Black-Box Testing Technique for Information Visualization. Sequencing Constraints with Low-Level Interactions

Martín L. Larrea[1]

[1] *Laboratorio de Investigación y Desarrollo en Visualización y Computación Gráfica, VyGLab, Departamento de Ciencias e Ingeniería de la Computación, Universidad Nacional del Sur, San Andres 800, Bahía Blanca, Argentina,* mll@cs.uns.edu.ar

Abstract

The software development process has matured significantly o ver t he p ast d ecade. W e are currently in a state where the need for the verification a nd v alidation o f t he p roduct under development is unquestionable. Visualizations, as software products, must go through such verification a nd v alidation c ontrol. A t the implementation level, a visualization software is no different from any other software, its source code can be verified a nd v alidated by using any available white-box technique. The usability studies and controlled experiments are helpful to understand how the user perceives and uses the visualization. However, at an interaction level, a visualization software is not like any other software. Most interactions on a traditional software, are based on buttons and text fields w hile o n a v isualization, t he interactions are mostly based on zooming, selecting and filtering visual e lements. The black-box techniques found in the literature, also known as functional tests, are not suitable for this context. This paper describes a black-box technique tailored for information visualization implementations. The technique is built on constraints imposed over the sequences of low-level interactions available in the visualization and the User Action Notation.

Keywords: V isualization systems and software, Software/Program Verification.

Recceived 11 October 2017 / Revised 11 December 2017 / Accepted 21 February 2017

1 Introduction

Verification and Validation (V&V) is the process of checking that a software system meets its specifications and fulfills its intended purpose. The software engineering community have acknowledged the importance of V&V process to ensure the quality of its software products. In its essence, the process consists of selecting elements from the program, module or function input domain, executing the program with those elements as input and comparing the actual outputs with the expected outputs. Even though the importance of the V&V process is well known and documented, it has commanded a partial attention in the information visualization community.

A search on IEEE Xplorer, ACM Digital Library and Google Scholar, with the query "visualization testing verification validation" returns a list of papers where the focus is not on the functional testing of the visualization software. These papers focus on the problem of evaluating visualization system for effectiveness on user experience. Without detracting from usability testing, functional testing is crucial in any software development. All visualization software must be submitted to the V&V process, pointed out by Kirby and Silva in [1]. On the other hand, Graphical User Interfaces (GUI) Testing is a more popular and developed topic. GUI Testing is a system testing of a software that has a GUI front-end. There are many research papers on this topic ([2, 3, 4, 5, 6]). As discussed later, these works are not well suited to the characteristics of a visualization. We will focus in one GUI testing technique in particularly, the User Action Notation (UAN), as it is one of the basis of our proposal.

The V&V process, also known as software testing or just testing, is composed of V&V techniques. There are many different V&V techniques which are applicable at different stages of the development lifecycle. The two main categories of testing techniques are white-box and black-box. In the first one, the testing is driven by the knowledge and information provided by the implementation or source code. In the second one the specification of the software, module or function is used to test the object under review. Black-box testing is not usability testing; Usability testing tests the extent to which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component. Black-box testing evaluates high-level design and the customer requirements specification, to ensure the system does what it is intended to do.

All black-box testing techniques, design its tests cases based on the software specification. Some

of these techniques involve the GUI components of the software and the interactions with them. Buttons, text fields and drop down list are common elements among those GUI. Nonetheless, a visualization constitutes a GUI by itself with more components than a regular user interface. Beside buttons and text fields, a visualization may have glyphs, 3D or 2D visual objects that change location or shape according to users interactions. In these cases, the black-box techniques that rely only on common GUI components are not suitable.

Those techniques which do not involve graphic components, use decision tables ([7]) or other forms of tabular representation to test the software. Among those, there are very informal techniques which are very difficult to methodize and rely heavily on the tester goodwill. Others allow systematizing the testing by using a formal specification, which is very complicated to achieve for a visualization ([8]).

A visualization software, at the source code level, is just another piece of software and as such, can be tested with any available white-box technique ([9]). Furthermore, researchers have investigated whether and how visualization software features can improve the effectiveness and user perception of information systems. However, there are still challenges ahead, in particularly, the black-box testing of information visualizations is an uncharted area. If the visualization community wants to embrace the V&V process as a way to ensure the quality of their software products, then we need to adapt existing techniques to this context or to define new ones. To respond to this gap, in this work a black-box technique based on UAN and sequences of low-level interactions is introduced.

We can identify two levels of interactions on visualization, high and low-levels. High-level interactions describe the reasons for the creation of the visualization, the user goals. Different authors describe different high-level interactions; Kovalerchuk and Schwing established ([10]) three high-level interactions: *Illustrate*, *Reason* and *Discover*; while Keller and Keller ([11]) described nine. Low-level interactions are those provided by the visualization to achieve the high ones i.e. the user's goal. Shneiderman ([12]) identified seven low-level interactions: *Overview*, *Zoom*, *Filter*, *Detail on demand*, *Relation*, *History* and *Extraction*.

Low-level interactions exist at the implementation level of the visualization, while high-level interactions are more abstract. Because we want to design a testing technique for information visualization implementation, this article focuses on low-level interactions.

In the remaining parts of this article, we first review the concepts of V&V in the software development process as well as in the context of visualizations. The following section provides background V&V in general and applied to GUI and visualizations. Then, we describe the proposed technique, a black-box testing method suitable for information visualizations. We introduce 2 examples of the use of such technique on existing visualizations. We conclude with a brief discussion on limitations and advantages of our approach and the proposed future work.

2 Background Review

2.1 Background on Verification & Validation

Software testing is involved in each stage of software life cycle, but how we test and what we test on each stage of software development is different, the nature and the goals of what is being tested are different. Based on [13], there are 8 types of testing in the life cycle: **Unit testing** is a code based testing which is performed by developers, this testing is mainly done to test each and individual units separately. The Unit testing can be done for small units of code, generally no larger than a class. **Integration testing** validates that two or more units work together properly, and inclines to focus on the interfaces specified in low-level design. **System testing** reveals that the system works end-to-end in a production-like location to provide the business functions specified in the high-level design. **Acceptance testing** is conducted by business owners; the purpose of acceptance testing is to test whether the system does, in fact, meet their business requirements. **Regression Testing** is the testing of software after changes have been made to ensure that those changes did not introduce any new errors into the system. **Alpha Testing** is usually applied at the developer's site, with the presence of the developer. **Beta Testing** is done at the customer's site with no developer in site. **Functional Testing** is done for a finished application, its goal is to verify that the system provides all the required behavior.

In the context of V&V, black-box testing is often used for validation (i.e. are we building the right software?) and white-box testing is often used for verification (i.e. are we building the software right?). In black-box testing, the test cases are based on the information from the specification. The software testers do not consider the internal source code of the test object. The focus of these tests is solely on the outputs generated in response to selected inputs and execution conditions. The software tester sees the software as a black box, where information is input to the box, and the box

sends something back out. This can be done purely based on the requirement specification knowledge; the tester knows what to expect from the black box to send out and tests to make sure that the box sends out what it's suppose to send out.

Oppositely, white-box testing, also called structural testing, designs test cases based on the information derived from the source code. White-box testing is concern with the internal mechanism of a system, it mainly focuses on the control flow or data flow of the program. White-box and black-box testing are considered to complement each other. In order to test software correctly, it is essential to generate test cases from the specification and source code. This means that we must use white-box and black-box techniques on the software under development.

Each test technique, both white-box and black-box, must describe a test model and, at least, one coverage criteria. Test models describe how to generate test cases, it can be a graph, a table or a set of numbers. Coverage criteria are used to steer and stop the test generation process ([14]), they are usually boolean conditions. They have widely accepted means of assessing the quality of a test ([15]). Both concepts will be revisited later, where we will discuss a particular testing technique.

2.2 Background on Verification & Validation in Visualization

In 2008, Kirby and Silva ([1]) stated the need to incorporate the V&V process into the visualization process. They said: *To gauge the extent to which verification already exists in the visualization community, we reviewed papers from the past five years (2003 to 2007) of the IEEE Visualization conference (http://vis.computer.org). To establish the extent to which the nomenclature is already used, we searched the texts of these papers for occurrences of "verify", "validate" and other forms of these words. Fewer than half of the papers ever used any of these words, let alone in the context of validation or verification. To check whether these concepts exist, we reviewed a subset of the papers in greater detail. While many include sections presenting a logical and mathematically sound derivation of their technique and evaluation of their algorithms' performance, only a handful discussed concepts related to either validation or verification. Those that did tend not to adhere to any standard organization or systematic methodology.*

We extended the search Kirby and Silva did and included ACM Digital Library, IEEE Xplorer and Google Scholar up to 2015. We found equivalent results; Xiaojun et. al ([16]) test how users perceive visualizations, Koshman ([17]) talks about testing user interactions but he actually does a test on user's cognitive preattentive processing. Aerts et. al ([18]) test the effectiveness of two visualization techniques. These are some of the works with titles that mention "testing", "verification" or "validation" and when reading them, they were actually some sort of usability evaluation. There were many more than explicitly mention usability testing in their title.

Etiene's work ([19, 20, 21]) on verifiable visualization checks the mathematical calculations involved in the visualization process. He stated that visualization, scientific volume rendering specifically, has not fallen under the same rigorous scrutiny as other components of the pipeline like mathematical modeling and numerical simulation. These works are more related to a white-box testing technique approach.

There is a significant effort into usability and user experience testing in the context of information visualization but no equivalent endeavor into black-box techniques. If a software is only tested by white-box and usability techniques then this means that we are not doing our best to ensure the quality of the final product.

2.3 Background on Verification & Validation in GUI

Banerjee et al. ([2]) define the term GUI testing to mean that a GUI based application, i.e., one that has a GUI front-end, is tested solely by performing sequences of events (e.g., "click on button", "enter text", "open menu") on GUI widgets (e.g., "button", "text-field", "pull-down menu"). From the user's point of view, GUIs offer many degrees of usage freedom, i.e., users may choose to perform a given task by inputting GUI events in many different ways in terms of their type, number and execution order.

Banerjee et al. also provided a study of the existing body of knowledge ([2]) on GUI testing since 1991. Hellman et al. ([22]) presented a review of test driven development of GUI and Memon and Nguyen ([23]) presented a classification based on model-based GUI test techniques.

Hellman et al. stated that GUI testing is very difficult in part due to the degree of freedom GUIs allow users. GUIs can enter a large number of possible states in response to user input, and it is often difficult to determine the validity of a given state in an automated fashion. This becomes more complex when we consider an information visualization. Among the techniques reviewed by Hellman et al. we can found Capture Replay Tools (CRT), Invariant Based Testing and Model Based Testing.

CRT techniques are based on recording the mouse and keyboard activities as scripts in order

to replay them later. These techniques are not suitable for information visualization because visual elements are not always in the same place. The same visualization technique on two different data set will produce different representations. A record from one representation will not work on the other.

In Invariant Based Testing rules which define expected or prohibited system behavior are created. After each interaction with the GUI elements, the invariants are checked. This approach is based on knowledge of how the GUI is implemented, so this is a white-box technique, not the subject of our study.

Finally, Model-based testing requires the generation of an intermediate version of the GUI, either by reverse-engineering a working GUI or generating one based on specifications. The main disadvantage of this type of techniques is that a model must be created and maintained. If the model describes the GUI widgets, which vary a lot during the development of an interface, then the maintenance of the model will require a lot of effort.

Memon and Nguyen ([23]) acknowledged that quality assurance is becoming increasingly important for GUIs as their functional correctness may affect the quality of the entire system in which the GUI operates. They presented 16 model based testing techniques. These techniques require the creation of a model, like a table, a graph or a tree and an algorithm to create test cases based on the model. As we said earlier, all models that are based on GUI widget must be updated every time there is a modification in the interface. The advantage of model based testing is that, having a model with a formal representation allows for the use of an algorithm to create the test cases. In many cases, this algorithm can be automated.

Banerjee et al. ([2]) developed a classification scheme for 136 articles about GUI testing. They concluded that there is a large focus on model based testing in the literature, but none of the commercially available tools are model based. Most commercial tools are based on CRT techniques. They saw this as a signal of disconnection between researchers and practitioners.

3 Our Proposal

Our goal in this work is to define a testing technique suitable for information visualization, focusing on a black-box approach. We want to provide an easy to use methodology, with an easy to read and understand representation of the visualization's interactions. More particularly, we want a method that the developer or even the user of the visualization can use, without the need of a testing specialist. We want to reduce the disconnection between commercial tools and literature proposal, between researchers and practitioners, mentioned by Banerjee et al ([2]).

Our proposal is based on two previous work, User Action Notation and Sequencing Constraints. Both concept are introduce in the following subsections and then our technique for black-box testing on Information Visualization is describe.

3.1 User Action Notation

User Action Notation (UAN) is a task and user-oriented notation for behavioral representation of asynchronous, direct manipulation interface designs ([4, 6]). At the lower level, user actions are associated with feedback and system state changes. The notation makes use of visually onomatopoeic symbols. UAN was created as a tool-supported technique capable of specifying the behavioral aspects of interaction systems, the tasks and the actions a user performs to accomplish those tasks.

The example of UAN presented by Hartson et al. ([4]) can be used here to clarify how it is used. Consider the task *moving a file icon*, which can be described in prose as:

1. Move the cursor to the file icon. Depress and hold down the mouse button. Depressing the mouse button selects the file, indicated by the highlighting of its icon.

2. With the button held down, move the cursor. An outline of the icon follows the cursor as you move it around.

3. Release the mouse button. The display of the icon is now moved to where you released the button.

These actions are described in UAN as:

1. $\sim [file\_icon] M^{\vee}$

2. $\sim [x, y]* \sim [x', y']$

3. $M^{\wedge}$

The $\sim$ denotes moving the cursor, in this case into the context of the file icon. $M^{\vee}$ represents depressing the mouse button. In the second part $\sim [x, y]$ indicates movement of the cursor to an arbitrary point x, y on the screen. The $*$ means to perform, zero or more times, the task to which it is attached. Thus, $\sim [x, y]* \sim [x', y']$ means to move the cursor to a succession of zero or more arbitrary points about the screen, ending at the point x', y'. Finally, in the third, part the mouse button is released.

UAN primary goal is the representation of the behavior of the interactions and the system state, in the interface development process. It was not intended to use as a testing tool. It has a rich symbolic alphabet and even allows for the representation of the system state and its changes. Its power of expression means that some simple actions described in UAN can be difficult to read and interpret.

3.2 Sequencing Constraints

In 1994, Kirani and Tsai ([24]) presented a technique called Message Sequence Specification that, in the context of an object oriented program, describes the correct order in which the methods of a class should be invoked by its clients. The method sequence specification associated with an object specifies all sequences of messages that the object can receive while still providing correct behavior.

Their strategy uses regular expressions to model the constraints over the correct order of the invocation of the methods i.e. the regular expression is the test model. Method names were used as the alphabet of a grammar which was then used to statically verify the program's implementation for improper method sequences. A runtime verification system identifies incorrect method invocations by checking for sequence consistency with respect to the sequencing constraints.

If a class C has a method M_1, this is noted as C_{M_1}. Sequence relationships between two methods were classified into three categories, sequential, optional and repeated. If the method M_1 of C should be invoked before the method M_2 of the same class, then this relationship is sequential and is represented as

$$C_{M_1} \bullet C_{M_2}$$

If one, and only one of the methods M_1 and M_2 can be invoked then this relationship is optional and is represented as

$$C_{M_1} | C_{M_2}$$

Finally, if the method M_1 can be invoked many times in a row then this is a repeated relationship and is represented as

$$(C_{M_1})^*$$

For example, if a class X has three methods *create*, *process* and *close*, a possible sequencing constraint based on Message Sequence Specification could look like

$$X_{create} \bullet (X_{process})^* \bullet X_{close}$$

If class X is part of a larger system S, then we could statically check the source code of S to see if all calls to X's methods follow the defined grammar. If a static analysis is not enough, we could implement a runtime verification system that tracks all calls to X's methods and checks dynamically the sequence of calls against the grammar.

This technique can also be used to test the robustness of a system. Continuing with the class X as an example, we can use the defined grammar to create method sequences that are not a derivation from the grammar, i.e. incorrect sequences methods. These new sequences can be used to test how the class handles a misuse. For example, how does the class X respond to the following sequence of calls:

$$X_{create} \bullet X_{close} \bullet X_{process}$$

Testing with the sequences generated by the grammar and with those that were not, are the two techniques presented in [26] a work that extended the research done in [24].

3.3 Testing Information Visualization using Sequencing Constraints with Low-Level Interactions

UAN is a suitable representation of the interactions in a GUI and, in our case of interest, in a visualization. But it has a complex alphabet and its output is not easy to read. Kirani and Tsai ([24]) share multiple similarities with UAN, where instead of working with GUI interactions they used class methods. Their work does not hold the same expressivity that UAN but it is clearer and easy to interpret.

In this work, we propose a merge of them in order to design a black-box testing technique for Information Visualization. Ji Soo et al. ([25]) described a visualization system as a two-part system, representation and interaction. They said: "*The representation component, whose roots lie in the field of computer graphics, concerns the mapping from data to representation and how that representation is rendered on the display. The interaction component involves the dialog between the user and the system as the user explores the data set to uncover insights.*"

The user uses the interactions to express orders to the visualization, the interactions work as a language and as such they may have restrictions or rules on how is used. Kirani and Tsai ([24]) idea of identifying the restrictions for the correct order in which the methods of a class could be invoked can be applied here. Our proposal is to use the basis of what was originally design as a white-box testing technique for object oriented programs in order to develop a black-box testing technique for visualization implementations.

3.4 Our Contribution

Instead of writing a sequence constraint on the methods of a class, we will write a Sequence Constraint on the Interactions (SCI). Each SCI

involves a set of binary or unary operators and a set of symbols. For simplicity, for now on the symbols will represent the actual interactions available in the visualization. Following the work by Kirani and Tsai ([24]), the operators will be:
Sequential: If an interaction I_2 must always go after the interaction I_1 then there is a sequential relationship among them, denoted as $I_1 \bullet I_2$.
Optional: If the user can choose between two interactions, said I_2 and I_1, then there is an optional relationship among them, denoted as $I_1|I_2$. Note that in this case, the notation $I_2|I_1$ is equivalent.
Repetition: If the user can use the interaction I_1 multiple times in a row, then it is a repetition. Unlike the work done at [24] and [26], we introduce two types of repetition, one that implies that at least one time the user must use I_1 and the other that allows for zero appearance of I_1. The symbol $*$ will be used to represent cardinality 0 or more, and the symbol + will be used for 1 or more. If I_1 can be used zero or more times, then this is represented as ${I_1}^*$; if I_1 must be used at least one time, it is expressed as ${I_1}^+$. ${I_1}^+$ is equivalent to $I_1 \bullet {I_1}^*$.

These elements can be combined to form more complex expressions. If the user can use one of three interactions multiple times, this can be expressed as $(I_1|I_2|I_3)^+$. In this case by using the symbol + we are saying that at least one of the interactions must be used once. Repetition operators have precedence over Sequential and Optional operators. The Optional operator takes precedence over the Sequential one. Parentheses can be used to define the interpretation of a SCI. Suppose we have three interactions I_1, I_2 and I_3, then the following SCI

$$ {I_1}^+ \bullet I_2 | I_3 $$

expresses that first we must consider the Sequential operator, *use I_1 one or more times* and then we must choose between using I_2 or I_3. By using parentheses we can change the interpretation of the SCI

$$ ({I_1}^+ \bullet I_2) | I_3 $$

In this case, we first consider the Optional operator, we must choose between using I_3 or the expression between the parentheses.

We can use the symbols in the SCI to represent more than the available interactions, we can also use them to modularize expressions. Let us imagine a visualization V_X with six interactions, *Open, Pan, Zoom, Selection, Detail* and *Close*. *Open* represents the creation of the visualization, from opening the source data to setting the visualization process; when *Open* concludes the user has the actual visualization on screen. *Detail* represents detail on demand and can only be used if the user selected something using *Selection*. *Pan* and *Zoom* allow the user to explore the visualization. The following grammar represents the constraints over the sequencing of interaction in V_X, considering O for *Open*, P for *Pan*, Z for *Zoom*, S for *Selection*, D for *Detail* and finally, C for *Close*.

SCI for V_X: $O \bullet (O|Z|P|(S^+ \bullet D^*))^* \bullet C$

We can define two new symbols *Selection&Detail* and *MainInteraction*, and redefine the SCI for V_X:

$$ Selection\&Detail = S^+ \bullet D^* $$
$$ MainInteraction = O|Z|P|Selection\&Detail $$
$$ SCI for V_X : O \bullet MainInteraction \bullet C $$

This grammar means that the first valid interaction with V_X is *Open*, then the user can *Open* again or *Zoom* or *Pan* or *Selection*. Note that if the user want *Detail*, first the user must do at least one *Selection*. The interaction with V_X ends when the user ends the visualization with the *Close* interaction. The symbols $Selection\&Detail$ and $MainInteraction$ allow a clearer expression for such behavior.

3.5 Coverage Criteria

In V&V the term Coverage Criteria is a measure or indicator of the amount of test to perform. In other words, it answers the question "how much test should we do?". Each testing technique, black or white-box, have its own indicator of Coverage Criteria. A single technique may have many Coverage Criteria and among them there is a quality/complexity relationship. On one hand, we find very simple criteria which will generate simple test cases with a not too extensive coverage; while on the other end, there will be complex criteria that will generate very complex test cases. The former detects fewer problems than the later.

With our proposal of Sequencing Constraints with Low-Level Interactions there are two types of test cases that we can generate, valid test cases based on valid interaction sequences and invalid test cases based on interaction sequences that cannot be derived from the defined grammar.

Let I be the set of interactions available on the visualization V_X, and G, the SCI for V_X using the elements of I. Consider T to be the set of test cases where each case is a sequence of the interactions in I. With these elements, we can now introduce the Coverage Criteria for Sequencing Constraints with Low-Level Interactions. These criteria are divided into two categories, as in [26], coverage criteria for valid sequences and for invalid ones. The criteria on each category were created for our technique.

3.6 Coverage Criteria for Valid Sequences

Base Coverage: Let i is the minimum length of valid sequences derived from G, then T satisfies the Base Coverage Criteria if and only if T contains all the possible sequences derived from G of length i. If i equals 0 then T is an empty set and satisfies the Base Coverage Criteria.

Base+1 Coverage: Let i be the minimum length of valid sequences derived from G, then T satisfies the Base+1 Coverage Criteria if and only if T contains all the possible sequences derived from G of length $i+1$.

Base+n Coverage: This is a generalization of the previous coverage. Let i be the minimum length of valid sequences derived from G, then T satisfies the Base Coverage Criteria if and only if T contains all the possible sequences derived from G of length $i+n$, where $n >= 2$. It is important to note that G may impose limits on how large n can be.

3.7 Coverage Criteria for Invalid Sequences

Invalid Coverage: T satisfies the Invalid Coverage Criteria if and only if T contains all the possible sequences of length 1 that are **not** derived from G.

Invalid-2 Coverage: T satisfies the Invalid-2 Coverage Criteria if and only if T contains all the possible sequences obtained from the combination of 2 interactions of I but are **not** derived from G.

Invalid-n Coverage: T satisfies the Invalid-n Coverage Criteria if and only if T contains all the possible sequences obtained from the combination of n interactions on I, where $n >= 2$, but are **not** derived from G.

3.8 Examples of Coverage Criteria

Continuing with the SCI G defined for V_X,

$$G = O\bullet(O|Z\ |P|(S^{+}\bullet D^{*}))^{*}\bullet C$$

we can now generate test cases based on the different criteria. Let T_1 be a set of test cases as follows:

$$T_1 = \{O\bullet C\}$$

T_1 does satisfy the Base Coverage Criteria because G does not allow an empty sequence or a sequence of length 1, and *Open* must always be present at the beginning and *Close* always ends the sequences. Because the middle of the SCI is enclosed in $*$ we can consider it empty. So, the smallest length for valid sequences from G is 2. Then can expand T_1 and create T_2 as:

$$T_2 = \{O\bullet C, O\bullet O\bullet C, O\bullet Z\bullet C, O\bullet P\bullet C, O\bullet S\bullet C\}$$

T_2 satisfies the Base+1 Coverage Criteria because it contains all the possible sequences derive from G of length 3, which is +1 on the smallest length possible. T_2 also contains all the possible sequences from G of length 2, so it also satisfies Base Coverage Criteria. If we now consider T_3:

$$T_3 = \{O\bullet C, O\bullet Z\bullet C, O\bullet P\bullet C, O\bullet S\bullet C, O\bullet S\bullet S\bullet D\}$$

T_3 satisfies Base Coverage Criteria but does not satisfy Base+1 Coverage Criteria because it does not contain the $(O\bullet O\bullet C)$ sequence, it neither satisfies Base+2 Coverage Criteria as it only has the sequence $(O\bullet S\bullet S\bullet D)$ of length 4.

These Criteria are based on the length of the sequences of interactions. Some SCI will impose an upper limit on that value but others may not. When there is no limit on how long these sequences can be, is up to the person using this testing technique to decide how far to go.

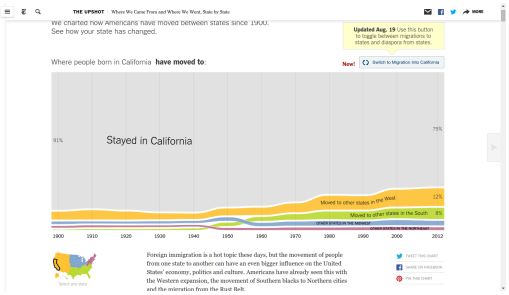

Figure 1: Theme river visualization for the migration of people within the United States

4 Test Cases using Sequencing Constraints with Low-Level Interactions

In this section we will use our technique to generate test cases on two visualizations. The chosen visualizations were part of the Best Visualization Projects of 2014 by the Flowingdata website[1].

4.1 Where We Came From and Where We Went

This visualization[2] shows how people in the United States have moved between states since 1900. As Fig. 1 shows, the authors of the visualization used a theme river technique ([27]) for the visual representation of information. The site offers 51 visualizations, one for each state. The jump from one visualization to another is achieved through a selector on the top of the web page. The low-level interactions are:

- *Select Origin State:* This is done by the selector on the top. After choosing a state,

[1] http://flowingdata.com

[2] http://www.nytimes.com/interactive/2014/08/13/upshot/where-people-in-each-state-were-born.html

the visualization will show in percentage format, where the people born in that state have move to.

- *Switch to Out of State:* This interaction changes the visualization to show where those who live in the selected state are from. It is done by clicking a button.

- *Detail:* When the user moves the mouse over the different rivers in the visualization, a pop up box appears showing the exact percent value at a specific time for a specific location. This interaction is possible in the Origin State and in the Out of State visualizations.

To do Sequencing Constraints with Low-Level Interactions we will replace each interaction with a simplified representation, *Select Origin State* will be *SO*, *Switch to Out of State* will be *LO* and *Detail* will be *De*. The visualization, as it is presented in the website, start automatically with where people born in California have move to. There is no need to explicitly load the first data set. There is no restriction in the order on which of the three low-level interactions can be used, this mean that the SCI for this visualization is

$$(SO|LO|De)^*$$

4.1.1 Coverage Criteria for Valid Sequences

Because no interaction in this example is obligatory, the minimum sequence of interaction valid for this visualization is the empty sequence. Because of this, the Base Coverage is achieved by just running the visualization, i.e. without any interaction. We access the website, and thus the visualization appears with California as default state. So the visualization behave correctly.

In order to achieve Base 1 Coverage we need to provide a set of test T such as T contains all the possible sequences of length 1 derived from $(SO|LO|De)^*$. T_1 satisfies this condition.

$$T_1 = \{SO, LO, De\}$$

T_1 means that we must execute the visualization three times. The first time we change the origin state and only do that, for example we chose to change from California to Texas. On the second run, we switch to *Out of State*. Because the default state is California, we change the visualization from "Where the people born in California have move to", to "Where are from those who live in California". On the final run for T_1 we use the interaction *Detail*. On the visualization for California we position the mouse on the vertical line corresponding to the year 1950, in the *Stayed in California* river. In all three cases the visualization behaves as expected, so we can conclude that T_1 was passed.

The following test set, T_2 accomplish Base 2 Coverage and was also passed by the visualization successfully.
$T_2 = \{SO\bullet SO, SO\bullet LO, SO\bullet De, LO\bullet S, LO\bullet LO, LO\bullet De,\ De\bullet SO, De\ \bullet LO, De\bullet De\}$

4.1.2 Coverage Criteria for Invalid Sequences

Because the sequence constraints for this visualization is $(SO|LO|De)^*$ there are no invalid sequences. Hence, it is not possible to satisfy any Invalid Coverage.

4.2 How the Recession Reshaped the Economy

This visualization[3] shows how during the ten years following the great recession experienced in the United States, 9 million lost jobs were recovered.

For the visual representation of information, the authors chose a line graph, where each line represents a different industry. The X-axis is used for the level of wages, from lower to higher; and the Y-axis is used from the number of jobs gain or lost since the recession. Each line shows a time period from 2004 to 2014.

This visualization is far more complex than the previous one, in its content and interaction model. We can divide the visualization into three parts, where the set of available low-level interaction on each part is different: the initial visualization (Figure 2), the main one (Figure 3) and finally the detailed representation (Figure 5).

The initial visualization (Figure 2) is the one that appears when the user opens the visualization web page. This a compact representation of the main visualization. The Y-Axis is compressed in order to gain space for the title and description. Like the previous example, the data set is loaded automatically when the user enters the website. We can detect four low-level interaction in this stage.

- *Hover over line:* This interaction works as *DetailOnDemand*. When the user places the mouse on any line in the visualization, a small visualization appears showing more detail about the industry in display. We will use the symbol H_{line} for this.

- *Go through the detail:* When the user moves the mouse over the small visualization created by H_{line}, the information in it is updated based on the position of the mouse within. We will use the symbol D for this. Because

[3] http://www.nytimes.com/interactive/2014/06/05/upshot/how-the-recession-reshaped-the-economy-in-255-charts.html

Figure 2: Visualization presented to the user when he/she enters the website. It is a compact representation of the main visualization.

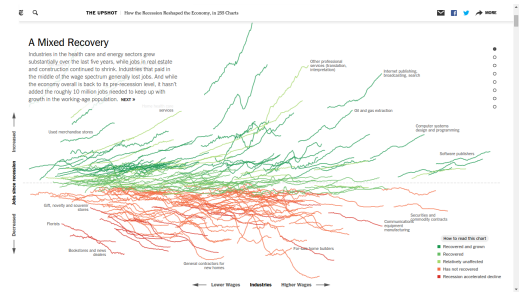

Figure 3: The main visualization of how jobs were created or lost during a ten year period on different industries.

D must always go after H_{line} we will create a new symbol $Z = H_{line} \bullet D^*$. Z indicates that the user can go through details after placing the mouse over a line, and he/she can go through details many times.

- *Click Scroll & Mouse Scroll:* These are two interaction that allow the user to move to the next part of the visualization. On the bottom of the initial visualization there is a label titled "Scroll" that when the user clicks on it, changes to the main visualization. The same effect is achieved by using the mouse scroll. We will use C_{scroll} and M_{scroll} for these ones.

The SCI for the initial visualization, called $V_{initial}$ is

$$V_{initial} = Z^* \bullet (S_{scroll} | M_{scroll})$$

This means that, once the user enters the website, he/she can do as many semantic zooms as wanted and then moves to the next part of the visualization by clicking the scroll label or by mouse.

The main visualization (Figure 3) expands the Y-Axis of the previous one allowing a more clear view of the different lines. It adds eight filters in the form of circles on the top right corner of the visualization, a description on how to read the information which appears on the bottom right corner and a description of the information on screen, in the top left corner. The *Click Scroll* interaction is no longer available but *Mouse Scroll* is possible. *Hover over line* and *Go through the detail* are available. The new low-level interactions are:

- *How to Read:* When the user places the mouse over the text in the bottom right corner of the visualization a brief explanation of how the visualization works appears. We will use the symbol R for this. It is important to notice that this interaction is only available when the first filter is selected. The remaining filters remove the legend.

- *Hover Labels:* Each filter is accompanied by a description of the data on screen. On each description, except for the first filter and fourth filters, there are labels that open the small visualization described earlier. This action happens when the user places the mouse over the labels. This interaction will be represented by H_{labels}.

- *Click Next:* Each description that appears on each filter contains a "Next" label. When the user clicks on it, the visualization moves to the next filter. We will use the symbol C_{next} for this. We will add a new symbol $N = (C_{next} | M_{scroll})$.

- *Filters:* The eight available filters allow the user to remove some elements from the visualization in order to highlight others. There are seven group of industries, each filter shows one of those groups. The eighth one shows all the industries (Figure 4). When the first filter is selected, the user can do the Z and R interactions; when the fourth filter is selected the user can only do Z. When the other filters are selected the user can Z and H_{labels}. The user can change the visualization by selecting a different filter or by C_{next} or M_{scroll}. When the user does C_{next} visualizing the last filter on screen, the visualization changes from the main one to the detailed one. The symbols for the filters will be F_1, F_2, F_3, F_4, F_5, F_6, F_7 and F_8. In order to simplify the SCI we will use the symbol $f = F_2|F_3|F_5|F_6|F_7$.

The SCI for the main visualization is

$V_{main} = (F_1 \bullet (Z\ |R)^* \bullet N\ |F_4 \bullet Z^* \bullet N\ |f \bullet (Z\ |H_{labels})^* \bullet N)^* \bullet F_8 \bullet (Z\ |H_{labels})^* \bullet N$

This means that if the user chooses the first filter then Z and R are the available interactions, if the fourth filter is selected then only Z is possible. Filters 2, 3, 5, 6 and 7 allow the user to Z and H_{labels}. The eighth filter is handled differently because when the user uses the N interaction, he moves to the detailed visualization.

The final part of this visualization is the detailed one (Figure 5). This visualization contains all the small visualizations accessible in the main part. There are two interactions available at this point, *Go through the detail* and *Mouse Scroll*. There are no new interactions possible at this point. The SCI for the last part is $V_{detailed} = (D|M_{scroll})^*$. This means that the user can scroll detailed visualizations using the mouse or go through the details on each small visualization.

The entire visualization is described by the SCI $V = V_{initial} \bullet V_{main} \bullet V_{detailed}$. This is $V = Zoom^* \bullet (S_{scroll} \mid M_{scroll}) \bullet (F_1 \bullet (Z \mid R)^* \bullet N \mid F_4 \bullet Z^* \bullet N \mid f \bullet (Z \mid H_{labels})^* \bullet N)^* \bullet F_8 \bullet (Z \mid H_{labels})^* \bullet N \bullet (D \mid M_{scroll})^*$

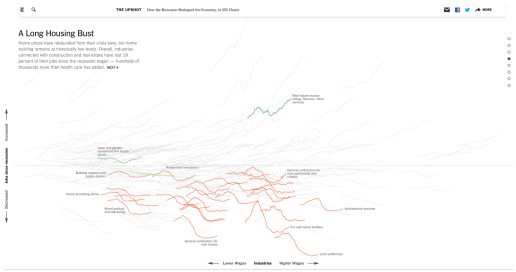

Figure 4: Eight filters are available on the main visualization. Each one allows the user to visualize a subset of industries.

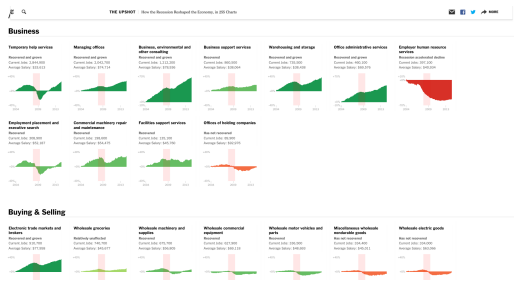

Figure 5: All the small visualizations are listed in the final part of the website. These visualizations are divided into industrial categories.

4.2.1 Coverage Criteria for Valid Sequences

The minimum sequence of interaction valid for this visualization is 3. A scroll as the first interaction, then the selection of the last filter on screen and finally, a click on the label "Next" or a mouse scroll. The following test set T_3 satisfies the Base Coverage criteria. No errors were found when executing T_3.

$T_3 = \{C_{scroll} \bullet F_8 \bullet C_{next},\ M_{scroll} \bullet F_8 \bullet C_{next},\ C_{scroll} \bullet F_8 \bullet M_{next},\ S_{scroll} \bullet F_8 \bullet M_{next}\ \}$

Valid sequences of length 4 are possible and the following test set T_4 contains all of them, satisfying the Base+1 Coverage criteria. No error were found when executing T_4. There are valid sequences of length 5 but they are too many to list in this article.

$T_4 = \{C_{scroll} \bullet F_8 \bullet C_{next} \bullet D,\ M_{scroll} \bullet F_8 \bullet C_{next} \bullet D,\ C_{scroll} \bullet F_8 \bullet M_{next} \bullet D,\ S_{scroll} \bullet F_8 \bullet M_{next} \bullet D,\ C_{scroll} \bullet F_8 \bullet C_{next} \bullet M_{scroll},\ M_{scroll} \bullet F_8 \bullet C_{next} \bullet M_{scroll},\ C_{scroll} \bullet F_8 \bullet M_{next}\ M_{scroll},\ S_{scroll} \bullet F_8 \bullet M_{next} \bullet M_{scroll},\ H_{line} \bullet C_{scroll} \bullet F_8 \bullet C_{next},\ H_{line} \bullet M_{scroll} \bullet F_8 \bullet C_{next},\ H_{line} \bullet C_{scroll} \bullet F_8 \bullet M_{scroll},\ H_{line} \bullet C_{scroll} \bullet F_8 \bullet M_{scroll}\}$

4.2.2 Coverage Criteria for Invalid Sequences

The following test set Ti_1 contains all the invalid sequences of length 1 for the current visualization. None of these sequences are possible when trying to execute them in the website. Notice that C_{scroll} and M_{scroll} are not present in the test set because they are valid first interactions.

$Ti_1 = \{H_{line},\ D,\ R,\ C_{next},\ H_{labels},\ F_1,\ F_2,\ F_3,\ F_4,\ F_5,\ F_6,\ F_7,\ F_8\ \}$

A test set containing the invalid sequences of length 2 will have close to 200 elements. As in the case of the valid sequences of length 5, it is to extensive to include in this article.

4.2.3 Test Cases Remarks

The last test cases showed the need for a tool to assist in the creation of the test sets. For some SCI the amount of sequences of interaction that can be derive from it can grow exponentially. It is our goal to develop a tool to create test sets based on the SCI and Coverage Criteria.

5 Conclusions and future work

In this paper we have described a black-box technique for testing information visualization implementations. The technique is based on the constraints found in the order in which low-level interactions must be used in the visualization. We presented several Coverage Criteria for valid and invalid sequences. Black-box testing on visualizations is an unexplored area, one that must be considered in order to ensure the quality of any visualization implementation. Our goal is to start with the development of test methods for visualizations, adapted for the needs and requirements of the area. These methods that can be used to provide a rigorous evaluation of each visualization.

Future research must include the development of a framework to test visualizations using the proposed technique. Such framework should be able to interact with any implementation for runtime verification, that is to check the order in which interaction are used at runtime against

the generated grammar. The framework must also generate test cases based on the SCI for the tester to used, valid and invalid sequences using the Coverage Criteria. We must continue investigating in order to achieve a framework with the full support of automated testing.

Finally, the proposed technique should be extended to include parameterized interactions. As it was presented in this article, a SCI can describe that, for example, before a *Detail on demand* the user must do a *Selection*, but nowhere it is said that the *Detail on demand* is performed on the items selected by the *Selection* interaction. This relation may seem trivial now, but a parametrized technique will be more expressive, and as such more powerful.

Acknowledgment

This work was partially supported by the following research projects: PGI 24/N037 and PGI 24/ZN29 from the Secretaría General de Ciencia y Tecnología, Universidad Nacional del Sur, Argentina.

References

[1] Kirby, Robert M., and Cláudio T. Silva. “The need for verifiable visualization”. Computer Graphics and Applications, IEEE 28.5, 78-83, 2008.

[2] Banerjee, Ishan, et al. ”Graphical user interface (GUI) testing: Systematic mapping and repository.” Information and Software Technology 55.10, 1679-1694, 2013.

[3] Chang, Tsung-Hsiang, Tom Yeh, and Robert C. Miller. ”GUI testing using computer vision.” Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, 2010.

[4] Hartson, H. Rex, Antonio C. Siochi, and Deborah Hix. ”The UAN: A user-oriented representation for direct manipulation interface designs.” ACM Transactions on Information Systems (TOIS) 8.3, 181-203, 1990.

[5] Appert, Caroline, Michel Beaudouin-Lafon, and Wendy E. Mackay. ”Context matters: Evaluating interaction techniques with the CIS model.” People and Computers XVIIIâĂŤDesign for Life. Springer London, 279-295, 2005.

[6] A. C. Siochi and H. R. Hartson. Task-oriented representation of asynchronous user interfaces. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI ’89), K. Bice and C. Lewis (Eds.). ACM, New York, NY, USA, 183-188, 1989.

[7] Ferriday, C. “A Review Paper on Decision Table-Based Testing”. Swansea University, CS339-2007, 2007.

[8] Seo, K. I., Choi, E. M. “Comparison of five black-box testing methods for object-oriented software”. In Software Engineering Research, Management and Applications, 213-220, 2006.

[9] Khan, Mohd Ehmer. “Different approaches to white box testing technique for finding errors”. International Journal of Software Engineering and Its Applications 5.3, 1-14, 2011.

[10] Kovalerchuk, Boris, and James Schwing. “Visual and Spatial Analysis”. Advances In Data Mining, Reasoning, And Problem Solving, 2005.

[11] Keller, Peter R., and Mary M. Keller. ”Visual cues: practical data visualization”. Vol. 2. Los Alamitos, CA: IEEE Computer Society Press, 1993.

[12] Shneiderman, Ben. ”The eyes have it: A task by data type taxonomy for information visualizations.” Visual Languages, 1996. Proceedings., IEEE Symposium on. IEEE, 1996.

[13] Jorgensen, Paul C. Software testing: a craftsman’s approach. CRC Press, 2013.

[14] Weileder, Stephan. Test models and coverage criteria for automatic model-based test generation with UML state machines. Diss. Humboldt University of Berlin, 2010.

[15] Friske, Mario, Bernd-Holger Schlingloff, and Stephan WeiÃ§leder. ”Composition of Model-based Test Coverage Criteria.” MBEES. 2008.

[16] Yuan, Xiaojun, Xiangmin Zhang, and Alex Trofimovsky. ”Testing visualization on the use of information systems.” Proceedings of the third symposium on Information interaction in context. ACM, 2010.

[17] Koshman, Sherry. ”Testing user interaction with a prototype visualizationâĂŘbased information retrieval system.” Journal of the American Society for Information Science and Technology 56.8, 824-833, 2005.

[18] Aerts Jeroen , Keith C. Clarke, and Alex D. Keuper. ”Testing popular visualization techniques for representing model uncertainty.” Cartography and Geographic Information Science 30.3, 249-261, 2003.

[19] Etiene, Tiago, et al. ”Verifiable visualization for isosurface extraction.” IEEE Transactions on Visualization and Computer Graphics 15.6, 1227-1234, 2009.

[20] Etiene, Tiago, et al. "Topology verification for isosurface extraction." IEEE Transactions on Visualization and Computer Graphics 18.6, 952-965, 2012.

[21] Etiene, Tiago, et al. "Verifying volume rendering using discretization error analysis." IEEE transactions on visualization and computer graphics 20.1, 140-154, 2014.

[22] Hellmann, Theodore D., Ali Hosseini-Khayat, and Frank Maurer. "Agile interaction design and test-driven development of user interfacesâĂŞa literature review." Agile Software Development. Springer Berlin Heidelberg, 185-201, 2010.

[23] Memon, Atif M., and Bao N. Nguyen. "Advances in automated model-based system testing of software applications with a GUI front-end." Advances in Computers, 121-162, 2010.

[24] Kirani, Shekhar H., and W. T. Tsai. Specification and verification of object-oriented programs. Diss. University of Minnesota, 1994.

[25] Yi, Ji Soo, et al. "Toward a deeper understanding of the role of interaction in information visualization." Visualization and Computer Graphics, IEEE Transactions on 13.6, 1224-1231, 2007.

[26] Daniels, F. J., and K-C. Tai. "Measuring the effectiveness of method test sequences derived from sequencing constraints." Technology of Object-Oriented Languages and Systems, 1999. TOOLS 30 Proceedings. IEEE, 1999.

[27] Havre, Susan, Beth Hetzler, and Lucy Nowell. "ThemeRiver: Visualizing theme changes over time." Information Visualization, 2000. InfoVis 2000. IEEE Symposium on. IEEE, 2000.