

StatWire: Visual Flow-Based Programming for Statistical Analysis

Bachelor's Thesis
submitted to the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

*by
Johannes Maas*

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Prof. Dr. Chat Wacharamanotham

Registration date: 27.07.2017
Submission date: 18.08.2017

Eidesstattliche Versicherung

Name, Vorname

Matrikelnummer

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/
Masterarbeit* mit dem Titel

selbständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ort, Datum

Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Ort, Datum

Unterschrift

Contents

Abstract	xiii
Acknowledgements	xv
Conventions	xvii
1 Introduction	1
1.1 Tools for Statistical Analysis	1
1.2 Trends	2
1.3 StatWire	3
1.4 Overview	4
2 Related Work	5
2.1 Statistical Programming	5
2.2 Visual Programming	6
3 Tools	9
3.1 Statistical Analysis	9

3.1.1	Data-Centered	9
3.1.2	Non-Linear	10
3.1.3	Structured	10
3.2	Textual Programming	11
3.2.1	Control Flow	12
3.2.2	Dumping	12
3.3	Visual Programming	13
3.4	Hybrids	14
4	Concept	17
4.1	Key Idea	17
4.2	Components	18
4.2.1	Code	19
4.2.2	Canvas	20
4.2.3	Node Pool	22
4.2.4	Additional Features	23
4.3	Usage Scenarios	25
4.3.1	Placeholders	25
4.3.2	Education	26
4.3.3	Collaboration	27
4.3.4	Advanced Visual Programming	28
4.4	Design Process	30

4.4.1	StatLets	30
4.4.2	Application to Real-Life Analyses . . .	31
5	Evaluation	33
5.1	Benefits	33
5.1.1	Power	34
5.1.2	Ease of use	34
5.1.3	Modularization	35
5.2	Suggestions for Research Questions	35
5.2.1	Power	36
5.2.2	Ease of use	36
5.2.3	Modularization	38
5.2.4	Acceptance	38
5.3	Issues	39
6	Summary and Future Work	41
6.1	Summary	41
6.2	Future Work	42
A	Differences between Prototype and Concept	43
	Bibliography	45
	Index	47

List of Figures

4.1	Overview of Components	18
4.2	Closeup of Node	20
4.3	Data Viewer	21
4.4	Subset Selection	23

List of Tables

A.1 Differences between Prototype and Concept	44
---	----

Abstract

We introduce StatWire, a tool for statistical analysis that combines the power of statistical programming with the ease of use of visual programming. While existing approaches already offer both statistical and visual programming and allow the user to switch between them, StatWire provides the user with a simultaneous integration of both modes. Because of this symbiotic integration, the user can write code in a powerful statistical programming language, but is naturally encouraged to modularize the code he produces in a visual structure. Splitting up the analysis into modules allows the user to efficiently modify and reuse parts of it. Additionally, StatWire provides a visual overview over the main steps in the analysis, making it easier to understand.

This thesis contributes the artifact design for StatWire which is accompanied by a prototype serving as a proof of concept. We explain the significance of StatWire by presenting scenarios that showcase its benefits. To pave the way for future work, we suggest research questions to explore. Other domains can also potentially benefit from the advantages that StatWire brings to statistical analysis.

Acknowledgements

My first thank you goes to my supervisor, Krishna Subramanian, for his patience and support. He offered great feedback on various levels and taught me about statistics, scientific research and chocolate marshmallows along the way.

I also thank Prof. Dr. Chat Wacharamanathan for agreeing to be the second examiner for my thesis and coming up with the new name StatWire. His feedback was very valuable and intriguingly succinct.

I am grateful to Prof. Dr. Jan Borchers who is the first examiner of this thesis. Additionally, I want to thank him and Dr. rer. nat. Simon Völker for having a meeting with Krishna and me that pointed us in a better direction to take this thesis.

Thank you to the members of the Media Computing Group, who provided feedback during the talks and cake during birthdays.

A last thank you goes to the students, research assistants and professor who provided us with some of their code for statistical analysis to pick apart.

Conventions

The whole thesis is written in American English.

No special conventions are used.

Chapter 1

Introduction

Statistical analysis is an essential method for validating experimental findings. Especially null hypothesis significance testing is used across many different domains, in industry as well as research, to objectively judge the results of tests and experiments. This broad spectrum of users results in wildly different requirements for the tools used in statistical analysis.

Statistical analysis has a wide applicability.

In this chapter, we introduce the challenges that the tools for statistical analysis face. This is followed by a short presentation of statistical and visual programming, as these two approaches show a recent trend of growing interest. We explain briefly how our concept manages to bring those approaches into a symbiosis and end on an overview over the different chapters of this thesis.

1.1 Tools for Statistical Analysis

To accommodate the vast differences in requirements, the tools for statistical analysis target very specific user groups. Those range from novices with no experience with statistics to experts with a very deep understanding of the mathematics behind it. Accordingly, many tools provide dedicated interfaces that guide novices through the different

Tools for statistical analysis specialize for different requirements.

Powerful tools are often harder to use.	steps of an analysis while others provide immense adaptability and power to sophisticated analysts. A tool's power is negatively correlated to its ease of use and learnability; improved capabilities require more complex controls and deeper understanding. This relationship causes the most powerful tools to be primarily used by experts who invested time and energy in learning them.
Usability is important for experts, too.	This difficulty in using the tool, however, is not only an impediment for novices who still need to learn it. By exploiting the user's experience, understanding and intuition, improved usability benefits experts, too. It reduces the effort being spent on handling the tool and allows to work more naturally and directly on the subject. An expert profits even more from a well designed tool than a novice, since the expert uses the tool frequently and extensively.
Both power and usability are important qualities for professionals.	While power is a very important factor, usability is crucial as well. A tool should not only allow its user to accomplish a task, it should do so conveniently and intuitively in order to allow for efficient, reliable work.

1.2 Trends

Statistics software loses interest of academics.	Classic statistics software, such as IBM SPSS Statistics or JMP, provides a reasonable compromise between power and ease of use. Their menu-based interfaces expose a set of commonly used functions which suffices for most use cases. Muenchen (2017) shows data indicating that those programs enjoyed very high interest from scholars up until around 2010. The number of articles on Google Scholar about especially SPSS is less than half in 2016 of what it was at its peak in 2010. The data suggests that academic interest in such packages declines.
Statistical programming gains interest.	Those data also show the rising interest in the statistical programming language R, as Google Scholar holds more and more articles about it. If the trend continues, Muenchen states, R would become the most used statistics package for scholarly data science by the end of 2018.

This trend is relevant, because arguably the most powerful tools are statistical programming languages like R. They provide their users complete control over every aspect of the analysis. A big disadvantage, however, is that those languages typically require substantial time and effort to learn. Moreover, as we will discuss in chapter 3, they do not lend themselves particularly well to the characteristics of statistical analysis. Statistical programming languages, while offering immense power, lack ease of use.

While powerful, statistical programming lacks usability.

There is, however, another trend that Muenchen notes on: Software using what he calls a *workflow* control style instead of being menu-driven. Examples are KNIME¹ and RapidMiner² that use visual programming to build analyses out of reusable widgets. In chapter 3 we will compare them to statistical programming and explain why visual programming is particularly well suited for statistical analysis. While they solve the key problems of statistical programming, they themselves lack its power.

Visual programming becomes more popular, too.

The tension between those approaches, statistical versus visual programming, arises because choosing one over the other sacrifices either power or ease of use. Visual programming does not offer the same control over the analysis as statistical programming, whereas statistical programming is more tedious and error-prone than visual programming. Because the advantages of one approach mitigate the other's drawbacks, a combined tool that keeps the best of both worlds would be powerful yet easy to use.

Both statistical and visual programming have advantages over the other.

1.3 StatWire

There exist different projects that try to combine statistical and visual programming that we will introduce in chapter 3. While they provide the user with a free choice between both approaches, they still force the user into only using one of them at a time.

Existing approaches do not combine the programming modes well enough.

¹<https://www.knime.com/>

²<https://rapidminer.com/>

Because the user still has to choose one over the other, the different approaches are not able to complement each other fully and thus fall short of reaping the full potential of a symbiosis.

Our approach integrates both modes to gain both their benefits.

StatWire integrates visual programming tightly into statistical programming. The approaches can complement each other, because the user takes advantage of both at the same time. This allows for a true symbiosis that keeps the power of statistical programming while bringing in the convenience and intuition that comes along with visual programming.

1.4 Overview

This thesis contributes the artifact design for StatWire and showcases its use. We will discuss why it is different from existing research and how it benefits statistical analysis.

Chapter 2 will present research that aims to improve both statistical and visual programming. We will explain how our concept differs or benefits from this work.

In chapter 3 we will discuss how these two programming approaches suit statistical analysis differently and why existing hybrid approaches fail to consolidate their benefits.

With chapter 4 the key idea, components and use cases of StatWire will be explored. It serves to communicate what makes up our concept. We will also discuss our design process when developing StatWire.

Chapter 5 will discuss the benefits of our concept and suggest research questions that serve as a basis for evaluation of those benefits. We will also point to issues we discovered when applying the concept to real-life analyses.

We will conclude with chapter 6 by summarizing the key motivation, components and benefits StatWire provides. Finally, we will point to the different directions future work can go into based on the contributions of this thesis.

Chapter 2

Related Work

A lot of existing research aims to improve on established technology and methods. This chapter will showcase work that tackles statistical and visual programming and briefly explain its relevance to StatWire.

2.1 Statistical Programming

There is active research to alleviate the inconveniences of traditional programming languages. While for example modularity of a program is widely accepted as desirable, it is difficult to measure and define. The inherent difficulty to navigate text efficiently is another issue that is researched.

Modularity Modular software is made up of distinct components, its *modules*. Separating functionality into a self-contained module helps to structure the program in a manner that makes it easier to understand, test and change, because each part can be worked on individually. Since modules are ideally isolated from each other, changes or errors in one part will not influence other parts. All these benefits come out of good modularity and help with maintaining software.

Modularity has many benefits for code.

Because modularity is such a desirable yet hard to objectively measure quality, there is research that tries to define it. One of the most important characteristics of good modular design is a low *coupling* between the modules. Offutt, Harrold, and Kolte (1993) propose a definition and measure of such coupling.

The benefits for code also count for the analysis.

Such work is of relevance for this thesis because the analysis is specified as a program. Improving the code's modularization positively affects the structure of the analysis, which is the key goal of StatWire. As will be discussed in chapter 5, evaluation that tries to validate whether StatWire improves modularity needs a measure of it. Research such as Offutt, Harrold, and Kolte, 1993 provides a foundation for such an evaluation.

Code Navigation Bragdon et al. (2010) present *Code Bubbles* as a method to facilitate navigation in, and understanding of, textual programs. In contrast to classic code editors' file-based interface, Code Bubbles displays fragments of codes (*bubbles*) side by side and provides hints of the references between these.

Navigating code is not StatWire's aim. It provides program visualization for the analysis.

The difference to StatWire is summed up nicely by the statement from Myers and Baniassad (2009) that *program* visualization is different from *code* visualization. We do not primarily aim to improve navigation in the code but rather to visualize the data flow of the analysis. While this is a different goal, navigating the analysis is still easier because of the virtues of visual programming we bring to traditional programming.

2.2 Visual Programming

There is broad variety in the landscape of visual programming and therefore there exist attempts to classify it into taxonomies. While the advantages of visual programming seem clear, it is surprisingly hard to find rigorous quantitative evaluations. However, there are some interesting pit-

falls to acknowledge when using visual programming.

Taxonomy Myers (1990) provides a general taxonomy for classifying visual programming and program visualization systems. He also discusses some general benefits and issues.

A very extensive survey of different visual programming languages was done by Hils (1992). It explores specifically *data flow visual programming languages*, a category to which StatWire’s visual programming component belongs.

StatWire employs data flow visual programming.

Hils suggests alternatives to a *pure data flow model*. Examples of such features are *procedural abstraction* and *distributors*. We propose similar functionality for StatWire in the context of use cases in section 4.3.

Pitfalls Meerbaum-Salant, Armoni, and Ben-Ari (2011) provide interesting insights into the pitfalls of visual programming. During their research with *Scratch* they noticed two problematic techniques students employed.

Students tend to fall into bad habits when using Scratch.

- Bottom-up programming
- Extremely fine-grained programming

They defined bottom-up programming as starting development of basic components which are later linked up to form the system, but their students’ extreme variant of this was described as programming by *bricolage* in reference to Turkle and Papert (1992). Extremely fine-grained programming was defined as the tendency to break down components into their elements beyond the point of logical cohesion, such that the resulting program was fragmented.

We encountered the issue of extremely fine-grained programming to some extent ourselves during our research. We will discuss this issue in section 5.3.

We encountered and counteracted similar habits.

Chapter 3

Tools

This chapter will discuss the characteristics of statistical analysis and how they are neglected or exploited by statistical and visual programming respectively. We will also explore why the approach of existing hybrid systems does not provide a good combination of these programming modes and introduce how a more intimate symbiosis can be created.

3.1 Statistical Analysis

Some aspects of statistical analysis stand in stark contrast to inherent characteristics of statistical programming while those same aspects are naturally exploited by visual programming. We will first present some key characteristics of statistical analysis.

3.1.1 Data-Centered

Statistical analysis is inherently data-centered. Because the goal of any analysis is to understand and evaluate a data set, every action taken revolves around manipulating and transforming data.

Traditional software is concerned with control flow.

In menu based software and statistical programming, however, the system accepts different commands from the user to execute, and is designed around this sequence of commands, the control flow.

What counts in statistical analysis is the data flow.

Yet ultimately it is not relevant to the user *when* he did what action. It is much more important to know *what data* the action was directed at. Whether a plot is based on data before or after transformation is an essential piece of information. Control flow, however, obscures the data flow by focusing on the chronology of actions.

3.1.2 Non-Linear

A closely related aspect is the fact that the data does not flow linearly from one step in the analysis to the next. It is crucial to notice when steps are based on different versions of the data. If for example a visualization is based on transformed data, while a test uses the data before transformation, the analyst has to know about this fact to prevent misjudgment.

The data flow in statistical analysis is often two-dimensional.

This aspect differentiates statistical analysis from other domains, where a one-dimensional data flow would be sufficient. Because of the importance of such non-linear paths in the data flow, statistical analysis requires a two-dimensional approach to its representation.

3.1.3 Structured

An analysis generally falls into a rough skeleton with four different categories of tasks.

1. Data loading
2. Data processing
3. Visualizations
4. Tests

All analyses we encountered and could think of can be categorized in this way. Inside of each category, there are tasks that are common among analyses. Loading data from a file, visualizing it as a boxplot and using one of the different test statistics are all functions that are needed in the majority of analyses. In fact, statistical software such as SPSS and JMP is based on the premise that most use cases are satisfied by the limited set of functions they ship with.

Analyses follow a structure.

While it is important to provide these commonly needed functions, it is impossible to have a complete tool set that satisfies every single use case. Statistical analysis investigates the properties of a unique data set, therefore it needs some flexibility to adapt to the unique case at hand. Providing commonly needed functions is important, but so is the ability to control precisely how the data is treated.

Inside this structure flexibility is needed.

In general, though, these four categories outline an analysis and can be used as a basis to provide a set of commonly needed functions.

3.2 Textual Programming

As explained in section 1.2, statistical programming languages enjoy a steep rise in interest. Examples of such languages include

- R¹
- SAS²
- Python³
- SQL

To differentiate these text-based languages from visual programming, we will use the terms *statistical* and *textual programming* interchangeably.

¹<https://www.r-project.org/>

²<https://www.sas.com/>

³<https://www.python.org/>

Statistical programming provides precise control.

The growing interest can be explained with the power those languages provide. With a complete programming language any specialized algorithm or procedure can be implemented. They are arguably the most powerful tools available for executing statistical analysis.

There are issues with statistical programming.

Nonetheless textual programming has some inherent disadvantages when used for statistical analysis. We will discuss those peculiarities in this section.

3.2.1 Control Flow

Textual programming is control-centered, not data-centered.

Textual programming languages are inherently linear and control flow oriented. Code is just a sequence of commands, which does not fit the analysis's focus on data.

Packages such as `dplyr`⁴, a “grammar of data manipulation” as the title of their website states, tackle this problem and offer a way to manipulate data more directly. But because code is a one-dimensional sequence, it cannot naturally represent the non-linearities mentioned in section 3.1.2.

While this issue does not prohibit the use of programming languages for statistical analysis, it is a discrepancy that hinders intuition and understanding, and obscures the object of interest, the data flow.

3.2.2 Dumping

During our research, we collected a small sample of code written in R from different types of people. The samples were meant to be used as examples to reproduce with our concept, but we noticed a peculiarity present in every single instance.

Analysts treat code structure as an afterthought.

Each of our subjects wrote their entire analysis in a single file with sparing use of functions.

⁴<http://dplyr.tidyverse.org/>

The small sample size hinders the generalizability of this statement, but this habit can be explained by considering the circumstances under which analyses are written. Analysts typically only care about the results of the analysis and often do not have a background in software engineering. Even if they know refactoring techniques they might not consider them worthwhile after having put together code that gets the job done. And because the programming language does not actively encourage a good structure, its users can get by without paying attention to it.

Clean code is important, however, because in statistical analysis the user often has to go back to tweak parameters and change her approach. Lack of a good structure makes this tedious. Additionally, components that can be reused in future analyses are not isolated and therefore more difficult to extract. Sharing and explaining the analysis is difficult as well, since the entirety of the code might overwhelm someone unfamiliar with or returning to it.

Good structure and modularity is important for efficient work.

These issues are solved elegantly by visual programming software.

3.3 Visual Programming

A very intuitive form of specifying an analysis is visual programming. Examples of tools include

- KNIME⁵
- RapidMiner⁶
- IBM SPSS Modeler⁷

Visual programming heavily uses visual metaphors and communicates the data flow clearly. It matches the analysis's focus on data and can naturally represent non-linear

Visual programming fits statistical analysis well.

⁵<https://www.knime.com/>

⁶<https://rapidminer.com/>

⁷<https://www.ibm.com/us-en/marketplace/spss-modeler>

data flow. It usually comes with a set of commonly used functions that fit right into the general structure of statistical analyses.

Visual programming cannot replace textual programming.

While it fits statistical analysis much more naturally and solves important issues of textual programming, it unfortunately is no replacement for it. Visual programming software has the same disadvantage as statistical software like SPSS and JMP; it provides the users with a fixed set of functions.

Often such software allows users to extend its functionality through scripting languages. But implement a new function is completely different from specifying what should happen to the data at hand. This results in a context switch that loses the benefits of visual programming.

Visual programming alone is thus not suited for providing the same control over the analysis that textual programming allows for.

3.4 Hybrids

We found two projects that combine textual and visual programming. This section will present them and show their common difference to StatWire.

Demšar et al. (2004) present *Orange*, originally a framework that integrated C++ procedures with Python scripts. Relevant for our case is the GUI it offered, that allowed using visual programming as an alternative to writing the analysis in code. By the time of writing, Orange⁸ has evolved towards promoting its visual programming capabilities over the C++/Python integration.

Orange provides a choice between textual and visual programming.

With Orange, the user has the choice to program textually or visually and thus can utilize the method most suitable for his case. However, Orange does not allow the user to

⁸<https://orange.biolab.si/>

start an analysis with visual programming and then, upon running into limits, continue it with a textual language.

A system that integrates statistical and visual programming more tightly is *ViSta*, developed by Young and Bann (1997). It allows users to switch between a graphical view and the underlying code at any point. This mitigates the respective disadvantages of the two programming modes somewhat, because whenever a user runs into issues with one of them, she can switch to the other easily.

While this brings textual and visual programming closer together, they still are treated as different representations and interaction modes. When switching from the visual representation to programming in the textual language, the benefits of visual programming are lost. Similarly, when switching from textual to visual programming, the power and expressiveness of the textual language is lost to the abstract visual language.

The difference of these approaches to ours is that StatWire brings together textual and visual programming simultaneously. The user can hence utilize the full capability of a statistical programming language while having the structured overview that visual programming entails. Only through such simultaneity can the two modes work in symbiosis.

ViSta allows to switch between textual and visual programming.

Any separation between the programming modes loses their complementary benefits.

Combining the modes simultaneously makes them complement each other.

Chapter 4

Concept

The key idea of StatWire for structuring an analysis will be introduced in this chapter. Subsequently, we will go through the components that make up the core functionality of StatWire and present some scenarios in which StatWire proves useful. Finally, we will discuss the design process.

4.1 Key Idea

Our concept's key idea is that the analysis be broken down into *steps*. Each step, defined using textual programming, is laid out on a canvas where it can be connected to other steps to form an entire analysis.

StatWire encourages to break down the analysis into steps.

Breaking down the analysis into steps is the key to modularizing the code, making it easier to glance over, understand, implement, reuse, explain and share. StatWire's concept makes this modularization tangible using visual programming and integrates this power into the workflow of textual programming.

Steps make modularization tangible.

While users are anticipated to write many functions themselves, providing a set of pre-made functions is important. In contrast to other software, though, StatWire does not rely on providing a complete such pool, and instead explicitly

Shipped functionality can easily be extended.

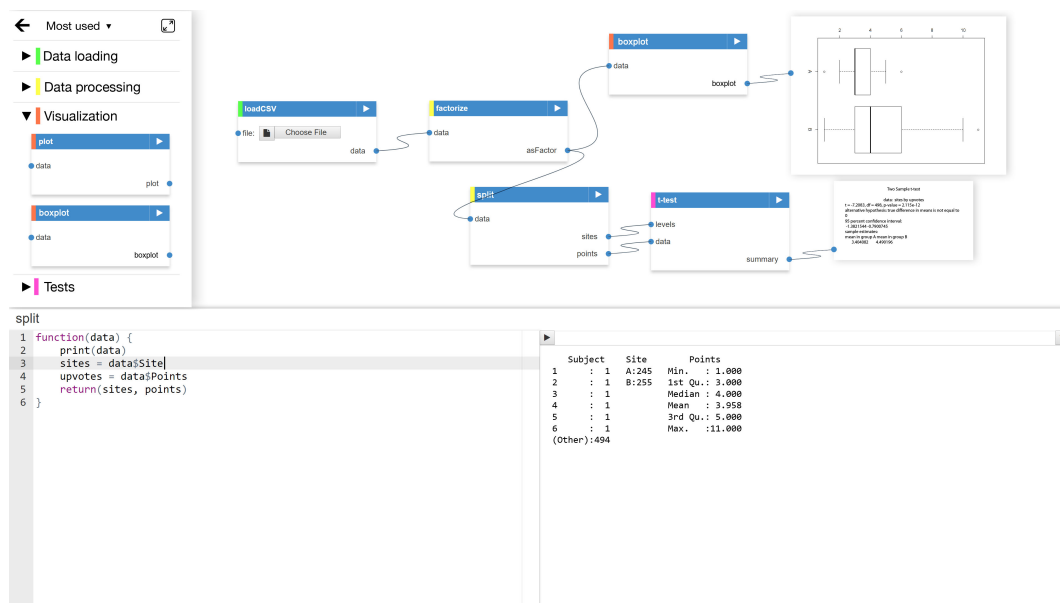


Figure 4.1: A mockup of a session in StatWire showing the components. The main areas are the code at the bottom half, the canvas at the top half to the right, and the node pool at the top-left.

provides the users with integrated means of writing own steps.

StatWire encourages modularization, which has important benefits.

We hypothesize that this focus on steps, especially when compared to traditional textual programming, encourages users to think intuitively about modularization of their code and analysis. This would counteract the dumping habit we observed in textual programming and provides many benefits, making working on and especially revisiting an analysis much more convenient and efficient.

4.2 Components

StatWire is composed of a code area, a canvas and a node pool.

The components that make up the core of StatWire will be presented in this section. They can be broken down into three areas, which can be seen in figure 4.1. The *code* area at the bottom is used for textual programming, the *canvas* above that is used for visual programming and the *node pool* at the top-left provides a collection of nodes to the user.

Figure 4.1 shows an example analysis. The different steps are to load a data set and indicate that the subjects' IDs should be treated as factors to prevent their interpretation as interval data. That cleaned data is then visualized as a boxplot as well as split to be validated using a t -test.

The prototype that accompanies this thesis follows the design presented here, but please note that it deviates slightly concerning some details because of implementation considerations. The prototype is not feature-complete and serves as a proof-of-concept and basis for future development. A table of most important differences between the concept and the prototype can be found in appendix A.

The prototype is incomplete, but serves as a proof-of-concept.

We will now present in more detail the components that make up StatWire. This presentation is structured by the three main areas: Code, canvas and node pool.

4.2.1 Code

The bottom half of the screen is dedicated to the code editor where the user can program in a statistical programming language such as R. The code of a node is a regular function with input parameters and return values¹.

The editor allows the analyst to use textual programming.

Inside this function the user is free to utilize the programming language as usual. StatWire provides the typical setup for development: An editor and a console.

StatWire's editor supports common features such as syntax highlighting, auto-completion and linting. The user can run his code at any time. Alongside the editor, all errors and print statements generated during execution are rendered in a console, allowing the user to debug his code.

¹While e. g. R does not allow multiple return values from a function natively, multiple outputs are essential for a node, in order to e. g. split data into columns. The prototype handles wrapping multi-returns behind the scenes.

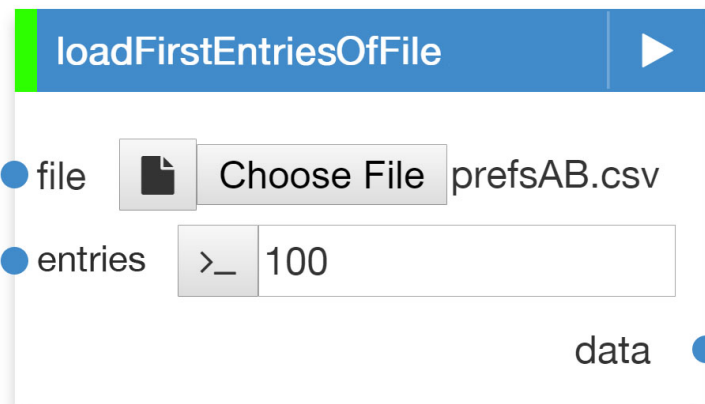


Figure 4.2: A node using both a file and an expression input. The type of input can be toggled by clicking on the button displaying the currently active type.

4.2.2 Canvas

The canvas allows
for visual
programming.

The canvas provides the visual programming interface. It holds the *nodes* that represent the steps of the analysis. The input and output *parameters* of the node can be connected to pass data between different nodes. Additionally, there exist *data viewers* to display graphics or complex objects from the output of a node. We will also explain how to execute nodes.

Nodes represent the
steps of the analysis.

Nodes Each step of the analysis is represented by a node, showcased in figure 4.2. By clicking on a node on the canvas, its code is displayed in the editor below. The parameters and return values of the underlying function are displayed as inputs and outputs, respectively.

The node is color-coded with respect to its category in the node pool. This provides a clue as to what general part of the analysis the node pertains to. Such organization is somewhat important as nodes can be arranged freely on the canvas, though they tend to follow a flow from left to right because of the layout of the parameters.

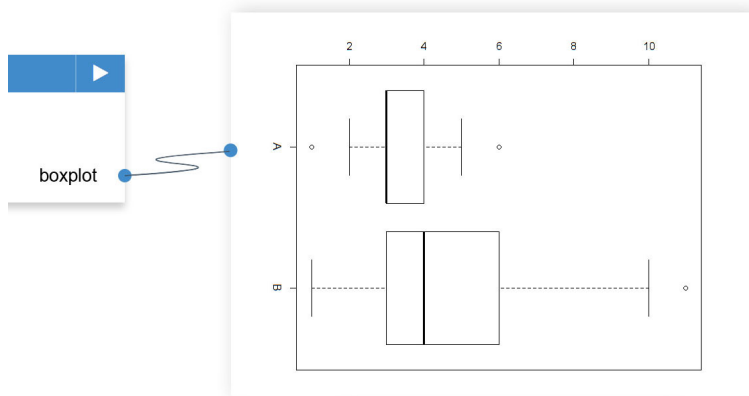


Figure 4.3: A data viewer displaying the graphic from the output of a node.

Parameters The parameters form the interface to the node's function. The inputs' values on the canvas get passed to the function's parameters during execution and its return values are provided as the output parameters of the node.

Parameters are the interface to a node's function.

There are two methods to specify the value of an input. It can be linked to the outputs of another node or specified manually. By linking two parameters together, the value of the output gets fed to connected input. Whenever the linked output changes, the input updates to that new value.

Parameters of different nodes can be connected.

The alternative method of specifying an input is available when the node is not connected. In that case a text input is displayed for entering expressions such as numbers, strings or even more complex statements able to generate entire objects. Those expressions will be evaluated in the textual programming language used by the node and passed into the node's function as input parameters. It is also possible to upload files to a node by switching the input type from a text field to a file browser.

Parameters' values can be specified manually.

Data Viewer To allow the user to see the current value of a parameter, primitive data can be displayed besides the name of the parameter. To examine more complex types

Data viewers display complex data.

such as entire tables of data or graphics, a popup viewer is used as shown in figure 4.3.

A data viewer can be resized and placed freely on the canvas. It can be connected to any output, adapting to display graphical, tabular, or console representations of the data as appropriate.

Execution can be manual or automatic.

Execution StatWire provides two modes to drive the execution of the nodes. In manual mode the user controls exactly when a node is executed by clicking a dedicated button. This is suitable for analyses in which resource intensive calculations are employed.

For most cases, automatic execution speeds up the analysis: Whenever an update to any input of a node occurs, that node will execute again. If its outputs change, this will cause its dependent nodes to be executed, with minimal delay between each other.

4.2.3 Node Pool

StatWire organizes its nodes in the node pool. This provides a central resource where all available functionality can be found. To organize the different nodes, categorization is used.

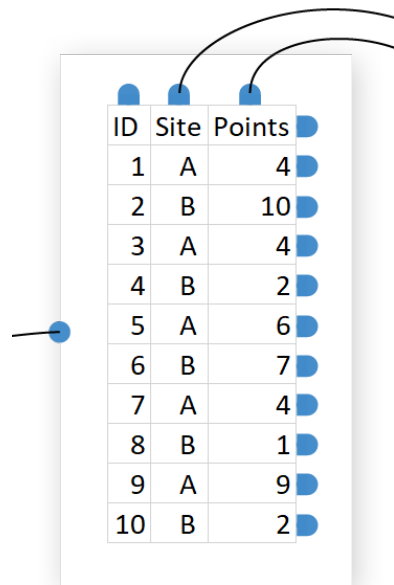
The node pool contains pre-made and custom user nodes.

Node Collection StatWire provides a set of pre-defined nodes that are available in the pool. Instead of adding a blank node, the user can choose from this set which covers commonly needed functions.

Any custom node the user created can be added to this pool. This makes it easy to reuse nodes in a later analysis and helps the user build a tool set that is personalized to her needs, while still easily extensible.

Nodes are grouped into categories.

Categories The default categories for the node are those



ID	Site	Points
1	A	4
2	B	10
3	A	4
4	B	2
5	A	6
6	B	7
7	A	4
8	B	1
9	A	9
10	B	2

Figure 4.4: Selection of the columns *Site* and *Point* from a data viewer displaying tabular data.

presented in section 3.1.3: data loading, data processing, visualization and tests. The user can rename, add and delete categories in order to establish a personalized structure. Such adaptability allows StatWire to comply to the user and her workflow.

Categorizing the nodes by what part of the analysis they address allows the user to choose from a range of relevant functions, she is not distracted by functionality which is only needed later. This allows users to explore the different tools available for accomplishing the next step.

4.2.4 Additional Features

There are some additional features that are not strictly necessary for StatWire to function, but that are useful in common situations. We will motivate their relevance by providing a use case for each.

Data viewers can allow for selecting subsets of tabular data.

Subset Selection Not always does an entire data set need to be passed. Often, a function expects only a specific subset of a table, as can be seen in the sample analysis in figure 4.1: The *t*-test expects two columns of the data not the entire table. While a node can be used as shown to split the data, the exact selection is somewhat hidden in the code and the node's parameters. Providing a mechanism for such selections on the canvas, as displayed in figure 4.4, would show more clearly how the data flows.

For data in tabular form, the viewer allows for selection of subsets such as rows and columns. By putting the same endpoints that parameters use for connections on the header cells of the rows and the columns, the user is signified about the ability to link rows and columns to inputs, just as with regular outputs.

Execution History Because tweaking parameters and exploration is important in statistical analysis, the user has to be confident that he can go back to a previous state and will not lose his work. Freeing him from being careful about changing parts of the analysis provides the confidence necessary for truly free exploration.

Saving the execution history allows for exploration.

StatWire can allow the user to browse through the history of a node. It could store all input configurations used and provide buttons to step back and forth through them. This would retain all the inputs the user tried previously and encourage him to teak them.

For comparing versions of complex data, viewers can be locked and placed besides each other.

In order to compare different versions of graphics or other complex data, the user should be able to duplicate and lock image viewers. Whenever he wants to compare different versions, he creates a copy of a viewer, locks one of them and selects a previous input configuration on the source node. Because the locked viewer retains its data, but the other one updates, the user can compare the different versions.

4.3 Usage Scenarios

Beyond those core features, StatWire can be used in a diverse set of scenarios. The most important of these are presented in this section.

4.3.1 Placeholders

The ability to visualize the data flow of the analysis and still modify each part of the code makes it very natural to reuse analyses in StatWire. This makes StatWire suitable for constructing skeletons for planning or complete templates that can streamline the implementation of analyses.

Planning A user can plan ahead by wiring together nodes representing different steps of the analysis without implementing them immediately. Laying out the dataflow visually provides a high-level overview of the potential analysis and can then guide the implementation.

Users can plan ahead in the canvas.

Such a visual representation is much more suited than the traditional use of comments, because the canvas can naturally represent the dataflow. This is especially important because the desire to plan ahead an analysis suggests that it will be complicated. Representing this complex data flow visually is much more helpful than embedding it in a linear textual program.

Templating Especially when a project requires multiple experiments, the analyses of their results is often similarly structured. Moreover, a researcher might develop a workflow that serves as a framework for future analyses. In those cases, having a template that exploits those similarities while being adaptable speeds up and eases the work.

StatWire allows to create templates that can be easily adapted.

Any analysis laid out in StatWire's canvas can be used as a template. The visual representation serves as an outline and makes identifying the parts that need change easy. The

template can be adapted both by adding and modifying nodes on the canvas or by changing their code.

4.3.2 Education

Because StatWire exploits visual programming to make statistical programming more accessible, it is suited for educational use. The canvas's representation of the steps in the analysis shows the general flow of the analysis while all the details can be accessed in the code that is readily available.

StatWire can be used to teach statistical analysis and statistical programming.

Tutorials In order to teach students statistical analysis and its application, an expert might put together a sample analysis that he provides his students with. When using plain statistical programming, the students might be overwhelmed by the amount of code and will have difficulties acquiring even a basic overview of the different steps involved.

By providing a sample analysis in StatWire, the students can get a first overview through the canvas and then tackle each of the nodes individually. This minimizes the risk that she is overwhelmed by too much information at once. The canvas allows for teaching general statistical analysis while the direct access to the code exposes examples of implementation.

This use case is inspired by ViSta's *guidemaps*, presented in Young and Lubinsky (1995). A notable difference is that ViSta treats the *guidemaps*, interactive walk-throughs, separately from the *workmaps*, where the analysis is implemented. In StatWire, the tutorial is a regular analysis that can be freely manipulated.

Smart hints can support novices.

Smart Hints Each node can provide hints and explanations of its usage based on how it is configured. When for example a node for a *t*-test detects that its input violates the assumption of normality, it can warn the user about this

and provide further instructions on how to resolve the issue.

This is especially helpful for novices learning statistical analysis, but can also serve as a reminder and safety net for experts.

4.3.3 Collaboration

StatWire has many characteristics that facilitate collaboration with other analysts. As well as supporting multiple users working on the same analysis it also can be used in organizations to streamline and unify workflows.

Language-Agnosticism Different researchers might prefer different programming languages for coding their analysis. There exist libraries that enable the communication between languages². Having two analysts with different preferences work together, though, requires them to keep that interface in mind.

StatWire's nodes are generally agnostic to the language they are programmed in.³ Writing any two steps in different languages only makes a difference when accessing their code.

Different programming languages can be mixed.

Repository The Node Pool does not have to be local to each user, it can be shared from a central repository. In an institution this allows coworkers to share their custom nodes with each other.

Custom nodes can be shared.

Additionally, the nodes can be versioned to allow for their controlled development and deployment.

²Examples are RPy (R in Python) and RSPython (Python in R).

³Special types that are specific to a programming language might require conversion to and from JSON. This can be done transparently by StatWire.

Standardization Inside an organization, different people might use different variants of a function or package. This burdens the exchange between coworkers since they have to learn each other's variants. In extreme cases, individuals might use deprecated, obsolete or malfunctioning versions that are inconsistent with the organization's standards of quality.

Organizations can streamline their workflow.

By regulating the pool of nodes provided to its members, an organization can achieve standardization and make their staff's code more consistent. When a member knows a function exists, she will probably just use it instead of reimplementing it. Because StatWire provides a pool of commonly used and also self-implemented StatWire, a user is encouraged to browse functions from a defined source, where they can be controlled and curated by the organization.

4.3.4 Advanced Visual Programming

StatWire is mainly targeted at analysts who are used to the control and flexibility statistical programming entails. There is a potential, though, to expand the capabilities of the canvas in order to exploit more of the benefits of visual programming.

Comments Text can be placed on the Canvas to serve as comment. Such an integrated way of explaining intention and choices that are not apparent from the data flow itself can drastically improve the understanding of the analysis by serving as documentation.

Grouping As explained in section 2.2, Meerbaum-Salant, Armoni, and Ben-Ari (2011) call attention to the tendency of students using visual programming to break down their programs into extremely small components. By cluttering the canvas with too many low-level nodes, the user has to parse details that are not relevant for understanding the

general steps of the analysis, negating the overview benefit. Additionally, the nodes are probably very specific to one narrowly defined task that is unlikely to be used elsewhere unchanged, hindering reusability.

To regain the benefits and counteract such clutter, a hierarchisation using groups is beneficial. A group unites multiple low-level nodes as a single step of the analysis. The user can then choose to hide the internal nodes and use the group just as a regular node. Whenever he wants to change one of the internal low-level nodes, he can reopen the group and access them.

Grouping combats bad habits in visual programming.

In some sense, the difference between a node and a group is that the node is programmed in a textual language while the group is programmed visually.

Control Flow Especially for educational tutorials in StatWire, control flow elements such as an if-construct can augment the expressiveness of the canvas and thus the tutorial. Consider the case in which a teacher wants to show that the choice of test depends on the normality of the data. Using an if-construct that visually indicates such a decision helps to convey this choice directly on the canvas without further need of explanation.

If-constructs can improve templates and tutorials.

By visually distinguishing an if-construct from regular nodes, its use as a control element becomes apparent, contrasting it to the data flow. In its most basic form, the if acts as a switch to reroute data. Which of the outputs the data passes through is visually indicated by fading the inactive paths, where no data is sent to by the if. This clearly shows which paths the data is taking.

While the if-construct is natural in a dataflow model, constructs such as loops are not as trivial. In statistical analysis, the user often goes back to tweak parameters and re-evaluate the data. But the dataflow itself does not normally loop, removing the immediate need for advanced control flow elements such as loops in StatWire.

Loops are not needed.

4.4 Design Process

Our work is based directly on the artifact developed by Ellers (2017) called *StatLets*. In addition to the issues that he identified during his evaluation, we gathered new ideas by presenting the prototype to expert users of R. The resulting list of features and potential issues was prioritized by subjective importance and guided the development of StatWire.

This section will explain the differences to StatLets and our insights from applying our concept to the sample of existing analyses.

4.4.1 StatLets

StatWire is based on StatLets.

Our concept is directly based on the artifact developed by Ellers (2017) which was called *StatLets*. It was developed using iterative design and explored the feasibility of the concept. Because the name StatLets was already taken by another project, we renamed the project to StatWire.

The work on the prototype started with a rewrite of the original program. We took this decision because it allowed us to update the underlying framework to the improved Angular 4 and revise the code. During our work on this thesis we implemented the majority of features from our list and identified several additions that future work can explore and implement.

We fixed issues with StatLets identified in previous work.

Ellers conducted user studies and identified some issues to fix. One of the improvements we made was the way nodes display their parameters. Previously, each node had one input and one output. After connecting two nodes, the user could specify the mapping of output parameters to input parameters using a menu. Because Ellers identified that some users had difficulties with this model, we changed the design. We exposed each parameter directly on the node, similar to Antimony⁴, a node-based generator for 3D ge-

⁴<http://www.mattkeeter.com/projects/antimony/3/>

ometry.

We removed StatLets division of the canvas into four fixed areas for data loading, preprocessing, visualizations and tests. These sections previously served to guide the user through the analysis by keeping nodes that belong to these categories close together. Because we considered this division too restrictive as it forces a certain layout onto the user's workflow, we decided to remove these areas in StatWire. Instead we introduced color-coding as a supplement, such that the user could quickly identify the categories but was free to arrange them as she pleased.

We also removed the interface located around the code editor that allowed the addition and modification of the selected node's parameters, because we wanted to focus the user's attention on the code. All parameters are fully editable in the code and an analyst experienced in statistical programming is used to this function-based interface. Because of these reasons, we left out this additional component.

For brevity, we omit the description of some additional, less important differences between StatLets and StatWire.

4.4.2 Application to Real-Life Analyses

In order to understand how StatWire would be applied in real life, we asked students, research assistants and professors to provide us with some of their code. We additionally utilized code from the learning materials of the online course *Designing, Running, and Analyzing Experiments*⁵ on Coursera.

We gathered a sample of actual analyses' code.

Early on, we tried to represent those analyses in a flow graph on paper. We noticed that it is problematic to accurately depict the data flow of the code literally. For example we encountered a statement that corrected the treatment of numeric columns as numbers that actually represented IDs. Before, R would try to calculate e. g. averages of these IDs

Literal representation of code is too complicated.

⁵<https://www.coursera.org/learn/designexperiments>

and had to be told to treat them as *factors*. The statement simply factorized this column.

A visual representation would be either a loop or a generic transformation. It can be represented as a loop, since the statement literally reads the original column from the data frame, transforms it and stores it back into the same data frame. It can alternatively be understood as a transformation of the entire data frame, because the statement transforms it from one with a numeric column to a data frame with a factorized column. The problem with the loop representation is that it is complex and difficult to even draw. Then again, seeing it as a transformation of the data frame is an abstraction from the actual statement. These problems stem from our initial attempt to visualize the *code*. As Myers and Baniassad (2009) state, this is different from *program* visualization where the intention and process is visualized instead of the specific implementation. This insight lead us to focus more on program visualization.

In practice, code structure is an afterthought.

Thanks to this sample we also discovered the issue of dumping, discussed in section 3.2.2. The sample is not large enough to be truly representative, indeed, but every single piece of code in our sample had this issue. This intrigued us and led to the realization that statistical programming languages do not have mechanisms that encourage attention to structure, one of the key benefits that StatWire provides.

This early experimentation with authentic code therefore not only helped us choose a more appropriate visual representation, but also helped us to discover and formulate some key improvements StatWire brings to statistical programming.

Chapter 5

Evaluation

There is a key benefit that StatWire provides over traditional statistical programming: Encouragement to modularize the code and the analysis.

This chapter will present the aspects that make up this key benefit and also explain the advantages that follow from it. We will give suggestions on research questions that further illustrate the benefits of StatWire and serve as a basis to evaluate the concept. We conclude by discussing some obstacles we encountered when recreating real-life analyses in StatWire.

5.1 Benefits

What separates StatWire from other software that combines textual and visual programming is that traditionally the user can only stay in one of these modes at a time. Our concept exposes both textual and visual programming simultaneously in order to create a true symbiosis between them.

This allows StatWire to keep the power of statistical programming while complementing it with the ease of use of visual programming.

The key innovation of StatWire is simultaneity of textual and visual programming.

5.1.1 Power

StatWire keeps the power of textual programming.

The most important aspects of textual programming that we want to keep are control and adaptability. We will call those capabilities its *power* for brevity. StatWire does not limit what the user can do when compared to traditional programming, because the user actually has access to the entire language.

It is true that other software, such as SPSS or Orange, allows to extend its functionality by using e. g. textual scripting languages. But the significant difference is that they treat the execution of an analysis differently than implementing a needed function. This is where StatWire innovates by keeping the approach of statistical programming languages, where code specifies the actual analysis.

Because it bases everything on the code, StatWire can retain all the power that comes along with textual programming, while improving on some of its disadvantages using visual programming.

5.1.2 Ease of use

Visual programming has great usability.

Visual programming is very intuitive and suits the characteristics of statistical analysis perfectly, thus easing the analyst's work.

The visual representation gives a very convenient overview, allowing users new to the analysis, or return to it, to quickly understand what is happening. It accurately represents the core of an analysis: The flow and transformation of data.

Another great advantage is the very tangible structure visual programming brings to the analysis. Because it is assembled using modules, the user can easily change and reuse parts of the analysis. In statistics the analyst often has to do some amount of exploration and frequently needs to tweak parameters. Making the analysis easy to assem-

ble and change is therefore particularly valuable in this domain.

These two advantages of visual programming mitigate the inconveniences of textual languages. The result is code that enjoys the benefit of better modularization by exploiting the benefits of a visual representation.

Visual programming solves issues of textual programming.

5.1.3 Modularization

The key benefit of StatWire is that it encourages the user to modularize the code. While the code is worked on, the canvas is still displayed. The layout of the analysis is represented there and we hypothesize that this influences the user when he programs.

Because the canvas is so convenient for structuring an analysis into its steps, the user will have an encouragement to think about the components and their links. These considerations are practically absent in traditional textual programming, where the user just writes the program line by line without having encouragement to reflect on the structure.

Bringing both programming modes together encourages modularization.

The simultaneity of textual and visual programming in StatWire brings out these complementary benefits and allows the analyst to write precise code that has an intuitive, visual structure.

5.2 Suggestions for Research Questions

To further illustrate what the benefits of StatWire are and to create a basis for future evaluation, we will present formulations of research questions and suggestions on how they can be tested. These questions correspond to the benefits just described.

5.2.1 Power

Does StatWire actually retain the capabilities of statistical programming?

While it seems obvious that StatWire should provide the same power, because it fundamentally uses statistical programming, this benefit is so essential that it is worth testing for even minor deviations. It could be that splitting up code into nodes has unforeseen side effects or that programmers simply feel hindered by the tight integration of the canvas into the workflow.

Expert evaluation can validate the power of StatWire.

Because assessment of this issue requires deep understanding of the possibilities of statistical programming and benefits greatly from rich experience, evaluation of this research question by experts is appropriate. An expert could get to know and try out the tool by redoing an analysis that required full use of a programming language in StatWire. In an interview, the expert can then explain what he perceives as the differences in power between traditional textual programming and StatWire.

As laid out by Greenberg and Buxton (2008), there is a risk that the interface can get in the way of the evaluation. Usability issues of the prototype could negatively influence the results that actually try to judge the usefulness of the concept. Additionally, the expert might need to get used to StatWire's way of structuring the code, since it offers such a strong encouragement for modularization which he might not be used to. Because these issues can have a significant influence on the data collected, a careful evaluation that follows the advice of Greenberg and Buxton is recommended.

5.2.2 Ease of use

Does StatWire help the user understand the analysis? Does it improve the modularity?

There is little existing research.

The advantages of visual programming seem clear, but there is very little research to validate these effects. Neither

Orange (Demšar et al., 2004) nor ViSta (Young and Bann, 1997) were evaluated in this regard, although Young and Bann suggest similar advantages to visual programming than we do.

There are two different dimensions and four environments to test. The two dimensions are the understandability and the modularity of the analysis. The four environments are textual programming, visual programming, hybrid textual/visual and simultaneous textual/visual.

A reasonable measure of both understandability and modularity is the time it takes to fix a bug in the code, as suggested and employed by Bragdon et al. (2010). This requires the subject to understand what is going on in the analysis to identify potential sources for the bug and it benefits from good modularity to be easily modifiable without introducing new bugs.

The time to fix a bug is an appropriate measure of understanding and modularity.

Alternatively, the understandability can be tested with a questionnaire asking questions about details of the analysis, and the modularity can be evaluated by asking subjects to adapt an existing analysis to another one that is different in only some key steps.

The different environments are difficult to compare since especially pure textual and pure visual programming have different power. The comparison of those requires tasks that are reasonable in visual programming, ignoring the power advantage of textual programming. The comparison to hybrid and simultaneous approaches is easier since they share the same power as textual programming and tasks applicable to textual programming can also be applied to them.

The differences in power hinder the comparison of textual and visual programming.

Our suggestion would be to ask subjects to fix a bug in an analysis using textual, visual, hybrid and simultaneous environments. A between-subject design helps combat difficult to control for differences in the complexity of the bug.

User studies can validate StatWire's ease of use.

5.2.3 Modularization

Does StatWire help programmers structure their code?

Modularization is a key benefit, but hard to validate.

As it pertains to the key benefit of StatWire, this question deserves special attention. The code that is generated using StatWire is anticipated to be modular, whereas with traditional programming there is a tendency for dumping. Proper modularization is the cause for the many benefits discussed in section 5.1.3, and is therefore an interesting subject for measurement.

Measuring modularity of code is not trivial and a part of code quality research. Offutt, Harrold, and Kolte (1993) provide a very detailed measure of coupling between modules. While such a precise analysis is interesting, a more naive assessment of modularity would suffice for a first evaluation.

An analysis from StatWire can be converted to pure code.

An experiment testing this research question will compare an analysis programmed traditionally with one done in StatWire. The analysis in StatWire can be represented as code by translating the links on the canvas to equivalent function calls. This resulting code can then be compared to the traditionally written program.

5.2.4 Acceptance

Do analysts who already use statistical programming accept and integrate StatWire into their workflow?

Even if our concept provides every single benefit we claim, the target users might still reject it. If it forces a workflow that they do not like, is too hard to learn or simply does not appear to them as beneficial, it will not be accepted as a tool.

Ethnography can validate whether StatWire is practical.

Testing this is best suited by ethnographic case studies that explore the real-life use of the tool. Investigating subjects with varying degree of experience with statistical programming should provide insights into whether StatWire

is opinionated. If experts with an established workflow have trouble utilizing StatWire, but people just getting into statistical programming adopt it quickly, this suggests that StatWire's concept breaks with the status quo but might be a viable alternative.

The goal of such a case study should be to understand the long term integration of StatWire into the workflow of analysts.

5.3 Issues

There are a few problems we came across while reimplementing analyses from the code samples we gathered in StatWire.

One issue is that we had a tendency to create nodes containing only few lines. This reminds of extremely fine-grained programming, discussed by Meerbaum-Salant, Armoni, and Ben-Ari (2011). Two symptoms of this problem that we identified are that often nodes only contain a single line, and that too many low-level nodes clutter the canvas.

Many nodes contained only a single call to a function. The problem of manually wrapping a single function in StatWire could be mitigated by automating the task. If the user wants to add a function as a node, StatWire could detect the input parameters and return values of that function automatically. It could then additionally fetch the source code for that function and allow the user to modify it, just as if he had written the node himself. This would incidentally also encourage users to utilize all the functions and libraries available for statistical programming languages.

The phenomenon of clutter might only occur when porting code to StatWire and might disappear when creating a new analysis entirely in StatWire. It could be due to the source code's lack of modularity and focus on low-level details which might make it hard to step back and identify the more general intention behind blocks of code. If this problem persists in the regular use of StatWire, however, there

Often a node only contains a single function. Automation alleviates this issue.

Too many nodes can negate the overview benefit.

Groups mitigate such clutter.

is a mechanism to combat it: Groups, introduced in section 4.3.4, mitigate this issue. Whenever such clutter appears, the proper level of abstraction can be re-established by hiding the low-level nodes away inside of a higher-level node group.

Chapter 6

Summary and Future Work

This concluding chapter reflects on the benefits of StatWire and the contributions of this thesis. We point at different areas of future work that we hope will evolve and improve our concept.

6.1 Summary

StatWire allows a symbiotic interplay between visual and textual programming. It manages to keep the best of those worlds by utilizing them simultaneously and not as alternatives to each other like existing hybrid approaches have.

This thesis contributes the artifact design of StatWire. It builds upon the work of Ellers (2017) and develops the concept further. We also present what we see as the key benefits of StatWire. Additionally, we suggested research questions to validate and ways to go about testing them.

The key benefit is the improved modularization of the code. While the user gets the full advantage of a statistical programming language, he will be encouraged to structure the analysis into its key steps through visual programming. As

a result of this modularization, the analysis will be easier to understand and modify.

6.2 Future Work

A rigorous user study was postponed, in order to focus on the development and definition of the concept. It is, however, highly encouraged that the concept be evaluated and validated. The implementation we provide can serve as a base and the research questions we suggested can guide the evaluation.

Another direction worth taking is to develop the artifact further and adapt it to other domains. While statistical analysis is a perfect fit for the characteristics of StatWire, other fields might benefit from a similar approach.

We hope to have conveyed why StatWire has potential to facilitate programming for statistical analysis. With some additional work, this concept can be turned into a fully-fledged system that actually helps analysts to work effectively and yet conveniently.

Appendix A

Differences between Prototype and Concept

The most important features of StatWire are listed in table A.1 along with an indication of whether the prototype implemented them at the time of this writing.

The prototype does not currently include the additional features from section 4.2.4 nor those presented in the usage scenarios in section 4.3.

Table A.1: Differences between the prototype and the concept concerning the core features.

	Concept	Prototype
Code	Syntax highlighting	✓
	Multiple return values	✓
	Console	✓
Parameters	Connections	✓
	Expression input	✓
	File input	✓
	Short representation of value	✓
Execution	Data viewer	
	Manual	✓
Node Pool	Automatic	
	Node collection	
	Categories	

Bibliography

- Bragdon, Andrew et al. (2010). "Code Bubbles: A Working Set-based Interface for Code Understanding and Maintenance". In: *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10*. Ed. by Elizabeth Mynatt et al. New York, New York, USA: ACM Press, p. 2503. ISBN: 9781605589299. DOI: 10.1145/1753326.1753706.
- Demšar, Janez et al. (2004). "Orange: From Experimental Machine Learning to Interactive Data Mining". In: *Knowledge Discovery in Databases: PKDD 2004: 8th European Conference on Principles and Practice of Knowledge Discovery in Databases, Pisa, Italy, September 20-24, 2004. Proceedings*. Ed. by Jean-François Boulicaut et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 537-539. ISBN: 978-3-540-30116-5. DOI: 10.1007/978-3-540-30116-5_58.
- Ellers, Michael Richard (2017). "Statlets: Improving Statistical Analysis with R". Bachelor's Thesis. Germany: RWTH Aachen University. URL: <http://hci.rwth-aachen.de/materials/publications/ellers2017a.pdf> (visited on 08/13/2017).
- Greenberg, Saul and Bill Buxton (2008). "Usability Evaluation Considered Harmful (Some of the Time)". In: *Proceeding of the twenty-sixth annual CHI conference on Human factors in computing systems - CHI '08*. Ed. by Mary Czzerwinski, Arnie Lund, and Desney Tan. New York, New York, USA: ACM Press, p. 111. ISBN: 9781605580111. DOI: 10.1145/1357054.1357074.
- Hils, Daniel D. (1992). "Visual Languages and Computing Survey: Data Flow Visual Programming Languages". In:

- Journal of Visual Languages & Computing* 3.1, pp. 69–101. DOI: 10.1016/1045-926X(92)90034-J.
- Meerbaum-Salant, Orni, Michal Armoni, and Mordechai Ben-Ari (2011). “Habits of Programming in Scratch”. In: *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education - ITiCSE '11*. Ed. by Guido Rößling, Tom Naps, and Christian Spannagel. New York, New York, USA: ACM Press, p. 168. ISBN: 9781450306973. DOI: 10.1145/1999747.1999796.
- Muenchen, Robert A. (2017). *The Popularity of Data Science Software*. URL: <http://r4stats.com/articles/popularity/> (visited on 08/06/2017).
- Myers, Brad A. (1990). “Taxonomies of Visual Programming and Program Visualization”. In: *Journal of Visual Languages & Computing* 1.1, pp. 97–123. DOI: 10.1016/S1045-926X(05)80036-9.
- Myers, Clayton and Elisa Baniassad (2009). “Silhouette: Visual Language for Meaningful Shape”. In: *Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications - OOPSLA '09*. Ed. by Shail Arora et al. New York, New York, USA: ACM Press, pp. 917–924. ISBN: 9781605587684. DOI: 10.1145/1639950.1640057.
- Offutt, A. Jefferson, Mary Jean Harrold, and Priyadarshan Kolte (1993). “A Software Metric System for Module Coupling”. In: *Journal of Systems and Software* 20.3, pp. 295–308. ISSN: 01641212. DOI: 10.1016/0164-1212(93)90072-6.
- Turkle, Sherry and Seymour Papert (1992). “Epistemological Pluralism and the Revaluation of the Concrete”. In: *Journal of Mathematical Behavior* 11.1, pp. 3–33.
- Young, Forrest W. and Carla M. Bann (1997). “ViSta: A Visual Statistics System”. In: *Statistical computing environments for social research*. Ed. by Robert A. Stine and John Fox. Thousand Oaks, Calif: Sage Publications, pp. 207–235. ISBN: 0761902694.
- Young, Forrest W. and David J. Lubinsky (1995). “Guiding Data Analysts with Visual Statistical Strategies”. In: *Journal of Computational and Graphical Statistics* 4.4, p. 229. ISSN: 10618600. DOI: 10.2307/1390852.

Index

Categories, 12

Components

- Canvas, 22
- Code, 21
- Node Pool, 24

Ease of use, 36, 38

Hybrids

- Orange, 16
- ViSta, 17

Modularization, 37, 40

Power, 36, 38

Statistical Programming, 13

StatLets, 32

Textual Languages

- Python, 13
- R, 13
- SAS, 13
- SQL, 13

Textual Programming, 13

Visual Languages

- IBM SPSS Modeler, 15
- KNIME, 3, 15
- RapidMiner, 3, 15

