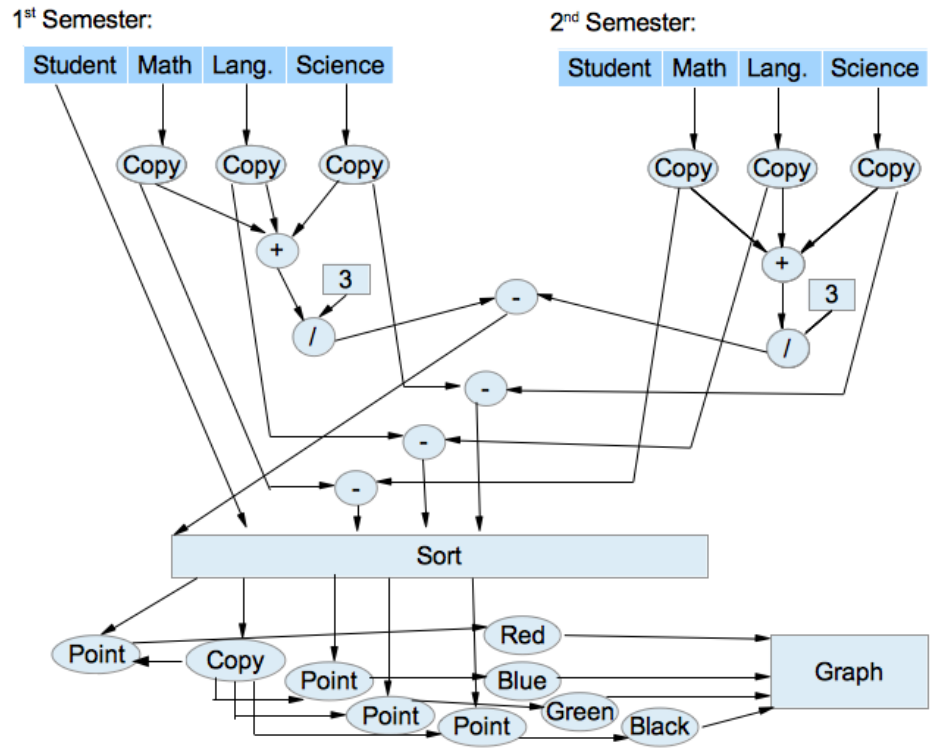


Figure 4.13: Design Phase: The Third Problem Scenario - Suggested Model

In this problem scenario, participants suggested to plot the difference of the students' performance during the two semesters for each single subject. They also suggested to calculate the average performance for each semester and calculate the difference of the two averages. Each student had four variables to measure the performance: the difference of each subject, and the difference of the overall performance. Each one of these four variables was visualized using a different colour. The last suggested idea was to sort the students' names based on the difference of their average performances to show the students with the highest grades' variance first. The suggested model is illustrated in Figure4.13

The only modification that was applied to this model is that I changed the location of **Sort** block. Sorting the students name will ensure having both tables with the same order of students names. Only then, the calculated differences are done correctly assuming we have the same list of students in both tables. Having a plot with differently coloured points for each measurement will cover the need to sort the calculated values as the comparison can be easily done by looking at the graph.The modified version is shown in Figure 4.14.

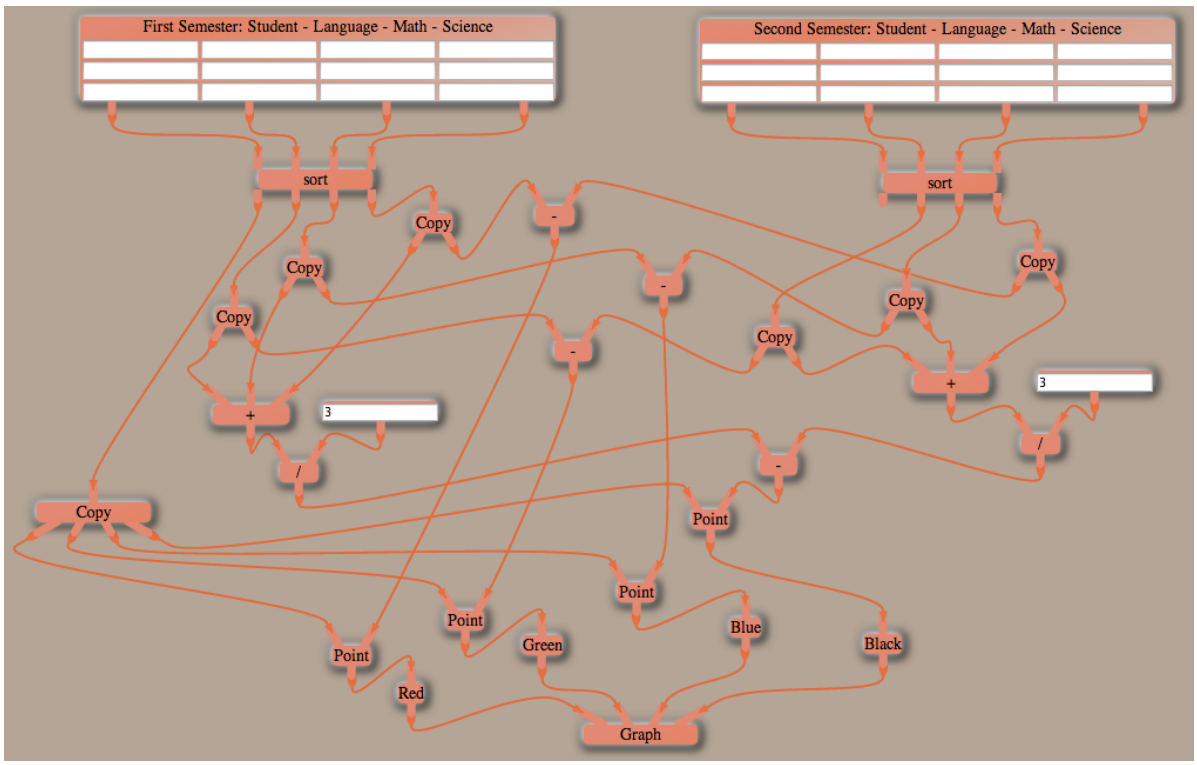


Figure 4.14: Design Phase: Third Kit Model

In this model, no new functionalities were added. I had already added all the suggested blocks such as:

1. Basic arithmetic operations.
2. "Sort" block, which sort the tuples based on the first input attribute.

The only part to be added is the visualization block such as the colouring and plotting blocks, which is done by another team member.

4.2.1.4 The Traffic Scenario

The problem Scenario:

> "William is a traffic engineer who works on road congestions. William is trying to find out the relationship between each congested intersection and the count of each different type of vehicles (i.e cars, buses, trucks, etc.) during different periods of time."

Assuming that I have the following information from the city's website:

1. A table has a list of intersections with number of each type of vehicles passing that intersection during the peak-hours:

Peak Hours				
Intersection	Buses	Trucks	Cars	Bikes

2. The same previously described table, but during the off-peak hours:

Off-Peak Hours				
Itersection	Buses	Trucks	Cars	Bikes

Participants' suggestions are modelled in Figure 4.15 , and those suggestion are listed below.

- Add a **Selector** functionality to pick which mode is to manipulate (i.e peak, or off-peak).
- Add a **Filter** block that contains a list of the required intersections to be filtered out of the original table.
- Add "Buses" to "Trucks"to be considered as a single group (They are both big vehicles).
- Multiply the value of each vehicle's type by a specific coefficient value to give it a certain weight.
- Add all the weighted values and feed the result into a **point** block to be created with a proportioned size to the assigned weight value.
- After creating the **point** with the desired size, locate it in its appropriate intersection in the map and colour it according to the types of vehicles.

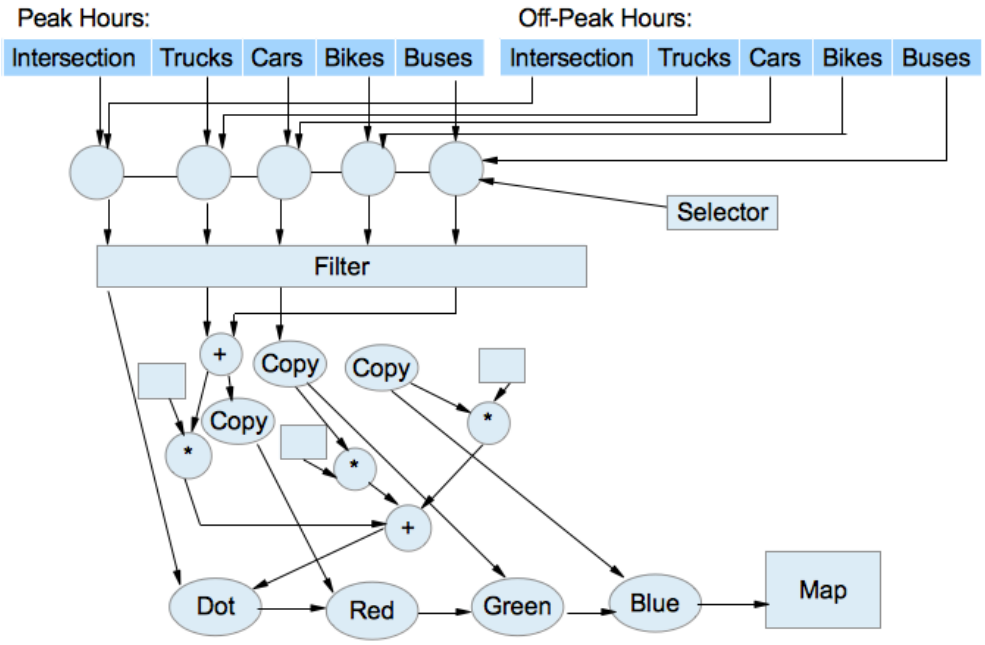


Figure 4.15: Design Phase: The Fourth Problem Scenario - Suggested Model

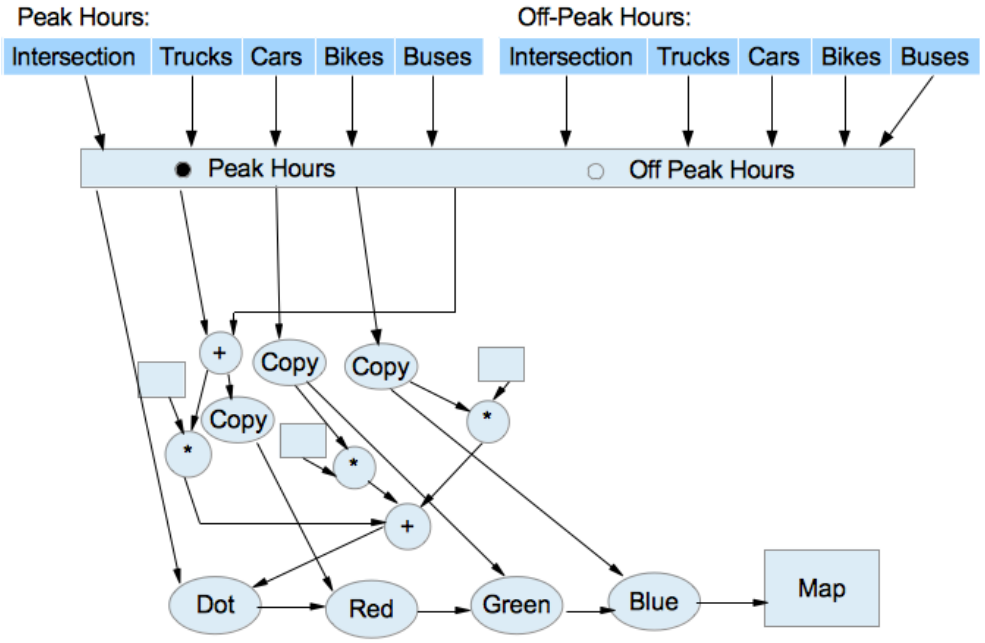


Figure 4.16: Design Phase: The Fourth Problem Scenario - Modified Model

- Each type will be presented in a different colour, all fed into the same point, for each intersection, in the map.

I added a **Switch** block to collect all the required modes as a group of radio buttons for the user to select the desired corresponding values based on the selected mode. The filter option can be added optionally by using the previously explained **Filter** block. The modified version is shown in Figure 4.16, and prototyped in Figure 4.17 as the only newly added functionality of the radio buttons option was not implemented till the time of writing this thesis.

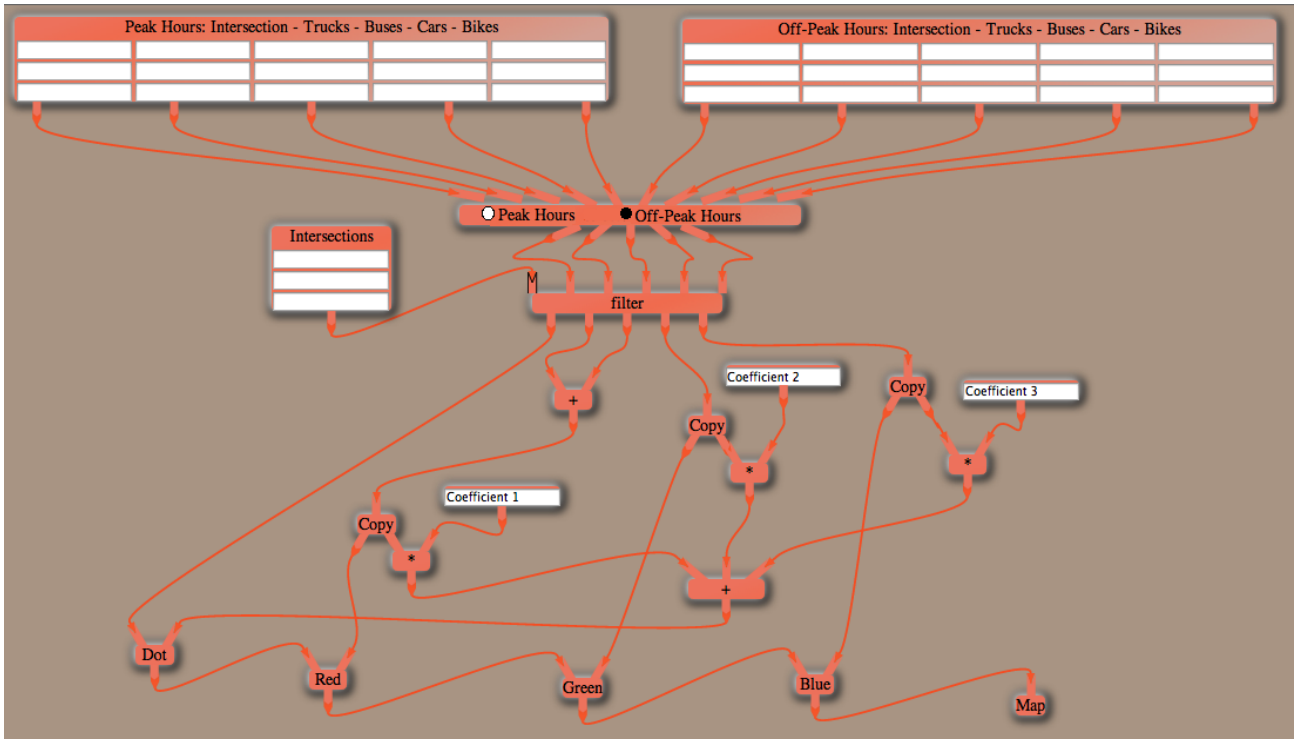


Figure 4.17: Design Phase: Fourth Kit Model

4.2.2 Thematic Analysis

As discussed before, the main focus in the design phase was to understand design requirements for the audience of Kit. Focus groups were freely structured with some previously designed problem scenarios to motivate participants to share new ideas and recommendations.

The sessions were audio recorded, then carefully transcribed and analyzed following the general guidelines of the thematic analysis methodology specified by Braun and Clarke [12]. Thematic analysis is a flexible qualitative analysis that allows researchers to look for general themes to explicitly highlight the most important aspects of the collected raw data.

The suggested themes are discussed as below:

4.2.2.1 Data Related Observations

The participating groups included those who are professionally data manipulators such as UPO employees, and those who are not. I was looking for their ability to collect required data, customize it to obtain the desired attributes, process it, and finally visualize it. Data related observations were categorized into the following taxonomy.

Collecting Data

This sub-theme included all the activities that were related to participants' ability to define input datasets required for the analysis, along with some potential sources of these datasets. Participants from all groups were able to suggest some useful variables (data fields)[8] to analyze the given problem scenarios such as: area code, current value, crime rate, etc. for the real-estate agent example. Additionally, participants suggested good examples of data sources such as Stats Canada, Census, police records, databases offered by the city, traffic control database, etc.

Customizing Data

After data was collected, it needed to be reshaped to include the required data fields and data records[9] and exclude the unnecessary ones. Participants from all groups suggested a merging functionality, and they were able to use it properly

[8]Data fields (or attributes) refers to the columns of a dataset.
[9]Data records refer to the rows of a dataset.

to merge two (or more) "in" tables into a single "out" table. They used phrases such as: "It would connect data", and "merge, because we are comparing [contents of] tables"[10]. Additionally, participants were able to reorganize contents of tables suggesting a sort block, and it was used with respect to the inTab order properly (i.e. contents were sorted with respect to the values of the first inTab of the block). Participants suggested to add a filter functionality where a user could limit the processed records to a user-specified constrained values. Additionally, they suggested a selector block to switch between modes (such as peak hours and off-peak hours) to limit data access to the values of the selected mode.

Processing Data

Once datasets were customized to the desired shape (fields and records), users had to process them by suggesting an analysis model and implement it. UPOs as domain experts had the best analyzing skills as they were able to suggest a solution model for each given problem scenarios. Meanwhile, participants from the GP group were able to come up with solution models for two out of the four scenarios. They were able to suggest correlations and computations such as suggesting a correlation variable to understand the relationship (dependency) between two different parameters. Simple mathematical operations such as addition and subtraction were easily used by all groups. A new idea was discussed in the UPO session when participants suggested to have a multiplicative factor that gives different weights for different factors. This idea was suggested to give different vehicle type a different weight of causing road congestion. Participating groups were able to come up with scaling blocks. UPO participants suggested a normalization block to scale the values from 0 to 1. Similarly, GP participants were able to come up with a percentage block to scale values from 1 to 100. Most participants were able to suggest an average block[11]. Ratio block was also suggested to indicate the relationship between two numbers by dividing one by the other.

Visualizing Data

Once the analysis was determined and translated into blocks and pipes, the output needs to be visualized. Out of the various available visualization techniques, users need to pick the right visualization that highlights useful insights out of the manipulated data. Participants from all groups were able to recognize and suggest various numbers of visualization techniques such as line graphs, bar graphs,

[10] Both comments were made by those who are not data manipulators.

[11] JGS participants were excepted. They were unable to give the appropriate formula to calculate the average.

pie charts, scattered plot, and maps. They were able to apply lots many features to indicate different aspects, such as using different colours for different parameters, scale colours into different shades, and use proportional size of visualized entities to indicate values. Additionally, they were able to describe labels and legends.

4.2.2.2 Reasoning Related Observations

In this section, analysis covers some hard to ignore observations related to different ways of thinking about the same problem.

Order and Organization

This sub-theme was related to the participants' way of choosing their favourable layout for the suggested model. When participants were trying to suggest visualizations, they had always different views on how to use the axes. Some of them recommended certain attributes to be presented on the x-axis, while others preferred the y-axis for an easier interpretation of the graph. One of the used expressions were: "Wouldn't it be better if we did the opposite?", or "It is easier to leave the country down here." What was really noticeable is that participants were trying to explain their point of view thoroughly to each other trying to convince the rest of the group with their "optimum" view. They were also able to create two output tables that were visualized into two similar graphs/maps for a side-by-side comparison.

Variables Problems

This sub-theme was related to the participants' encountered problems after defining variables. As mentioned before in "Collecting Data" sub-theme, participants were able to suggest many data attributes to be used in the process of solving the problem scenarios. Some participants seemed to over-look data attributes. They suggested more than ten attributes to one of the problem scenarios (the real estate scenario). Some suggested variables were hard to measure and compare (such as convenience, and facilities) to predict the house value in the real-estate agent scenario. Nevertheless, when they came down to analyze data, they could figure out that such attributes were hard to be measured and compared.

Detailed Requirements

This sub-theme was related to the situation where participants would ask for very specific definitions or constraints. During the design session, domain experts asked more detailed question compared to other participants. One of their

concerns was: what if filter block was unable to match two tables because of the different format for the same key value (such as writing the same zip code with and without a space)? They asked for specific definitions to the suggested attributes such as: are grades represented in numeric or alphabetic? (teacher scenario); What are the region boundaries? (real estate agent scenario); what is the definition of peak and off-peak hours, is congestion measured by cameras? can cameras detect the car make? (traffic engineer scenario).

Groups Variation

This sub-theme was related to the overall assessment for each group, and how the diversity among participants showed different analytical skills. In the GP session, participants were curious to see what the domain experts had suggested for each problem scenario after they were done with the discussion. I did not mind showing them the suggested models since I thought it might get them motivated to provide better analyses. The real-estate agent scenario was an overwhelming example to the GP group. They spent most of the time thinking about what attributes might help to predict the value of a house. Then, they were unable to think of a way to process the suggested attributes to create a good solution model. When they saw the analysis suggested by UPO participants I recorded the following comments:

- "I think we got too technical with this', that makes more sense."
- "We are thinking more mathematical than non-mathematical" this comment was made about their choice of a map visualization."
- "We looked more into details."
- "The other group sounds smarter than we are [laugh]."
- "They are more familiar than we are."
- "So you need the people who know and the people who do not know." In reply to the idea of that those models were suggested by domain experts.

However, I explained that this is the objective of having different participating groups. I wanted to observe the domain experts, yet ordinary individuals with average data knowledge are valuable as well.

JGS group as mentioned before was overwhelmed with the session. I tried to prompt their participation by giving them some ideas based on the previously discussed models created by the UPO group, they showed an acceptable degree of understanding but they were stuck at the mathematical part considerably. The

GP group were able to come up with various ideas, however the challenge was to put these ideas and suggestions together to build a coherent model. The UPO group did not encounter problems, yet they were always asking for more detailed specification for the problems.

4.2.2.3 Interpretation Related Observations

There were times when the participants' ability (or inability) to interpret the analysis was related to the understanding of the problem itself or part of solving it.

Goals Misunderstanding

In the real-estate agent problem scenario, some participants were unable to understand the general goal of the problem scenario. Two participants were unable to analyze the problem because they did not know what they were looking for. However, the rest of the participants volunteered to explain the problem. The conversation was as following:

Person 1: "Are we talking about how much to invest?"

Person 2: "No, what would make us wanna invest in it."

Person 3: "Which house to buy to make money."

Another example is:

Person 1: "What values are we looking at by the end?"

Person 2: "What are we looking for? I'm lost."

Both conversations were recorded at the same session (GP). The first one belonged to the real-estate scenario, and the second belonged to the traffic example.

Partial Misunderstanding

Sometimes, participants were stuck in understanding a specific part of the problem such as:

Person 1: "Why would you find the difference?"

Person 2: "So you know how much they improved or progressed."

Another example:

Person 1: "Type of vehicle? What does that have to do with traffic?"

Person 2: "That is true. It takes up more space."

4.2.2.4 Experience Related Observations

This theme discussed the effects of having previous experience related to the data-manipulation tools, or the problem domain.

Tool-Related Experience

This sub-theme contained observations made during the sessions about other tools of data manipulation and visualization. Popular tools such as Microsoft Excel influenced some participants sometimes.

A participant suggested to filter values manually, and fill an input table with the desired key value. I explained that we are talking about "Big Data" (see Section 2.6). She replied: "Yeah, I can put it in Excel."

Another tool was MLS[12]. Participants used examples of MLS visualizations to express their ideas on how to visualize certain data attributes on a map. Observations made in this section proved that even non-programmers are exposed to data and its applications as a part of their daily life.

Interestingly, some participants were influenced by Kit examples. When they were asked to think of a block to complete a given task, they were referring to the examples of the "Introduction" phase to express their suggestions.

Domain-Related Experience

This sub-theme discussed the domain-experience effect on the participants' ability to analyze data. When the teacher example was presented, GP participants had suggested that we had a teacher participating at the session, so she was the one to suggest that analysis. In fact, the teacher provided a solution similar to the one that was suggested by the UPO group. This coincidence showed that a person can provide better analysis if he or she had a good understanding of the problem domain itself. Meanwhile, experience can predict some potential problems. UPO's suggestion of providing a technique to match keys of tables in a smarter way is a good example of these predictions. Such awareness was derived from their previous experience of a similar issue.

4.2.3 The Main Findings

The data-related theme showed that some participants were able to come up with knowledge to be able to collect, structure, process, and visualize data. These par-

[12]MLS refers to Multiple listing Services such as: www.realtor.ca

ticipants were not necessarily programmers nor data manipulators which indicated that the capability of understanding and using the concept of manipulating data using blocks and pipes covers a relatively large audience (4.2.2.1).

The reasoning-related theme exposed some unexpected issues. The "order and organization" sub-theme showed that different people might want to represent the same piece of information differently (Section 4.2.2.2), meanwhile, the other sub-themes showed that each individual has their own style of thinking that is reflected in the way he or she thought of the problem. UPO participants tended to look more into details and definitions, while GP participants tended to focus on suggesting attributes rather than focusing on how to use these attributes.

Another important finding was the importance of considering the variation of the users (more about users' variation considerations can be found in [26, 83, 71]). Expert users needed some advanced features that will not be necessarily used by non-experts. Meanwhile, non-experts needed to be supported with some extra features to facilitate their ability to recognize the problem, suggest an appropriate solution model to it, and be able to interpret that model without getting lost in the details (Section 4.2.2.2 and Section 4.2.2.3).

The last theme showed the positive effect of having previous experience with similar tools or problem domains (Section 4.2.2.4).

Based on the finding stated above, the following design recommendations can be considered to enhance the usability of Kit when used by different users:

1. Integrate new functionalities that would be used in developing the suggested solution models. Some of these suggested blocks were: Filter, Normalize, Switch, and Average (more examples can be found in Section 4.2.1).

2. Design recommendations

    - add features that can facilitate transposing tables or switching axes to ease switching representation whenever needed.
    - give advanced users an access to expand certain features, or even add new ones to the environment such as specifying two different formats to match certain keys.
    - consider the general concepts and metaphors used in popular tools to make use of users' previous experience.

3. Help facilities

    - give users the ability to use secondary notations, which are extra visual cues that are not a part of the program's execution, such as enabling comments. This would enable users to maintain the understandability of the code without getting lost, and enable the user to add certain definitions for attributes.
    - provide some planning facilities that enable users to break down tasks and implement them progressively so they stay focused.
    - integrate some help examples and manuals to support new users who might not understand a certain part of the code.

4.3 Recognition Phase Results

The main research question here was: *How recognizable/readable are the generated Kit models?*.
Users of Kit should be able to recognize their previously implemented models, and be able to comprehend those that were built by others previously to facilitate code expanding and editing. Consequently, this would make Kit a good choice for those who don't want to waste their mental energy trying to remember what this part of the code does when they need to modify a program (more about aiding modifications by improving readability can be found in [36]).
The objective of this phase was to assess the created model's readability. Hence, participants were tested to measure their ability to recognize the models built in Kit and match these models with their corresponding scenarios. As discussed before (see Section 3.3), this phase began with an introductory example to refresh the participants' (15 participants) memory with Kit syntax and introduce some newly integrated functionalities. The "real estate" example was picked as an introductory example because it included lots of functionalities to be illustrated. However, I tried to expand the suggested model and add more blocks to expose participants to as many blocks as possible (refer to Section 3.3 to see the introductory model and the recognition test). This phase involved 15 participants, and the duration was about 30-40 minutes including the introductory example, and the participants questions after taking the test.

During the introductory part, participants asked the following questions:

1. "Can postal codes be determined by the region you select?" The question was answered by illustrating how **Filter** block works where user can easily filter contents

of a table with user-defined values.

2. "I'm just wondering what if I want to skip an output attribute?" The answer was: If you don't need the outTab, don't pipe it.

3. "Why some tabs are black?" The answer was to explain the control tabs where users have to manually feed these "black" tabs with values of their preference.

4. "Can copy block create as many copies as you need?" The answer was: yes, just right click the block and choose "AddOutput" as many times as required.

5. "What is the value of assigning average income as a default value?" The answer was that this value was suggested by others, but can be changed based on user's analysis (can be assigned to zero for example).

Such questions, showed that participants were able to interpret the model and ask about alternative scenarios or further details, rather than getting stuck in reading the presentation itself.

By the end of the introductory part of the session, participants were asked verbally about how understandable the model was. They all agreed with its understandability using the following expressions: "very understandable", "very straight forward", and "It's neat." Most of the participants asked whether the code actually runs or not. They were curious to see the output of the model. Unfortunately this part of the project was still under development by another member of Kit's team.

Finally, participants completed the recognition test which included three Kit models, with three scenarios provided for each model: a correct, a close, and a wrong answers. . Participants showed high confidence in the choices they made, and they were under the impression that they did well in the test. However, the results are illustrated in Figure 4.18.

As can be seen, most of the participants were able to exclude the wrong scenarios. The figure shows that the easiest question was the second one. All the participants except 2 from the JGS participants were able to recognize the correct scenario for the second model. The first and the third questions have similar results as 5 subjects were able to make the right choices. JGS was the only group who could not exclude the wrong choices[13]. Meanwhile, JGSs were not able to pick any correct choice out of the first

[13]JGS participants were hardly convinced to come back for validation as it was a very busy time of the semester, which might be an additional reason for their lack of interest as they are busy doing other work.

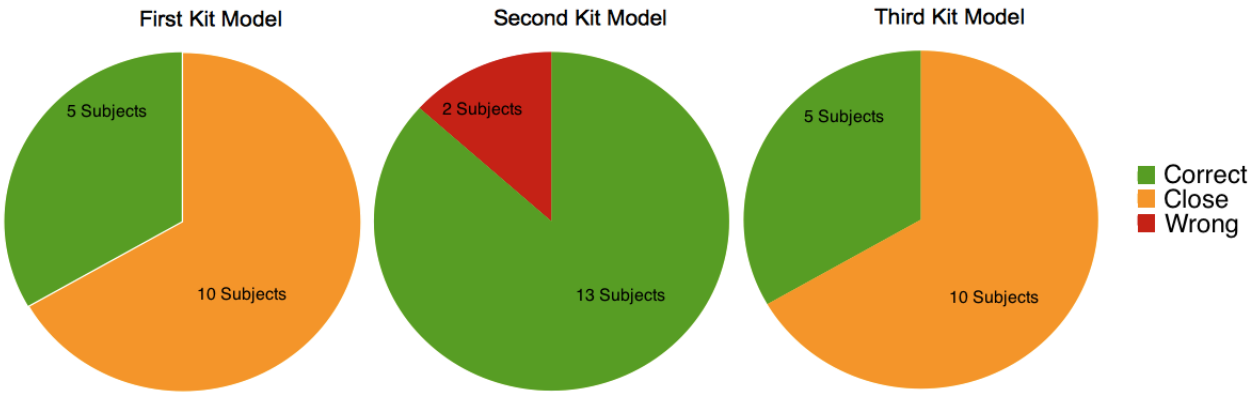


Figure 4.18: Recognition Phase: Test Results - All Participants

model. UPOs were challenged with the third model as one participant only was able to figure out the correct scenario. GPs had mixed results, however, the only participant who was able to figure out all the correct scenarios was from the GP group.

After the test was completed, participants were asked to share their ideas about how hard this test was. Most of participants, including those who made mistakes, showed high confidence in the choices they made. Some of them shared the confusing parts in the test such as:

- Some of the scenarios specify the colours blocks explicitly, while the others do not.
- The visualization techniques such as scatter plots and plot lines can not be distinguished in the model.
- One of the UPO participants suggested to colour the path of each attribute with a different colour for easier tracing of pipes.

When the correct answers were revealed, everyone agreed with the correct choices. Participants were asked if their incorrect answers have changed their impressions about Kit's recognizability, and the answer was no. No matter, how many correct or incorrect choices they made, participants were still satisfied with Kit's ability to represent models in a straight forward manner. One of the senior members of the UPO, who was skeptical of Kit's effectiveness during the introduction's session, was happy to share his satisfaction on how Kit turned out to be in the final stages of this study. He thought that Kit was a very promising data manipulation tool in spite of his test's score of 1 out of 3.

Extra Session

After finishing with the groups' sessions, I thought to ask some of Kit's team members to take the same test. Kit has two students (in addition to me) working on different aspects of the project. Their knowledge of my research was limited to the functionalities that I had added to Kit.
One member was able to get the first and the second scenarios correctly, whereas the second member made correct choices for the second and the third scenarios.

Surprisingly, the Kit's team members were not able to recognize all the correct answers. The scenarios were too ambiguous even for Kit team members. However, this test was a one time thing. It can not be retaken by the same participants since the test has already been revealed to them. Unfortunately we were unable to recruit new participants either.

4.3.1 The Main Findings

Aside from the test results, users were satisfied with the final models created by the researcher, which was evident when participants answered positively about how understandable they thought the models were. It seemed that the scenarios might be too ambiguous, which would explain the Kit's team members' inability to match the models with the correct scenarios. However, most participants were able to exclude the wrong answer which can be considered as a relatively positive result.
Overall, participants (including domain experts) shared positive impressions of the final outputs and thought that Kit is a very promising data-manipulation tool.

4.4 Interaction Phase Results

In this phase, the research questions were: *How easy it is to use Kit to perform a specific task?*

During this phase, I asked a subgroup of the participants to complete some given tasks using Kit (full description of this phase test can be found in 3.4). Four participants from the GP group were available to complete this test. They were given the tasks and provided with a "Kit Manual" booklet only. No further instructions were provided during the test. They were allowed to ask questions related to the data-analysis, but not Kit. Each session was about 45 minutes long.

Participants who participated in this phase were asked to complete the following tasks:

1. The first problem scenario:
   You have a 4*4 input stream. You need to calculate the sum of the first three values of each row, and the average of the third and fourth value. The results from both calculations will be output in a 2*4 output stream.

2. The second problem scenario:
   John is a sale manager who works in a department store. He wants to investigate the sales' performance throughout the past five years. Assuming that he has the sales history table for the last 5 years that includes: Year, Dept1, Dept2,Dept3, and Dept4 how would he retrieve the following information:

   (a) Avg. of sales for all the departments each year.

   (b) Avg. of sales for the last three years for each department.

   (c) Output the last year's sales only.

Following are the results of the four interactive sessions. The performance of Participant1 was described with the most thorough details. Participant2, Participant3, and Participant4 reacted similarly except for the described issues described later.

Participant1

Problem1:

This participant was a male who had been already a part of the previous stages of this study. He started solving the given problem by creating a data table with the required size. The participant clicked on Kit's window. When the user menu popped-up, he was able to choose the **Input** option followed by **In Table** to create a 1*1 table. The participant clicked on the table to check the options available for the table. He was able to find the required options of **Add Column** and **Add Row** to expand the size of the table.

After the table was created and filled with arbitrary values, the participant tried to find the "+" functionality by clicking on the table. He figured out that clicking on any block will show the options related to that block only. It was not hard for him then to click on the main window to pick **Math**, then + options. The participant linked the first two outTabs to the "+" block. He waited for a minute to

think how to add the third value. He tried to click on the addition block, then he picked **Add Output**. That was not what he needed. He clicked the same block again, and picked **Remove Output** to omit the wrong outTab. Then, he clicked the same block one more time, and fixed the inputs size by choosing **Add Input**. He added the last value by creating a third pipe to link the third attribute to the addition block.

Next, the average was to be created. The participant was hesitant on how to use the third attribute twice. He clicked on the attributes to check whether there are any available options to duplicate the value. The only way to do was is by creating a **Copy** block. He tried clicking on the main screen with no success. Then, he looked into the manual and realized that a **Copy** block will do the job. The participant created an **Average** block thinking that this how we calculate the average of each third and fourth values. I stepped in here to ask him to read what **Average** block does from the manual. He saw the manual's example, and realized what the mistake was. He asked me whether it is allowed to calculate the average using the basic arithmetic operations, I nodded and told him that this is how it should be done. He added a + block, and linked the correct values to the block's inTabs. He created a / block, and linked the outTab of the + block to it.

It took him a while to figure out how to divide the output result by 2 to find out the average. He tried to click almost everywhere before he figured out that what he needed was an input. When he picked the **Input** option from the menu, it was easy for him to recognize that the correct choice was a **Simple** input. He filled it with 2 and linked it to the / block. The last step of creating the output table was by clicking **Output**, followed by **OutTable** option.

Problem2:
This time, it was easier and faster for the participant to create the table, and adjust its size. The values were filled in no time. The participant added one extra row by mistake. However, he was able to fix that issue successfully. The next task was to calculate the average departments' sale per year. This time, calculating the average was easier and faster than the first time.
For the second part, the participant calculated the average of the five past years per department mistakenly. I had to ask him to read the question again. He was

able to fix the mistake. He looked into the manual, and created a **Drop** block to drop the first two years' tuples. I asked him what if the table was not sorted. After a second thought he created a **Filter** block instead of using **Drop**. Next, he looked for the filter table, he thought that it is automatically created once the block is placed. I informed him with the fact that he needs to create it manually. He did it successfully, and linked it to the **Filter** block. He clicked on the block to select **Add Field** to increase the number of inTabs, and outTabs. The participant linked all the attributes to the **Filter** table successfully.

The participant created four different **Average** blocks and fed each of them with an output coming out of **Filter**.
For the third and last part, the participant wanted to use **Filter** again. This time, I asked him to use **Drop** instead to try another functionality. The participant created the block, added new fields, and outputted them successfully with no troubles encountered.

Out of this session, my main observation was the participant's struggle to create and link pipes. The participant clicked outside the tab for most of the time because of the small size of the clickable area. It was kind of frustrating for him, that I had to offer my help in linking. However, he had to tell me where to click. When all the tasks were completed, I asked the participant to evaluate Kit on a scale from 1 to 5 (least to most) in the following aspects, and here is his responses:

Memorability: How memorable is Kit?	3
Understandability: How understandable is Kit?	4
Learnability: How easy is Kit to learn?	4
Ease of Use: How easy is Kit to use?	4

Meanwhile, I asked the user to share the most difficult part when using Kit, and the answer was creating the pipes. The tabs are small and hard to be clicked. The other complaint was a data-related issue, when he stated that it took him a while to understand the difference between the average of column's values, and the average of row's values, whether as a concept or as an implementation.

Participant2

Problem1:

This participant was a male who participated in all the previous sessions. When

he started with the first problem, he created two 2*1 tables instead of 2*2. I asked him to create a 2*2 table, and he was able to fix it. He calculated the sum, added the **Copy** block, and calculated the average successfully.

It took him some time to figure out the **Copy** block, but it was not hard for him to remember its usage from the previous sessions. The participant thought to add the first three attributes in two steps by adding the first two, then adding the third to the result the first sum. When he was asked to do it in one step, he was able to figure it out. This participant was under the impression that the output table is a place to save the output value to be used later. He was able to correct his assumption when he noticed that the output table is a dead-end block that has no outTabs. He misused the "Average" block in a similar way as the first participant, but then I asked him to check the manual's example he understood the difference.

Problem2:
The first part of the problem was completed successfully without any significant observation. The participant was able to create the table, fill it with data, calculate the average, and output the results correctly. The ease of this part comes from its similarity with the first problem.

For the second part, he used **Filter** block and created the filter table to be linked in. The participant was able to use **Average** block correctly this time. For the third part, the participants wanted to use **Filter** again to output the last year's sales only. I asked him to replace it with **Drop** to try a new block, and he was able to use it successfully.
When all the tasks were completed, I asked the participant to evaluate Kit on a scale from 1 to 5 (least to most) in the following aspects, and here are his responses:

Memorability: How memorable is Kit?	4
Understandability: How understandable is Kit?	4
Learnability: How easy is Kit to learn?	5
Ease of Use: How easy is Kit to use?	4

When the participant was asked about the encountered difficulties when he was using Kit, he suggested to add a tiny plus sign at the corner of the tab to show that the number of tabs per block is increasable.

Participant3

Problem1:

This participant was a male, who participated in all the previous sessions. This participant created the table, adjusted the size, and filled the cells easily with no problems encountered. As expected by now, he misused **Average** block same as the previous participants, but realized the difference later. When he created the output table, he added the columns correctly. He tried to find out how to add rows since the output has 4 rows. I explained to him that the number of rows is adjusted automatically based on the table's contents.

Problem2:

For the first part, the first observation made about this participant was that he did not respect the division order when calculating the average. He linked the simple input with the value of 4 to the first inTab of the division block, and linked the output of the sum block to the second inTab of the division block. He was supposed to do the opposite to get the correct answers. The second observation made was that he forgot to link the year to the output table after calculating the average and linking them to the output table. The output that he created contained the average of sales per year without indicating the year attribute in the output. When he was asked to fix this issue, he completed it successfully.

For the second part, The task was completed successfully. He was able to create the **Filter** block and link it to a created filter table. He adjusted the inTabs, and outTabs of **Filter** block. Next, he calculated the average per department. Finally he successfully outputted the results.

For the third part, participant used drop correctly, and outputted the results correctly. No difficulties were encountered. When all the tasks were completed, I asked the participant to evaluate Kit on a scale from 1 to 5 (least to most) in the following aspects, and here is his responses:

How memorable is Kit?	4
How understandable is Kit?	4
How easy is Kit to learn?	5
How easy is Kit to use?	4

When participant was asked about the encountered difficulties when he was using Kit, linking the blocks was the major difficulty when he used Kit. He also

suggested using different colours for different paths of data.

Participant4

Problem1:
This participant was a female, who had participated in all the previous sessions. This participant had some programming knowledge, and she was a good data-manipulator which explained her good performance compared to the other participants who completed this test. This participant could recall all the details and the functionalities presented in the previous sessions. The first part was done easily in no time with the correct output.

Problem2:
In the first part, the task was performed easily. For the second part, it was not hard for her to use **Filter** and **Average** blocks. She completed the task in an appropriate output format. For the third part, she was able to use both **Drop** and **Filter** blocks alternatively to complete the requested task.

Her responses to the usability's questions were the best among the participants, probably because of her knowledge in both programming and data-manipulation areas:

Memorability: How memorable is Kit?	5
Understandability: How understandable is Kit?	5
Learnability: How easy is Kit to learn?	5
Ease of Use: How easy is Kit to use?	4

When participant was asked about the encountered difficulties when she was using Kit, linking the blocks with pipes was the reason of giving ease of use the evaluation of 4.

By the end of the sessions all participants were able to manipulate the given tasks with minimal help. They had high evaluation for Kit's usability. Figure 4.19 shows participants' responses to the evaluation of Kit's usability:

4.4.1 The Main Findings

To summarize testing sessions, the encountered problems are listed below along with some preventative recommendations (most of these issues can be found in [70]):

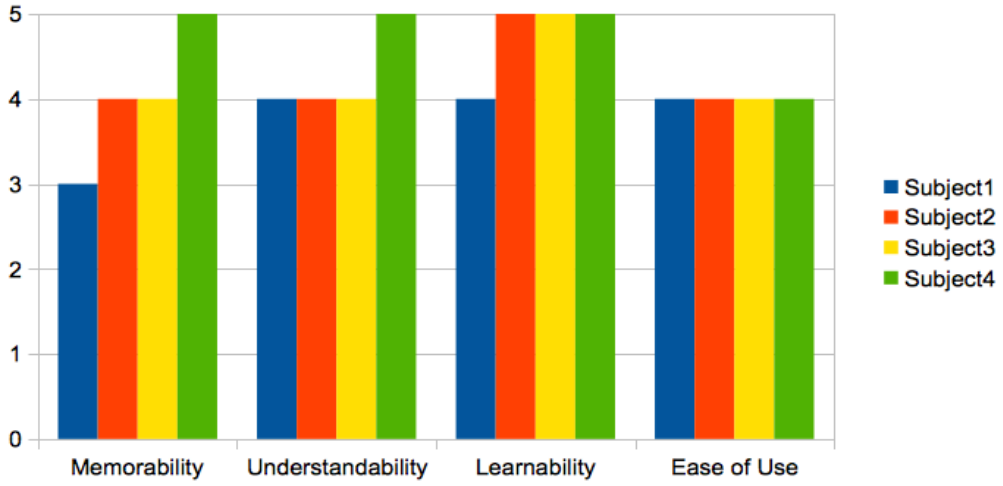


Figure 4.19: Testing Phase: Participants Evaluations of Kit's Usability

1. Users found it challenging to locate the block in the menu most of the time. The functionality might be more discoverable with different terms in the menus or a contextual search capability.

2. Users found it challenging to click on the tabs to create pipes linking blocks. This problem can be solved by reshaping the clickable area of the tab and/or probably enlarge it to make it easier to click.

3. When user needed to explore the features associated with the blocks, it was hard for them to locate these features (add inputs or outputs to the block for example). Participants clicked on the main screen before they figured out that they need to click on the block instead. This problem can be solved by using an expandable icon beside the block to expose the hidden features of each block.

4. Some users were unaware of the **Copy** block. Kit's development team agreed that copy block is not as natural as it should be. This should be replaced with a more natural technique, probably adding the feature to the pipe itself to show the possibility of replicating the value.

5. Most of the users were unclear about the difference between calculating the average of the values of the same row, and calculating the average of the values of the same column. Providing a help manual that shows how functions are used and showing a relevant example has already been created (see Appendix C).

6. When users created a **Filter** block, they did not know what they needed to do with the inTabs. The problem could be solved by adding a pop-up description describing the block's inputs and how the various inputs work whenever the block is created or pointed at.

7. When a block like '/' (division) was created, one user forgot that the order of the inputs matters. This problem can be solved by providing an enforcement technique to preserve the order of the non-commutative operations. An example would be to enable the inTabs individually with respect to their order.

8. Some users found it difficult to distinguish different paths through the diagram. This could be solved by providing tracing facilities such as using different colours to indicate the transformation of each single data attribute throughout the processing path starting from the input format, until the output is reached.

4.5 Summary

Through the different phases of this research, many different research questions were asked either to assess certain aspects of Kit's usability, or to suggest some techniques and mechanisms to be added to Kit to enhance its usability.

I used various usability evaluation tools such as focus groups, personas and scenario-of-use, surveys, task scenarios, and user observation. The main goals of this research were:

1. Assess non-programmers ability to solve data-related problems using a prototype of Kit, and highlight their needs and limitations.

2. Assess how usable the prototype of Kit is to be used by non-programmers.

3. Assess how capable the prototype of Kit is to handle realistic inquiries. And how easy it is to recognize/comprehend existing Kit models.

The main findings of the different phases of the research were:

1. Most participants showed adequate knowledge of data manipulation as can be seen in Section 4.2.2.1. However, the diversity of the participants showed different approaches and needs when solving problems (refer to Sections 4.2.2.3, 4.2.2.2, and 4.2.2.4).

2. When the usability was assessed for all the explicitly measured aspects of usability such as: learnability, understandability, effectiveness, importance, memorability, and ease of use, we found that Kit is mostly usable by non programmers.

3. Kit's models were recognizable to a certain degree. Most of the participants were able to exclude the wrong scenario, however they were not able to select the correct one (except for one of the models). The problem might reside in the design of the scenarios as most of the participants agreed on the understandability of the models created in Kit.

4. Kit's environment was capable to handle realistic inquiries with corresponding flow-based models.

Findings from the different phases of the study exposed many opportunities to enhance the environment usability such as:

1. enhance the visibility by making it easy for the users to see the available options of the environment.

2. enhance the discoverability by making it easy to discover the locations of the required actions from the environment's menu.

3. use the standard metaphors by considering the design of the popular data manipulation tools to make it easier for the users to use it.

4. enhance the scalability by making it easy for the users to track the data paths in the model either by compressing the view or provide coloration techniques for the different paths of data.

5. enhance the reliability by making the environment less error-prone (such as making it easier to click the tabs), and provide easy recovery facilities (such as fixing the output layout).

6. give users the support they need by considering different limitations and providing many help facilities such as manuals and secondary notations.

Chapter 5

Conclusions

My thesis is that *Kit can provide an easy to learn and use environment that facilitates data manipulation for non-programmers.*

To verify my thesis statement, I experimented on the usability of Kit, which represents the flow-based paradigm, to be used by non-programmers. These experiments were held in four different phases with a variety of usability evaluation techniques. The study was structured into four main phases, each of which contributed towards answering one or more of the study's objectives.

The objectives of this study are: First, I wanted to evaluate non-programmers ability to come up with data-flow models to some realistic data-related problem scenarios and shed some light on some of their needs and limitations. Second, I wanted to assess how easy it is for non-programmers to understand and use the available prototype of Kit. Third, I wanted to assess how capable Kit is to handle some real-life like inquiries and represent these inquiries into easy to comprehend models.

Throughout the different phases of this study, we asked various research questions to assess the usability of Kit and to address some design enhancement's opportunities. The results were mostly positive . However, the usability evaluation techniques used in this research, highlighted lots of recommendations to improve Kit's capability to accommodate the different needs of non-programmers.

5.1 Findings

The findings of the study's different phases can be summarized in these points:

1. Non-programmers have shown adequate capability of manipulating data using the flow-based paradigm. However, the diversity of participants exposed different needs to be considered by developers of similar environments (for further details see 4.2.3).

2. Non-programmers found Kit mostly usable. This finding was derived from the results of the survey and from users' ability to create flow-based models during the different phases of the study.

3. In terms of recognizability, the results were somewhat mixed. Most of the subjects were able to exclude the wrong scenarios when asked to match Kit models with their corresponding scenarios, but they were less successful at recognizing the correct solution. We attribute this more to our ambiguous problem description than to an actual problem with Kit.

4. We believe that the scenarios we used for these focus group represented realistic and somewhat complicated inquiries. Kit is still under development, but the flow-based paradigm offered an implementation for the solution models suggested by the participants.

5.2 Contribution

I was able to come up with some useful observations about non-programmers' ability to create data-related models in a visual flow-based environment. These observations were accompanied with a list of recommendations to be considered when designing environment similar to the one in this study. These recommendations included enhancing the usability by considering the variation of the users' needs and provide them with appropriate help facilities. Further recommendations included enhancing the visibility, discoverability, reliability, and scalability of the environment. Last but not least, the design of the environment should consider the standards and metaphors used in other popular tools such as spreadsheets (more details can be found in Section 4.5).

5.3 Limitations

While overall I believe the findings are sound, the following limitations need to be considered:

1. the study was performed on a prototype of Kit. Therefore, findings are applicable to the tested version of Kit and further studies need to be conducted on the fully-developed environment.

2. Participants were divided into three different groups which made it hard to deal with them as a single group of 21 participants. Such a limited number of participants made it difficult to draw quantitative conclusions.

3. Some of the results came out of non-validated and non-standard surveys.

4. The recognition test was biased by the ambiguity of the scenarios.

5. The Investigator had limited intervention at some stages of the study, which could affect the performance of the participants.

6. Some phases of the study were not audio recorded, which might affect the accuracy of the recollections made of those phases.

5.4 Future Work

1. As Kit is evolving, the chance of having new design issues always exists. I would like to track the usability evolution through the different cycles of Kit's agile development. More usability evaluation and testing techniques with larger groups of participants can be conducted[1] to provide better support to the target audience.

2. Enhance the design of the environment by implementing some of the recommendations provided in sections 4.2.3 and 4.4.1.

[1] An A/B testing for the user interface could be a good task to start with.

Appendices

Appendix A

Consent Form

Ryerson University
Consent Agreement
(A comparative study of UI features for naive programmers.)

You are being asked to participate in a research study as a part of a thesis work of a graduate student (Alia Shubair). Before you give your consent to be a volunteer, it is important that you read the following information and ask as many questions as necessary to be sure you understand what you will be asked to do.

Investigators:

Alia Shubair (Primary Investigator)
Graduate Student
Computer Science Department
Ryerson University

Dr.Dave Mason
Professor
Computer Science Department
Ryerson University

Purpose of the Study:

The purpose of this study is to find out what interesting problems professionals want to solve using large data tables, which data tables they use to do so, how they use those tables, and what features of our tool would assist them in this endeavour. We are recruiting up to 40 adult participants who will be either faculty members, students, or professionals in a specific discipline.

Description of the Study:

We will be holding 3 to 6 focus group sessions in a meeting room on the Ryerson campus, each one lasting 1 to 2 hours. In each session a group of 3 to 6 participants from a specific discipline will be brainstorming ideas related to the purpose of the study. You will only attend one focus group. During these brainstorming sessions, we will probably use smartboards and whiteboards to brainstorm and projectors to display publicly available data tables relevant in their discipline (e.g. from Statistics Canada).

What is Experimental in this Study:

A new tool "Kit" is being developed to support non-programmers who want to manipulate and extract information from large amounts of data. We are trying to figure out how to design the most usable interface for this population.

Risks or Discomforts: You will be located in a meeting room to share and discuss your ideas in a very friendly environment. The only discomfort (if any) might be boredom or lack of interest. The sessions are not tightly structured: you will be free to move around and we will be providing refreshments and regular breaks.

Benefits of the Study:

We cannot guarantee any direct benefit to individual participants. On the other hand, the results of this study will be redirected to form guidelines for another phase of experiment. After collecting ideas and suggestions from different Focus Groups, actual experiments would be held on naive users to try different combination of UI features. Then the most usable design would be picked for our "Kit".

Confidentiality:

The problems discussed during the sessions will be generic/domain specific work problems. No personal relationships or history will be solicited and all personal information are kept privately with the study researcher.

During the sessions a video camera will be fixed to photograph/tape the board only to document your suggestions and ideas. You will be captured in the tape only if you choose to explain something there by yourself. Only the three investigators will keep copies of these sessions. Nevertheless the sessions are photographed and Video/Audio taped for research purposes only. Each will keep a copy saved on their personal work computers, but no other copies will be made or kept.

Clips from these sessions may also be shown to other members of the research team to help them understand users requirements. However, they will not be otherwise disseminated. The privacy of the participants shown in any clips will still be respected if requested by blurring identifying features of them and pseudonyms will be used to obscure their identities if asked to by checking the box at the end of this consent form.

Although the researchers will maintain confidentiality, we cannot guarantee this on behalf of other participants in the focus group.

Voluntary Nature of Participation:

Your participation in this study is voluntary. Your choice of whether or not to participate will not influence your future relations with Ryerson. If you decide to participate, you are free to withdraw your consent and to stop your participation at any time without penalty or loss of benefits to which you are allowed.

At any particular point in the study, you may refuse to answer any particular question or stop participation altogether.

Questions about the Study: If you have any questions about the research now, please ask. If you have questions later about the research, you may contact.

Alia Shubair (Graduate Student and Primary Researcher)
alia.shubair@ryerson.ca
Dr.Dave Mason
Tel: (416)979-5000 (x7061)

If you have questions regarding your rights as a human subject and participant in this study, you may contact the Ryerson University Research Ethics Board for information.

Research Ethics Board
c/o Office of the Vice President, Research and Innovation
Ryerson University
350 Victoria Street
Toronto, ON M5B 2K3
416-979-5042

Agreement:

Your signature below indicates that you have read the information in this agreement and have had a chance to ask any questions you have about the study. Your signature also indicates that you agree to be in the study and have been told that you can change your mind and withdraw your consent to participate at any time. You have been given a copy of this agreement.

You have been told that by signing this consent agreement you are not giving up any of your legal rights.

_____
Name of Participant (please print)

_____ _____
Signature of Participant Date

_____ _____
Signature of Investigator Date

□ I understand that the focus group will be video and audio recorded and the recordings will only be shared with the research team.

_____
Signature of Participant

Appendix B

Usability Survey

B.1 Personal Information

1. Age Category:
    - <25
    - 25 - 35
    - 36 - 45
    - 46 - 55
    - >56
2. Gender:
    - Male
    - Female
3. Highest level of education:
    - High School Graduate
    - Some Collage / Associate Degree
    - Bachelor's Degree
    - Graduate Degree

4. Do you have any programming experience:
    - Not at all
    - A little
    - Yes

If you did not choose the first option in the previous question, please list the languages you tried before:

B.2 Kit Environment

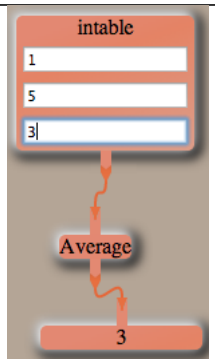
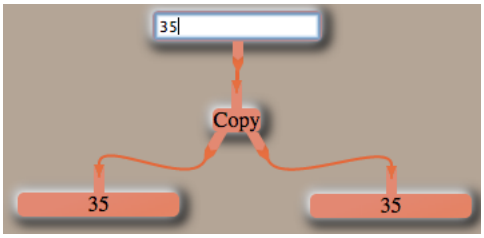
Based on the previous brief orientation of our KIT, please answer the following questions:

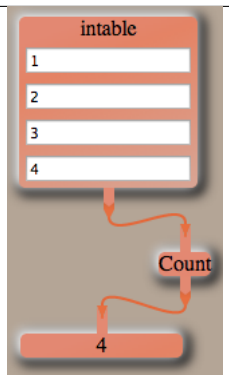
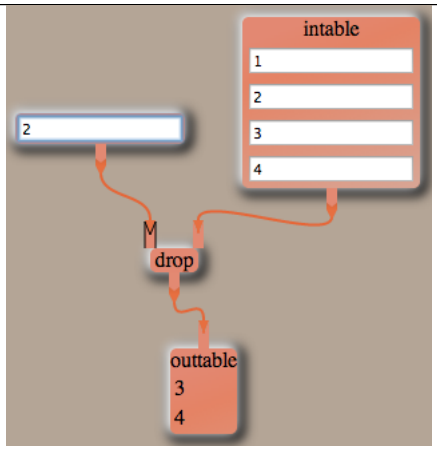
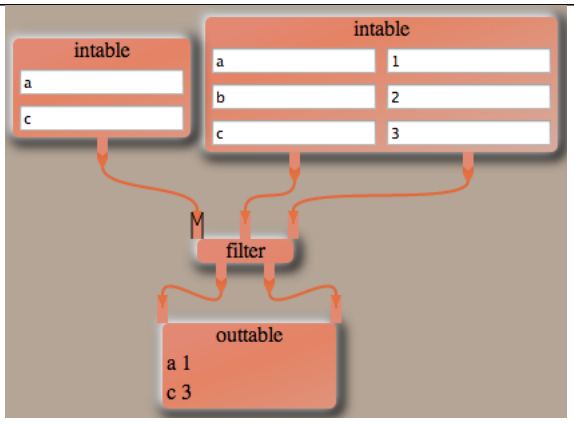
1. On a scale from 1 to 5 How understandable did you find the orientation session?
   Least 1 2 3 4 5 Most
2. On a scale from 1 to 5, how easy do you think this tool would be to learn?
   Least 1 2 3 4 5 Most
3. How effective and realistic do you think this KIT is going to be to handle data-related problems?
   Least 1 2 3 4 5 Most
4. How important do you think it is for someone to make an environment that would help you manipulate a broad range of data?
   Least 1 2 3 4 5 Most
5. Please state your degree of interest in learning how KIT works:
   Least 1 2 3 4 5 Most
6. Has lack of programming experience been an issue in following with the orientation?
   Yes No

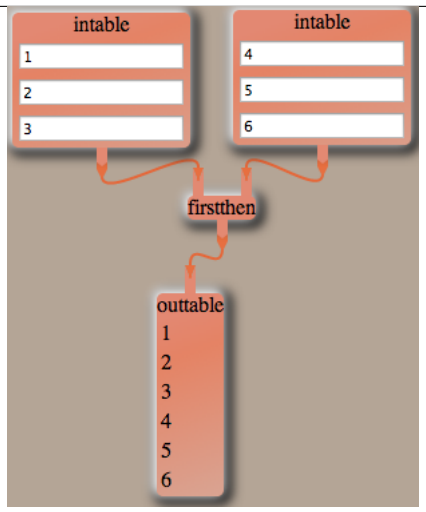
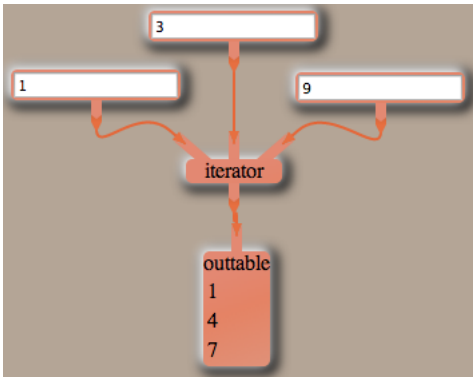
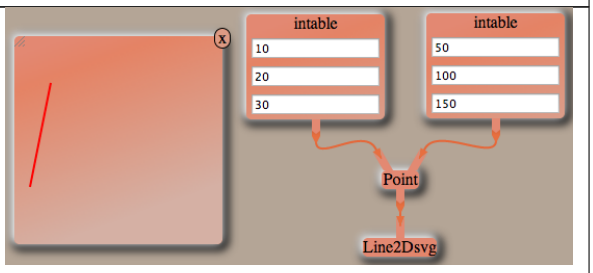
Other Comments/Suggestions:

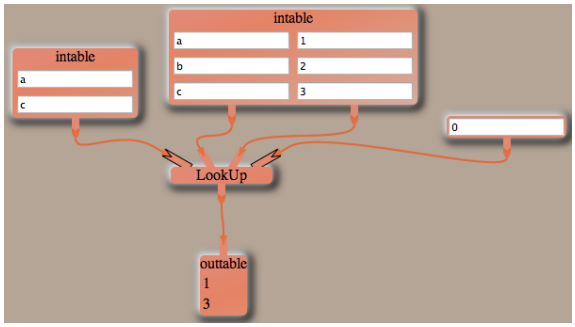
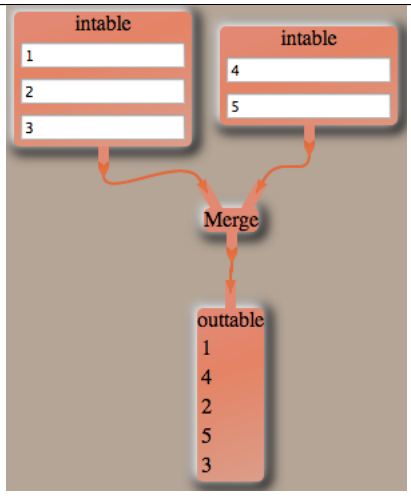
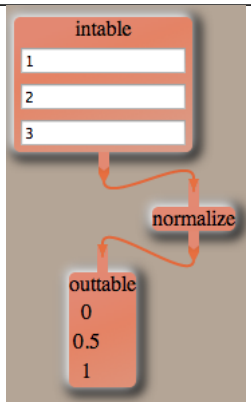
Appendix C

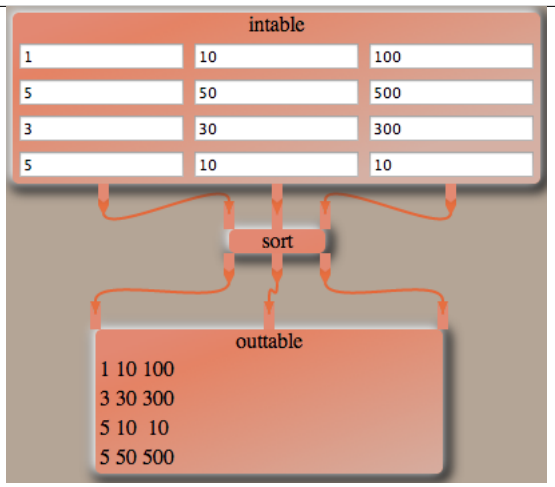
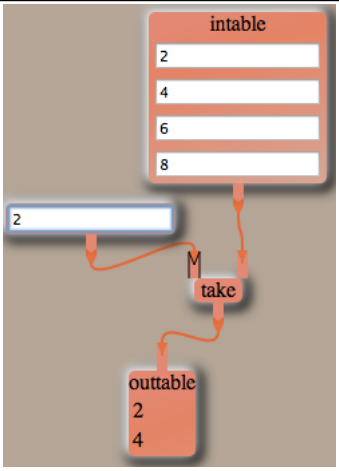
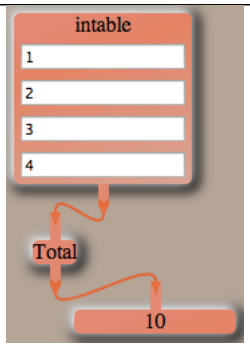
Kit Manual

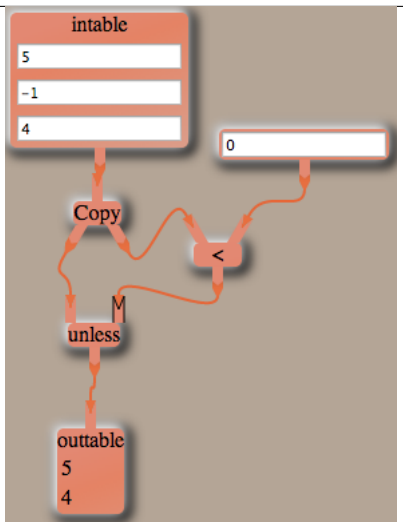
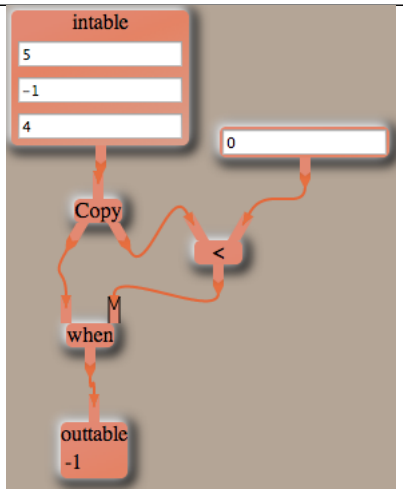
Block's Name	Functionality	Example
Average	Average block calculates the average of an input stream of values and outputs the result as a single numeric value.	
Copy	Copy block will create another copy of any input wether it is a single value or a stream of values. You can add as many output streams as you need with a single copy block to have a multiple duplications of the original value.	

Count	Count block calculates the number of elements in an input stream and outputs the result as a single numeric value.	
Drop	Drop block has two inTabs. The first inTab is a control tab that saves the desired number of elements to be dropped out of the second inTab which is a stream of values. The resulted output will be an output stream that contains the original input stream after dropping the number of elements specified in the control tab.	
Drop	Filter block filters input streams based on matching the values given in the first control tab. The output is valid only if the first attribute's value existed in the control filter stream.	

FirstThen	FirstThen block merges two input streams by outputting all the elements of the first input stream followed by the elements of the second input stream. Contents of the two inputs will not be mixed up in the resulted output stream.	
Iterator	Iterator block generates a sequence of numbers based on the following inTabs rules: First input determines the start point of the sequence, the second determines the pace length, and the third determines the maximum limit to reach. The output will be presented as a stream of the resulted elements.	
Line2Dsvg	Line2Dsvg block is an output block that creates the graphical representation of its input data. A small window will pop-up showing the graph of the corresponding sequence of points.	

LookUp	LookUp block matches two keys from two different tables. The first inTab is a control tab and it represents the required keys. If those values were found in the second inTab they will be outputted with their corresponding rest of values, if not: a default value from the last control inTab will be assigned to the required non-existing key.	
Merge	Merge block combines the values of two input streams in one output stream considering mixing them up equally with respect to their order.	
Normalize	Normalize block normalizes the value of an input table to range from 0 to 1. The output will be a sequence of normalized values.	
Point	Point block combines each x-value, with a y-value to create a 2D point that can be represented in a graph.	Refer to Line2Dsvg example.

Sort	Sort block sorts the values with respect to the first inTab, then the second, etc. Each sort block is linked to at least one inTab's stream of values, and one sorted outTab's stream.	
Take	Take block has two inTabs. The first inTab is a control tab that saves the desired number of elements to be taken from the second inTab which is a sequence of elements. The resulted output will be an output stream that contains the original input table after taking the number of elements specified in the control tab only.	
Total	Total block calculates the sum of the input elements. Each total block has a sequence of elements as an input, and a single numeric value represents the summation of inputs' elements as an output.	

Unless	Unless block is a conditional block. The output will not be generated unless the input condition is false.	
When	When block is a conditional block, The output will be generated only if the input condition is satisfied.	

Appendix D

Kit Code

This chapter contains the implementation of the blocks which has been suggested through the study. The code is written in Smalltalk language.

D.1 Average Block

```
DataBlock subclass: #AverageBlock
        instanceVariableNames: 'collecting counter total'
        package: 'Kit'!


initTabs
        self initTabsIn: 1 out: 1
!

populate
        self name: 'Average'.
        super populate.
!

thunkReset
        collecting := true.
        counter:=0.
        total:=0
!
```

```
thunk

        collecting ifTrue: [
                self inputsAtEnd ifTrue: [
                        collecting := false.
                        ^ self thunk
                ].
                counter:=counter+1.
                total:= total+ (inputs at:1) get.
                (inputs at:1) isChangeable.
                ^ true
        ] ifFalse: [
                (counter==0) ifFalse:[^self output:(total/
                   ↪ counter) changeable:false
                ] ifTrue: [ ^ self eof]
        ]
```

D.2 Count Block

```
DataBlock subclass: #CountBlock
        instanceVariableNames: 'collecting counter'
        package: 'Kit'!


initTabs
        self initTabsIn: 1 out: 1
!

populate
        self name: 'Count'.
        super populate.
!

thunk
        collecting ifTrue: [
                self inputsAtEnd ifTrue: [
                        collecting := false.
                        ^ self thunk
                ].
                counter:=counter+1.
                (inputs at:1) get.
                (inputs at:1) isChangeable.
                ^ true
        ] ifFalse: [
                ^self output:counter changeable:false.
        ]
!

thunkReset
        collecting := true.
        counter:=0
```

D.3 Filter Block

```
StreamBlock subclass: #FilterBlock
        instanceVariableNames: 'collecting array'
        package: 'Kit'!


initTabs
        self initTabsIn: 2 out: 1.
         inTabs first makeControlTab.
        self thunkReset
!

!

thunkReady
        |filterTable|
        self thunkOutputReady ifFalse: [ ^ false].
        filterTable:=inputs at:1.
        collecting ifTrue: [
                ^ filterTable isAvailable
                ].
        2 to: (inputs size) do: [: i | (inputs at: i)
           ↪ isAvailable ifFalse: [^ false] ].
        ^ true
!

thunkReset
        array:= OrderedCollection new.
        collecting:=true
!

thunk
        collecting ifTrue:[     | filterTable |
                filterTable:=inputs at:1.
                (filterTable atEnd) ifTrue: [
```

```
                collecting := false.
                ^ self thunk
        ].
        array addLast: (filterTable get).
        filterTable isChangeable ifFalse: [
                collecting := false.
                ^ self thunk
        ].
        ^ true
] ifFalse: [    | filteredValues |
        filteredValues:=inputs at:2.
        filteredValues atEnd ifTrue: [^ self eof].
        (array includes: filteredValues get) ifTrue: [
           ↪ | allChangeable |
                allChangeable := false.
                inputs withIndexDo: [: pipe : i |
                        i > 1 ifTrue: [   | changeable
                           ↪ value |
                                pipe atEnd ifTrue: [^
                                   ↪ self eof].
                                value := pipe get.
                                changeable := pipe
                                   ↪ isChangeable.
                                allChangeable :=
                                   ↪ allChangeable |
                                   ↪ changeable.
                                (outputs at: i-1) put:
                                   ↪ value changeable:
                                   ↪  changeable
                ]].
                ^ allChangeable
        ].
        inputs withIndexDo: [: pipe : i |
                i > 1 ifTrue: [
                        pipe atEnd ifTrue: [^ self eof
                           ↪ ].
```

```
                                pipe get.
                                pipe isChangeable ]].
            ^ true
    ]
```

D.4 LookUp Block

```
DataBlock subclass: #LookUpBlock
        instanceVariableNames: 'arrayKey arrayValue collecting'
        package: 'Kit'!


initTabs
        self initTabsIn: 4 out: 1.
        "Two tables of the same attribute e.g:postal code,
                another attribute to be looked up and matched e
                   ↪ .g:avg. income,
                and a default value to be assigned if none
                   ↪ existed
                The output is the value only without the key"
                 inTabs first makeControlTab.
                 inTabs last makeControlTab.
        self thunkReset
!

populate
        self name: 'LookUp'.
        super populate.
!

thunk

        collecting ifTrue:[
                |currentValue currentKey|
                (inputs at:2) atEnd ifTrue:[
                        collecting:=false.
                        ^self thunk
                        ].
                currentKey:=(inputs at:2) get.
                currentValue:=(inputs at:3) get.
                arrayKey addLast:currentKey.
```

```
                arrayValue addLast:currentValue.
                ((inputs at:2) isChangeable) & ((inputs at:3)
                    ↪ isChangeable) ifFalse:[
                        collecting:=false.
                        ^self thunk
                        ].
                        ^true
                ]
        ifFalse: [
                (inputs at:1) atEnd ifTrue: [^self eof].
                (arrayKey includes: ((inputs at:1) get)) ifTrue
                    ↪ : [
                        arrayKey withIndexDo:[ :value :i |(
                            ↪ value == ((inputs at:1) get ))
                            ↪ ifTrue:[ self output:(arrayValue
                            ↪ at:i)  changeable:true]].
                        (inputs at:1) isChangeable.
                        ] ifFalse:[
                                self output:((inputs at:4) get)
                                    ↪  changeable:true .
                                (inputs at:1) isChangeable.
                                ].
                ^true
        ]
!

thunkReset
        arrayKey:=OrderedCollection new.
        arrayValue:=OrderedCollection new.
        collecting:=true.
```

D.5 Normalize Block

```
StreamBlock subclass: #NormalizeBlock
        instanceVariableNames: 'collecting array'
        package: 'Kit'!


array
        ^ array
!

array: a
        array := a
!

initTabs
        self initTabsIn: 1 out: 1
!

normalize
        |maxVal minVal|

        array isEmpty ifFalse: [
                |diff|
                maxVal:= array inject:(array at:1) into: [:mxV
                    ↪ :each| mxV max:each].
                minVal:= array inject:(array at:1) into: [:mnV
                    ↪ :each| mnV min:each].
                diff:=maxVal-minVal.
                array:=array collect:[:each| (each-minVal)/diff
                    ↪ ]
        ]
!
```

```
thunkReset
        array := OrderedCollection new.
        collecting := true
!

thunk
        collecting ifTrue: [
                self inputsAtEnd ifTrue: [
                        self normalize.
                        collecting := false.
                        ^ self thunk
                ].
                array addLast: ((inputs at:1) getNumber).
                (inputs at:1) isChangeable.
                ^ true
        ] ifFalse: [
                array isEmpty ifTrue: [^ self eof].
                ^ self output: array removeFirst changeable:
                   ↪ true
        ]
```

D.6 Total Block

```
DataBlock subclass: #TotalBlock
        instanceVariableNames: 'collecting totalVal'
        package: 'Kit'!


initTabs
        self initTabsIn: 1 out: 1
!

populate
        self name: 'Total'.
        super populate.
!

thunk

        collecting ifTrue: [
                self inputsAtEnd ifTrue: [
                        collecting := false.
                        ^ self thunk
                ].
                totalVal:=totalVal+((inputs at:1) getNumber).
                (inputs at:1) isChangeable.
                ^ true
        ] ifFalse: [
                ^self output:totalVal changeable:false.
        ]
!

thunkReset
        collecting := true.
        totalVal:=0
```

Appendix E

Validation Test

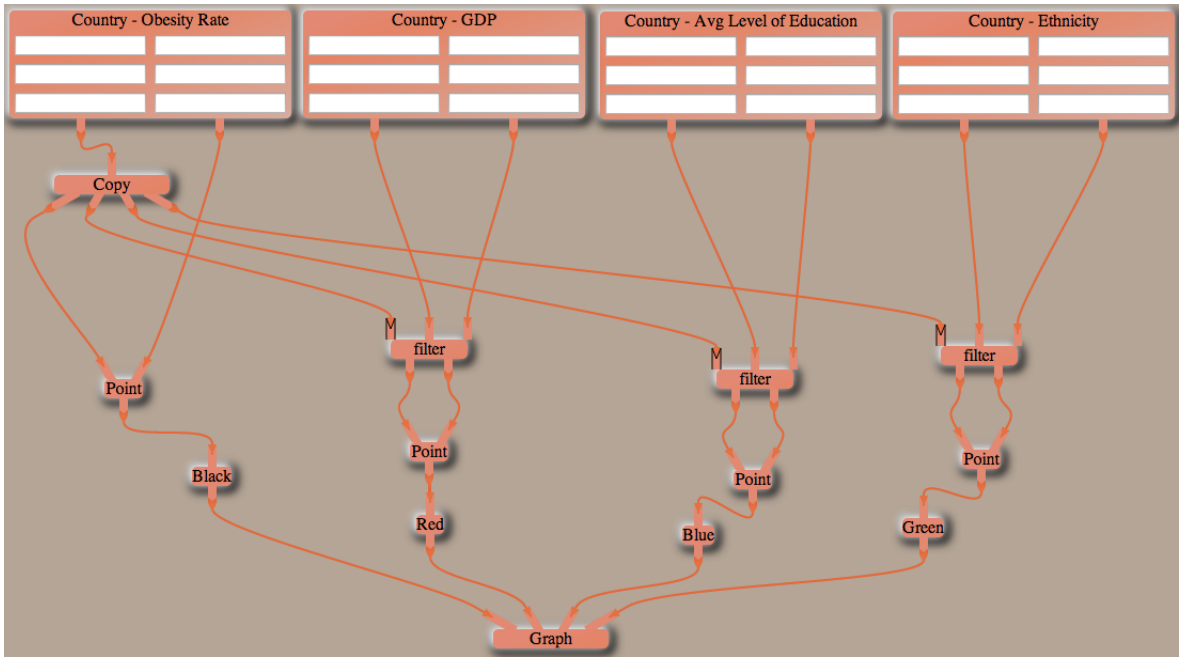
E.1 Model Validation

This validation test is built to measure the usability of Kit. It has multiple problem scenarios that have been already presented in the earlier discussion session. What we need is for you to choose the problem scenario closest to the Kit solution shown. Please take a look at the attached Manual if you find a block is hard to understand. Kit's Manual was built to explain each block's functionality by example.

Please provide all answers on the supplied answer sheet.

- First Kit Model:

  Emily is a Nutrition student. She is working on a study to find the link between the obesity rate in different countries with the following factors: GDP, average educational level, and ethnicity.

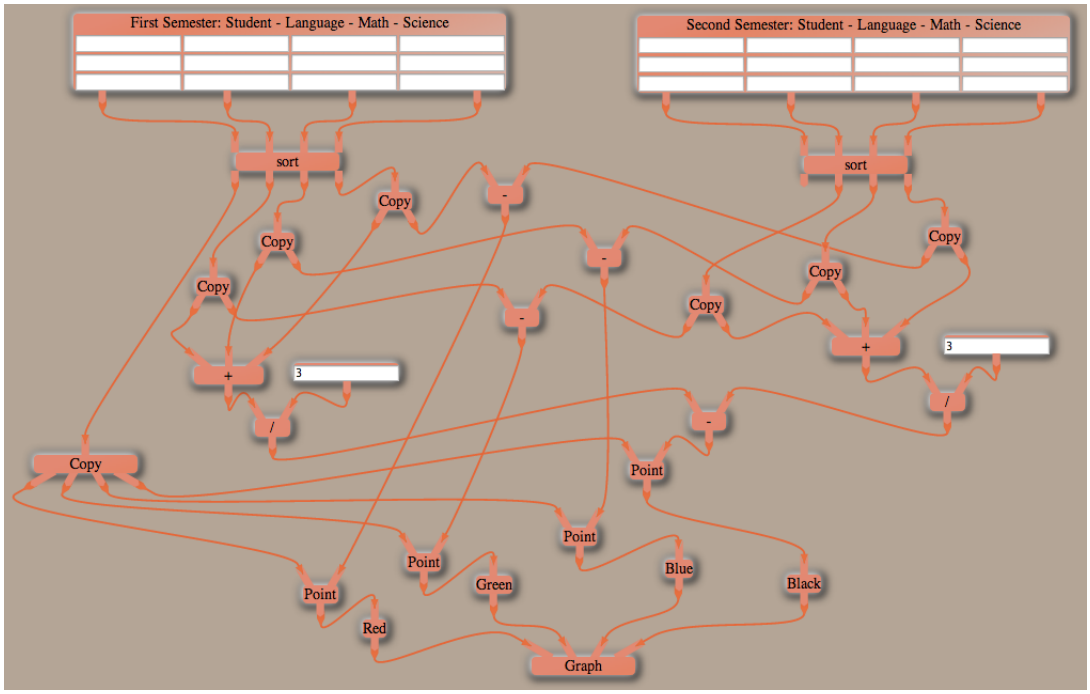


  In your opinion, which one of the following scenarios matches that diagram:

  - Scenario 1: She decides to plot lines showing the ratio of each of the factors divided by obesity level on a graph by country.
  - Scenario 2: She decides to plot lines showing the obesity level and each of the factors on a graph by country.
  - Scenario 3: She decides to produce a scatter-plot with obesity by country, with each datapoint coloured by the combination of factors.

- Second Kit Model:

  Lily is a teacher who wants to compare the students' performance in two different semesters to compare their progress in three different subjects.

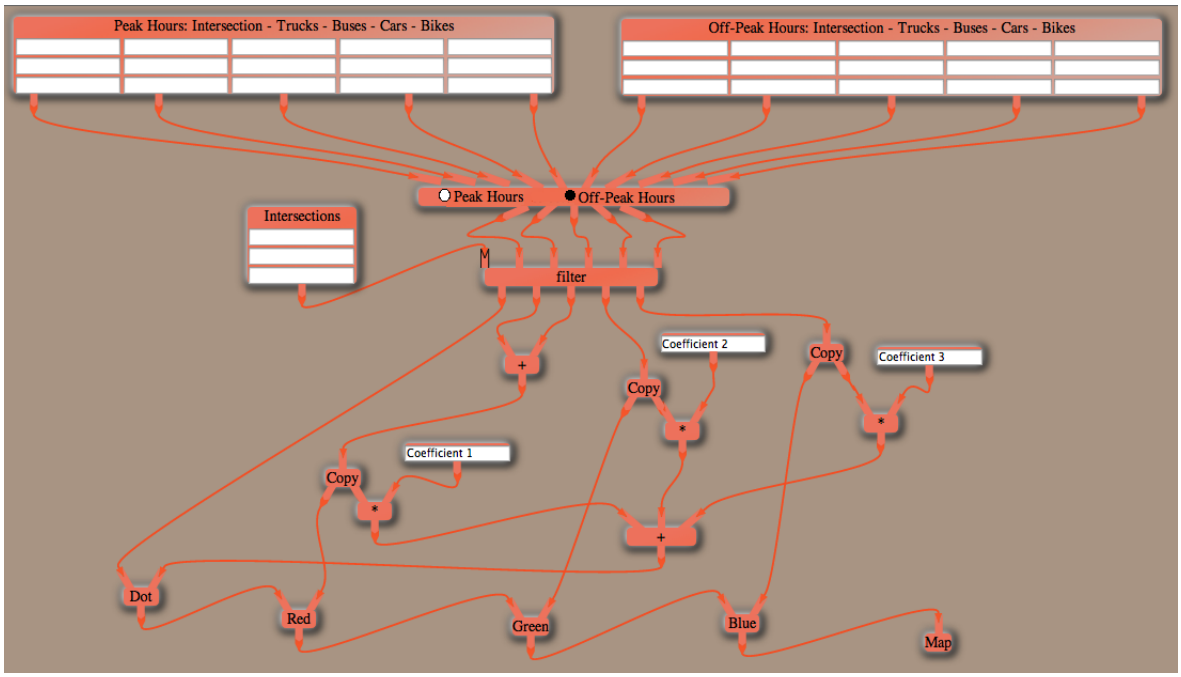


In your opinion, which one of the following scenarios matches that diagram:

- Scenario 1: She decides to plot the average for all students by subject, for each term.
- Scenario 2: She decides to produce a scatter-plot with average grade in first term by average grade in second term, per student.
- Scenario 3: She decides to plot the differences between the two semesters by student, showing the differences in each subject and the average of the three.

- Third Kit Model:

  William is a traffic engineer who works on road congestion. William is trying to find out the relationship between each congested intersection and different types of vehicles (i.e cars, buses, trucks, etc.) during different periods of time.



In your opinion, which one of the following scenarios matches that diagram:

  - Scenario 1: He decides to put a toggle so he can switch between the peak-hours/non-peak-hours data, and for each graph plot a dot on a map for each intersection that is sized by the weighted traffic volume, and coloured by the truck+bus / cars / bikes ratio.
  - Scenario 2: He decides to put a toggle so he can switch between the peak-hours/non-peak-hours data, and for each graph plot a dot on a map for each intersection that is sized by the weighted traffic volume, and coloured by the truck+bus / cars / bikes ratio. He also plots an extra point that is sized by the overall traffic weight.
  - Scenario 3: He decides to plot a graph for each of peak-hours and non-peak-hours data, and for each graph plot a dot on a map for each intersection that is sized by the weighted traffic volume, and coloured by the truck+bus / cars / bikes ratio.

E.2 Interactive Validation

1. The first problem scenario:
   You have a 4*4 input stream. You need to calculate the sum of the first three values of each row, and the average of the third and fourth value. The results from both calculations will be output in a 2*4 output stream.

2. The second problem scenario:
   John is a sale manager who works in a department store. He wants to investigate the sales' performance throughout the past five years. Assuming that he has the sales history table for the last 5 years that includes: Year, Dept1, Dept2,Dept3, and Dept4 how would he retrieve the following information:

   (a) Avg. of sales for all the departments each year.

   (b) Avg. of sales for the last three years for each department.

   (c) Output the last year's sales only.

References

[1] A. C. Acock. *A Gentle Introduction to Stata*. Stata Press, second edition, Sep 2008.

[2] W. Albert and T. Tullis. *Measuring the user experience: collecting, analyzing, and presenting usability metrics*. Newnes, 2013.

[3] Apple. Siri. URL `http://www.apple.com/ca/ios/siri/`.

[4] R. Bellamy and D. Gilmore. Programming plans: Internal or external structures. *Lines of thinking: Reflections on the psychology of thought*, 2:59–72, 1990.

[5] S. Bhatia, T. Lilley, D. S. McCrickard, and P. Kienzle. Emerging from the MIST: A connector tool for supporting programming by non-programmers. 2010.

[6] S. Bhatia, D. S. McCrickard, T. Lilley, C. North, and P. Kienzle. Scientists in the MIST: Simplifying interface design for end users. 2006.

[7] A. Blackwell. What is programming? In *14th workshop of the Psychology of Programming Interest Group*, pages 204–218, 2002.

[8] A. Blomquist and M. Arvola. Personas in action: Ethnography in an interaction design team. In *Proceedings of the Second Nordic Conference on Human-computer Interaction*, NordiCHI '02, pages 197–200. ACM, New York, NY, USA, 2002. ISBN 1-58113-616-1. URL `http://doi.acm.org/10.1145/572020.572044`.

[9] A. Borning. The programming language aspects of thinglab, a constraint-oriented simulation laboratory. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 3(4):353–387, 1981.

[10] M. Boshernitsan and M. S. Downes. *Visual programming languages: A survey*. Citeseer, 2004.

[11] D. Boyd and K. Crawford. Six provocations for big data. *A Decade in Internet Time: Symposium on the Dynamics of the Internet and Society*, Sept. 2011.

[12] V. Braun and V. Clarke. Using thematic analysis in psychology. *Qualitative research in psychology*, 3(2):77–101, 2006.

[13] C. Brom, J. Gemrot, M. Bıda, O. Burkert, S. J. Partington, and J. J. Bryson. POSH tools for game agent development by students and non-programmers. In *The Nineth International Computer Games Conference: AI, Mobile, Educational and Serious Games*, pages 126–133. University of Bath, 2006.

[14] D. R. Brooks. *An Introduction to HTML and JavaScript: for Scientists and Engineers (p. 1)*. Springer, Jun 30 2007.

[15] M. M. Burnett. Seven programming language issues. In *Visual object-oriented programming*, pages 161–181. Manning Publications Co., 1995.

[16] M. C. Carlisle, T. A. Wilson, J. W. Humphries, and S. M. Hadfield. Raptor: a visual programming environment for teaching algorithmic problem solving. In *ACM SIGCSE Bulletin*, volume 37, pages 176–180. ACM, 2005.

[17] J. M. Carroll. Five reasons for scenario-based design. *Interacting with computers*, 13(1):43–60, 2000.

[18] A. L.-G. Chava Frankfort-Nachmias. *Social Statistics for Diverse Society (p. 150)*. Pine Forge Press, July 2005.

[19] W. Citrin, M. Doherty, and B. Zorn. Control constructs in a completely visual imperative programming language. Technical report, University of Colorado, 1993.

[20] W. Citrin, M. Doherty, and B. Zorn. The design of a completely visual object-oriented programming language. *Visual Object-Oriented Programming: Concepts and Environments. Prentice-Hall, New York*, 1995.

[21] A. Cooper. The origin of personas. *INNOVATION-MCLEAN THEN DULLES VIRGINIA-*, 23(1):26–29, 2004.

[22] P. T. Cox and T. Pietryzkowsky. *Using a pictorial representation to combine dataflow and object-orientation in a languageindependent programming mechanism*. Visual Programming Environments: Paradigms and Systems. Los Alamitos, CA: IEEE Computer Society Press, 1990.

[23] S. P. Davies. Externalising information during coding activities: Effects of expertise, environment and task. *ESP*, 93(744):42–61, 1993.

[24] H. E. Dunsmore. Designing an interactive facility for non-programmers. In *Proceedings of the ACM 1980 Annual Conference*, ACM '80, pages 475–483. ACM, New York, NY, USA, 1980. ISBN 0-89791-028-1. URL http://doi.acm.org/10.1145/800176.810003.

[25] A. Eckerdal, M. Thuné, and A. Berglund. What does it take to learn 'programming thinking'? In *Proceedings of the First International Workshop on Computing Education Research*, ICER '05, pages 135–142. ACM, New York, NY, USA, 2005. ISBN 1-59593-043-4. URL http://doi.acm.org/10.1145/1089786.1089799.

[26] G. Fischer. User modeling in human–computer interaction. *User modeling and user-adapted interaction*, 11(1-2):65–86, 2001.

[27] D. Fisher, R. DeLine, M. Czerwinski, and S. Drucker. Interactions with big data analytics. *Interactions*, 19(3):50–59, May 2012. ISSN 1072-5520. URL http://doi.acm.org/10.1145/2168931.2168943.

[28] V. Fix, S. Wiedenbeck, and J. Scholtz. Mental representations of programs by novices and experts. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*, pages 74–79. ACM, 1993.

[29] W. Fizner and L. Gould. Rehearsal world: Programming by rehearsal, 1993.

[30] Google. URL https://www.google.com.

[31] J. D. Gould and C. Lewis. Designing for usability: key principles and what designers think. *Communications of the ACM*, 28(3):300–311, 1985.

[32] T. Green and M. Petre. When visual programs are harder to read than textual programs. In *In*, pages 167–180. Citeseer, 1992.

[33] T. Green, M. Sime, and M. Fitter. The art of notation. *Computing skills and the user interface*, pages 221–251, 1981.

[34] T. R. G. Green, R. K. E. Bellamy, and M. Parker. Parsing and gnisrap: A model of device use. In G. M. Olson, S. Sheppard, and E. Soloway, editors, *Empirical studies of programmers: Second workshop*, pages 132–146. Ablex Publishing Corp., Norwood, NJ, USA, 1987. ISBN 0-89391-461-4. URL http://dl.acm.org/citation.cfm?id=54968.54977.

[35] T. R. G. Green and M. Petre. Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *Journal of Visual Languages & Computing*, 7(2):131–174, 1996.

[36] P. Gross and C. Kelleher. Non-programmers identifying functionality in unfamiliar code: strategies and barriers. *Journal of Visual Languages & Computing*, 21 (5):263–276, 2010.

[37] P. Gross, C. Kelleher, and J. Yang. An investigation of non-programmers' performance with tools to support output localization. In *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on*, pages 55–58, Sept 2011. ISSN 1943-6092.

[38] P. Gross, J. Yang, and C. Kelleher. Dinah: An interface to assist non-programmers with selecting program code causing graphical output. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 3397–3400. ACM, New York, NY, USA, 2011. ISBN 978-1-4503-0228-9. URL http://doi.acm.org/10.1145/1978942.1979448.

[39] InternationalOrganizationforStandardization. ISO 9241-10:1996. URL ISO9241-10:1996.

[40] A. Jacobs. The pathologies of big data. *Commun. ACM*, 52(8):36–44, Aug. 2009. ISSN 0001-0782. URL http://doi.acm.org/10.1145/1536616.1536632.

[41] B. E. John and D. E. Kieras. The GOMS family of user interface analysis techniques: Comparison and contrast. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 3(4):320–351, 1996.

[42] E. Keppel and D. Kropp. Interactive programming by end-users. In *Proceedings of the 1977 Annual Conference*, ACM '77, pages 301–307. ACM, New York, NY, USA, 1977. ISBN 978-1-4503-2308-6. URL http://doi.acm.org/10.1145/800179.810219.

[43] K. R. Koedinger, V. Aleven, N. Heffernan, B. McLaren, and M. Hockenberry. Opening the door to non-programmers: Authoring intelligent tutor behavior by demonstration. In *Intelligent Tutoring Systems*, pages 162–174. Springer, 2004.

[44] J. Kontio, L. Lehtola, and J. Bragge. Using the focus group method in software engineering: Obtaining practitioner and user experiences. In *Proceedings of the*

2004 International Symposium on Empirical Software Engineering, ISESE '04, pages 271–280. IEEE Computer Society, Washington, DC, USA, 2004. ISBN 0-7695-2165-7. URL http://dx.doi.org/10.1109/ISESE.2004.35.

[45] A. Kriegel. *Discovering SQL: A Hands-On Guide for Beginners*. John Wiley and Sons, Mar 2011.

[46] C. Lewis and G. Olson. Can principles of cognition lower the barriers to programming? In *Empirical Studies of Programmers: Second Workshop*, pages 248–263. Ablex Publishing Corp., 1987.

[47] S. Lohr. The age of big data. *New York Times*, 11, 2012.

[48] P. Lomax. *VB and VBA in a Nutshell: The Language*. O'Reilly Media, Inc., 1998.

[49] A. M. Lund. Measuring usability with the use questionnaire. *Usability interface*, 8(2):3–6, 2001.

[50] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers. Big data: The next frontier for innovation, competition, and productivity. *Insights and Publications - McKinsey Global Institute*, 2011. URL http://www.mckinsey.com/insights/business_technology/big_data_the_next_frontier_for_innovation.

[51] J. L. Marin. Data driven journalism in spain with pro bono público. URL http://www.openeconomy.net/2010/12/data-driven-journalism-in-spain-with.html.

[52] D. Mason. Flexible structures for end-user programming. In *Proceedings of the 3rd International Workshop on Free Composition*, FREECO '12, pages 9–11. ACM, New York, NY, USA, 2012. ISBN 978-1-4503-1635-4. URL http://doi.acm.org/10.1145/2414716.2414720.

[53] D. Mason. Data programming for non-programmers. *Procedia Computer Science*, 21:68–74, 2013.

[54] D. Mason. Programming for the rest of us - why?, Aug. 2014. URL http://programmingfortherestofus.com/Others/why.html.

[55] MathWorks. MATLAB - the language of technical computing. URL http://www.mathworks.com/products/matlab/.

[56] S. Matwin and T. Pietrzykowski. Prograph: a preliminary report. *Computer Languages*, 10(2):91–126, 1985.

[57] L. A. Miller. Programming by non-programmers. *International Journal of Man-Machine Studies*, 6(2):237–260, March 1974.

[58] J. Morrison. *Flow-Based Programming: A New Approach to Application Development*. van Nostrand Reinhold, first edition, 1994.

[59] J. P. Morrison. Flow-based programming. URL `http://www.jpaulmorrison.com/fbp/`.

[60] J. P. Morrison. Patterns in flow-based programming. URL `http://www.jpaulmorrison.com/fbp/morrison_2005.htm`.

[61] J. P. Morrison. Flow-based programming. *Journal of Application Developers' News*, 1994.

[62] P. Morrison. Flow-based programming, Apr. 2012. URL `http://flowbasedprogramming.wordpress.com/article/flow-based-programming/`.

[63] B. A. Myers. Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing*, 1(1):97–123, 1990.

[64] B. A. Myers, J. F. Pane, and A. Ko. Natural programming languages and environments. *Commun. ACM*, 47(9):47–52, Sept. 2004. ISSN 0001-0782. URL `http://doi.acm.org/10.1145/1015864.1015888`.

[65] J. V. Nickerson. *Visual programming*. PhD thesis, New York University, 1994.

[66] J. Nielsen. The usability engineering life cycle. *Computer*, 25(3):12–22, 1992.

[67] J. Nielsen. 10 usability heuristics for user interface design. *Nielsen Norman Group: Evidence-Based User Experience Research, Training, and Consulting*, 1995.

[68] J. Nielsen. Usability 101: Introduction to usability, 2003. URL `http://www.nngroup.com/articles/usability-101-introduction-to-usability`.

[69] J. Nielsen and R. Molich. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '90, pages

249–256. ACM, New York, NY, USA, 1990. ISBN 0-201-50932-6. URL http://doi.acm.org/10.1145/97243.97281.

[70] D. A. Norman and J. Nielsen. Gestural interfaces: a step backward in usability. *interactions*, 17(5):46–49, 2010.

[71] J. Pane and B. Myers. Usability issues in the design of novice programming systems. 1996.

[72] J. F. Pane, C. Ratanamahatana, B. A. Myers, et al. Studying the language and structure in non-programmers' solutions to programming problems. *International Journal of Human-Computer Studies*, 54(2):237–264, 2001.

[73] J. M. Paul Wilton. *Beginning JavaScript*. John Wiley and Sons, Jan 2011.

[74] M. Petre and T. R. G. Green. Learning to read graphics: Some evidence that'seeing'an information display is an acquired skill. *Journal of Visual Languages & Computing*, 4(1):55–70, 1993.

[75] P. G. Polson, C. Lewis, J. Rieman, and C. Wharton. Cognitive walkthroughs: a method for theory-based evaluation of user interfaces. *International Journal of man-machine studies*, 36(5):741–773, 1992.

[76] I. Pouncey and R. York. *Beginning CSS: Cascading Style Sheets for Web Design*. John Wiley and Sons, May 2011.

[77] J. Pruitt and J. Grudin. Personas: Practice and theory. In *Proceedings of the 2003 Conference on Designing for User Experiences*, DUX '03, pages 1–15. ACM, New York, NY, USA, 2003. ISBN 1-58113-728-1. URL http://doi.acm.org/10.1145/997078.997089.

[78] Y. S. Ryu. *Development of usability questionnaires for electronic mobile products and decision making methods*. PhD thesis, Virginia Polytechnic Institute and State University, 2005.

[79] F. E. Sandnes. Eazytagz-an environment for building powerful interactive teaching portals for non-programmers. In *Proceedings of the ICEE International Conference on Engineering education*. Citeseer, 2002.

[80] J. Sauro. Seven tips for writing usability task scenarios seven tips for writing usability task scenarios seven tips for writing usability task scenarios seven tips

for writing usability task scenarios seven tips for writing usability task scenarios. URL https://www.measuringusability.com/blog/task-tips.php.

[81] J. Scholtz. Usability evaluation. *National Institute of Standards and Technology*, 2004.

[82] R. L. Shackelford and A. N. Badre. Why can't smart students solve simple programming problems? *International Journal of Man-Machine Studies*, 38(6):985–997, 1993.

[83] B. Shneiderman. Promoting universal usability with multi-layer interface design. In *ACM SIGCAPH Computers and the Physically Handicapped*, number 73-74, pages 1–8. ACM, 2003.

[84] N. C. Shu. *Visual programming*. Van Nostrand Reinhold Co., 1988.

[85] D. C. Smith. Pygmalion: a creative programming environment. Technical report, DTIC Document, 1975.

[86] R. B. Smith. The alternate reality kit: An animated environment for creating interactive simulations. In *Proceedings of 1986 IEEE Computer Society Workshop on Visual Languages*, pages 99–106, 1986.

[87] J. C. Spohrer and E. Soloway. Novice mistakes: Are the folk wisdoms correct? *Communications of the ACM*, 29(7):624–632, 1986.

[88] SPSS. SPSS software: Predictive analytics software and solutions. URL http://www-01.ibm.com/software/analytics/spss/.

[89] G. Stein. How an arcane coding method from 1970s banking software could save the sanity of web developers everywhere.

[90] UNData. UNdata sources. URL http://data.un.org/DataMartInfo.aspx.

[91] UPO. About the University Planning Office. URL http://www.ryerson.ca/upo/about/index.html.

[92] UsabilityNet. Questionnaire resources. URL http://usabilitynet.org/tools/r_questionnaire.htm.

[93] UsabilityNet. Scenario of use. URL http://usabilitybok.org/scenario-of-use.

[94] K. C. Viktor Mayer-Schönberger. *Big Data: A Revolution that Will Transform how We Live, Work, and Think*. Houghton Mifflin Harcourt, 2013.

[95] W. Visser. More or less following a plan during design: opportunistic deviations in specification. *International Journal of Man-Machine Studies*, 33(3):247–278, 1990.

[96] G. M. Weinberg. *The psychology of computer programming*, volume 932633420. Van Nostrand Reinhold New York, 1971.

[97] C. Wharton, J. Rieman, C. Lewis, and P. Polson. The cognitive walkthrough method: A practitioner's guide. In J. Nielsen and R. L. Mack, editors, *Usability Inspection Methods*, pages 105–140. John Wiley & Sons, Inc., New York, NY, USA, 1994. ISBN 0-471-01877-5. URL `http://dl.acm.org/citation.cfm?id=189200.189214`.

[98] Wikipedia. Daniel D. McCracken. URL `http://en.wikipedia.org/wiki/Daniel_D._McCracken`.

[99] Wikipedia. Kathleen Booth. URL `http://en.wikipedia.org/wiki/Kathleen_Booth`.

[100] J. M. Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, 2006.

[101] J. M. Wing. Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1881):3717–3725, 2008.

[102] M. Wrubel. *A primer of programming for digital computers*. McGraw-Hill series in information processing and computers. McGraw-Hill, 1959. LCCN 58013895. URL `http://books.google.ca/books?id=vPs4AAAAIAAJ`.