Vrije Universiteit Brussel

FACULTEIT WETENSCHAPPEN EN BIO-INGENIEURSWETENSCHAPPEN
Departement Computerwetenschappen
Web & Information Systems Engineering Laboratory

# Intelligent Source Code Visualisation for the MindXpres Presentation Tool

Proefschrift ingediend met het oog op het behalen van de titel Master of Science in Applied Sciences and Engineering: Computer Science, door:

Paul-Cătălin Meștereagă

Promotor: Prof. Dr. Beat Signer
Begeleider: Reinout Roels


JUNI 2014

Vrije Universiteit Brussel

FACULTY OF SCIENCE AND BIO-ENGINEERING SCIENCES
Department of Computer Science
Web & Information Systems Engineering Laboratory

# Intelligent Source Code Visualisation for the MindXpres Presentation Tool

Graduation thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Applied Sciences and Engineering: Computer Science, by:

Paul-Cătălin Meștereagă

Promoter: Prof. Dr. Beat Signer
Advisor: Reinout Roels


JUNE 2014

# Abstract

Presentations are an important medium to share and transfer knowledge. They are used in our daily life in domains such as education, business or even personal leisure activities. Since their introduction in the 1980s, digital presentation tools have not evolved much. The core ideas remain the same. Presentation tools are built around the slide concept, retaining the features of their predecessor the overhead or slide projector. Spreading information over multiple slides makes it difficult for the presenter to transfer knowledge and for the audience to understand it. The visualisation of content became more important than the content itself. Also there is a lack of tools to present special content types such as source code.

MindXpres is a presentation tool that brings a shift of paradigms regarding the creation, sharing and deliver of presentations. It is based on recent research in domains such as hypermedia, spatial hypertext and zoomable user interfaces. The focus is put on content. Information can be presented in its original form, not being spread over several slides, using a zoomable user interface and psychological concepts that enforce spatial reasoning. The plug-in architecture is also an important feature that differentiates it from current presentation tools.

In this thesis, we will focus on the problem of presenting special content such as source code and moreover how to do it efficiently. Through a literature study about difficulties in teaching and learning computer programming, we concluded that some concepts are hard to comprehend just because of students' incapability to create a mental model of the program execution. An efficient mental model is one that uses visuals and shows the user the changes in the source code, such as variable changes, as the program runs. We study the current implementations, approaches and efficiency of such applications. Finally we created an intelligent plug-in for MindXpress to present source code by applying what we learned through our research.

This plug-in is able to detect the specific programming language, highlight syntax, go through the execution step by step, observe variable changes and visualise recursion. All is being done using visualisation techniques and interactivity, giving the ability of the presenter to go to any step of the program's

execution and observe the program's state. This way the audience can build an efficient mental model and have a better understanding of the matter presented.

# Acknowledgements

First of all, I would like to thank my promoter Prof. Dr. Beat Signer and my advisor Reinout Roels for giving me the opportunity to realise this thesis. They were always there to help, give guidance and push the project in the right direction.

Next, I would like to thank my family for their support. To my parents, for encouraging and supporting me to follow this amazing opportunity of studying. Of course, my siblings are not to be missed out in my gratitude.

Finally I would like to thank my friends for making my life beautiful even far away from home and for their support.

# Contents

# 1 Introduction

We use presentations in many areas of our life, from education to business and personal activities. The web-based presentation sharing platform SlideShare has 60 million monthly visitors [1] and it is estimated that each day over 30 million presentations are created [33]. Tools such as Microsoft PowerPoint[1], Apple's Keynote[2] and OpenOffice Impress[3] are familiar to most of us. Evolving form the overhead projector, most presentation tools are built around the slide concept, which limits the presentation to a sequential order. The presenter must spread the information over multiple slides, making it harder to transfer knowledge. The emphasis moves from the content to visualisation and there is almost no support to include special content types into the presentation.

MindXpres is a modern presentation tool that is based on research in domains such as hypermedia, spatial hypertext and zoomable user interfaces. I brings a new approach regarding the creation, sharing and delivering presentations. MindXpres makes a separation between content and visualisation. While creating presentations the focus is put on content. MindXpres tries to resolve problems such as lack of overview, distinction between level of detail or optimal use of spatial reasoning. The navigation between slides shifts from a linear to a non-linear order. Using a zoomable user interface and psychological concepts that enforce spatial reasoning, the information is not spread over several slides and retains its original form. Moreover MindXpres is an extensible framework. Its architecture makes it very different from

[1] http://office.microsoft.com/en-us/powerpoint/
[2] http://www.apple.com/iwork/keynote/
[3] http://www.openoffice.org/product/impress.html

other presentation tools. Based on a plug-in architecture it can be extended very easily, giving the users the power to create specific plug-ins to present special content types.

In this thesis we focus on the problem of presenting special content such as source code and moreover how to do it efficiently. We conducted a literature study through which we identified what are the difficulties in teaching and learning programming. Based on this study we extend MindXpres with an intelligent plug-in that gives the presenter the ability to present source code.

The first question that we put ourselves is "*what are the main difficulties in presenting source code?*". To answer this question we conducted a literature study about teaching and learning computer programming. Through this study we found that there are real problems and difficulties especially for novice students in understanding some concepts. These concepts are hard to comprehend just because of student's incapability to create a mental model of the program execution. This led us to further study the matter in hand and to determine what an efficient mental model is and how mental models contribute to a better understanding. An efficient mental model is one that uses visuals and shows the user the changes in the source code, such as variable changes, as the program runs. We also found out that there is active research in this area. We studied the current research and tools used to help students create such mental models, by looking at their approach and implementation but also at their efficiency. Finally we decided over a set of features that our plug-in will have. As this will be a MindXpres plug-in it must retain all the characteristics of the tool. The user will focus on content, so the plug-in has to be "intelligent" enough to automatically implement its features. The user will insert the code and the plug-in should be able to detect the specific language, highlight syntax, go to variables and method definitions, go through the execution step by step, observe variable changes and visualise recursion. To be able to simulate the execution flow and show variable changes the plug-in will use some specific log files that will be loaded by the presenter in the moment of presentation creation. The plug-in will support multiple programming languages. Because of time limit, for this thesis, we will implement this plug-in just for two programming languages but it's architecture will allow the extension for any other programming language.

As a contribution of this thesis we created a tool to be used during presentations which help novices to gain an effective mental model of the programs execution. Moreover this tool is not limited to one programming language but can be extended to support any programming language. This makes it different from any other tool, which were specifically built for one programming language. We also succeeded to integrate the recursion visualisation

and successfully create a copies model of the recursion execution. Most of the other tools which are dealing with recursion visualisation are specifically built for this purpose and do not offer many other features. Another contribution is also the literature study and the analysis of the current tool used for teaching programming.

In Chapter 2 we present the findings of our literature study that we conducted starting from our problem statement. We find out how presentation are used in education and what are the difficulties in learning and teaching programming languages. We present and analyse some related work and based on the literature study we define the requirements for our plug-in.

In Chapter 3 we introduce the MindXpress presentation tool. We describe the plug-in architecture, which is important to support us in developing our own plug-in.

A description of our plug-in is given in Chapter 4. We describe the plug-in specifications, requirements and its architecture.

In Chapter 5 we describe the implementation and the technologies used. We describe how the plug-in is implemented and explain its integration with the MindXpres presentation tool.

An example scenario for the plug-in is given in Chapter 6. We conclude this Master's thesis in Chapter 7 where we discuss our contributions and and how this plug-in can be extended in the future.

# 2 Literature Study

In this chapter we present the literature study we conducted. The focus of this thesis is to extend MindXpres with an intelligent plug-in that will give the presenter the ability to present source code. The first question that we ask ourselves is "*what are the main difficulties in presenting source code?*". To answer this question we first must understand what are the main problems in teaching and learning programming languages. The findings of our study will be the support for the requirements of our plug-in.

## 2.1 Teaching and Learning Programming

Learning to program is a fundamental part of degree-level education in computer science. Given its importance, teaching programming in an efficient way is still a problem in today's Higher Education [3, 18]. It is also known that students have difficulties in learning how to program [16, 19, 26]. Tony Jenkins argues that the main role of a teacher of programming should be one of a motivator [17]. In many other subjects or areas of computing the teacher is just a communicator of information. Just to present information such as syntax and structure in a lecture is not enough.
A study has been made regarding different programming languages, such as procedural or object-oriented languages [45], which showed that the choice of programming language is influencing the comprehension of programs. Another study [35] showed that, for novice programmers, a strong domain model is good for comprehending object-oriented style programs and that a strong program model is good for a comprehending procedural style programs. One study goes further and investigates individual features of a

language that are causing difficulties to novices [30]. They investigate concepts and topics of object-oriented programming that novices found difficult. The results of this study are very interesting and promising for our work. The topics that proved to be most difficult are those which are relying on a very good understanding of pointers and memory-related concepts. The authors conclude that these topics are difficult just because the student does not have the ability to comprehend what is happening to the program in memory. The main reason is that they are incapable of creating a mental model of the program execution. Most of this dynamic aspects of program execution is presented by teachers using traditional methods such as pen and paper, blackboard [34] or through laboratory sessions. Debugging tools are used to observe the program execution and to watch variable changes in different points of execution, but they are made to be used by experts and make no real effort to reinforce a mental model [38]. Novice programmers can be helped more in this aspect if the instructors would use a clearer approach to teach such topics.

A programming concept which is found difficult to understand by many students is recursion [2, 9, 10, 13, 14]. Recursion is not just difficult to learn but also difficult to teach [8, 15]. A study [20] shows that there is a difference in understanding recursion between novices and experts. Experts are able to conceptualise the unique recursive invocations and the execution flow of a recursive program, known as the copies model. Novices usually adopt the notion of iteration [44], the loop model. The copies model defines a recursive method as a method capable of creating new instances of itself passing the control forward to new instances and back to the suspended ones. In the loop model the recursive method is viewed as a single entity, not a series of instances, which has a start point, an action part and a propagation mechanism. An evaluation [11] has been made to examine if novices who are helped to acquire the copies model of recursion, can effectively use this model. Results showed that a large percentage of novices when using explicit diagrammatic traces are demonstrating an understanding of the copies model but are failing when not using diagrammatic traces. Trying to mentally evaluate the recursive programs novices are showing an evidence for the use of the incorrect loop model. These results are a good evidence that novices have an inconsistent mental model and that the use of graphical representations is necessary to help gaining a correct mental model. The study concludes that teaching novices how to simulate a recursive execution using diagrammatic traces can improve their ability to gain a correct mental model of recursive process. Also there are other studies [7, 37] that are showing the importance of using visualisation when teaching recursion.

## 2.2 Mental Models

When trying to solve a problem by writing a program, usually the user has three main tasks to accomplish. Firstly he must have a goal to achieve, which is usually given in the form of program specifications. Secondly the user must know what the computer is capable of, having a mental model of how computer works. Finally he must follow some steps to use the computer for realising the proposed goal. This involves the steps taken from the abstraction of the problem to the writing of the program.

Computer programming is not an easy skill to learn. Research has been made toward finding effective ways that facilitate the learning of programming [6]. It is generally accepted that having access to a mental model of the system, learning and practicing programming is done much more effectively. How to enhance a novice's mental models became important, therefore research has been made in this area [28,38]. A representation of the components and the operating rules of the system is what we call a mental model and the completeness of this representation may vary [29]. Thus, a mental model can have more or less details of the components and the operating rules of the system and be more or less closely to the real system. The more complete the mental model, the more useful it is in supporting the apprehension of programming. Having a deficient mental model results in an incomplete understanding of how the computer works and will cause the novice programmer to have difficulties with writing correct programs.

In a procedural programming language the program becomes a sequential process. This process is represented by various changes of states. The value of the program variables are defining a program state at a certain location of execution of the program. In a procedural language program states are changing after an expression has been executed.

A possible solution for providing an effective mental model is to use visuals and show to the user the changes in the source code, such as variable changes, as the program runs. transparent to the user as the program executes [6].

## 2.3 Related Work

In this section we will review some relevant related work that has been done in the field of software visualisation with the purpose of aiding novice students to learn a programming language.

### 2.3.1 Bradman

The Bradman [38] tool was specifically created for novice programmers who are learning for the first time a programming language, implemented for the C language. The main purpose of this tool is to emphasise the model of a program and to help users to create a clear mental model of its execution. Bradman works at the statement level. A statement is the smallest independent component of a programming language which defines an operation to be executed. A program is composed of one or more statements and a statement it is composed of one or more expressions. There are plans to extend the Brandman tool to work at the expression level. As each program statement is executed Bradman explicitly displays how the program states are changing. For each program statement it gives a textual explanation about its function and context. Bradman is a window-based tool and in its architecture are included the code window, variables window, explanations window and the input/output window.

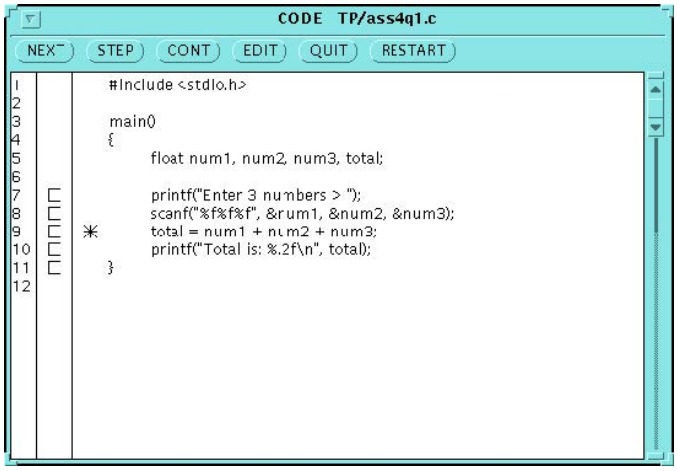


Figure 2-1: The code window.

The code window presented in Figure 2-1 displays the program code and points to the current statement that is executed. The current statement is indicated by the asterisk symbol on the left side of the statement. For longer programs, scroll bars are used to see the entire code. The code window reinforces the model by showing explicitly how in the program state concerned with control location changes after the execution of a statement. The asterisk symbol moves to the new location as the program executes. For novices this can be useful when the new control location is one that they did not expect.

The variables window presented in Figure 2-2 shows how each execution of a statement changes the state of the program variables, thus improving the

programs model. Variables and their values are presented in a list. When a function is called during execution the local variables of that function are displayed under those of the calling function while this is executed. Values are displayed in two columns. The left column contains the variables value before the current statement is executed. The right column shows the variables value after the current statement was executed. The current statement is displayed in between these two columns. With this representation the novice programmer can see the statement as agent who acts on the values of variables in its pre-state and altering the values in its post-state.

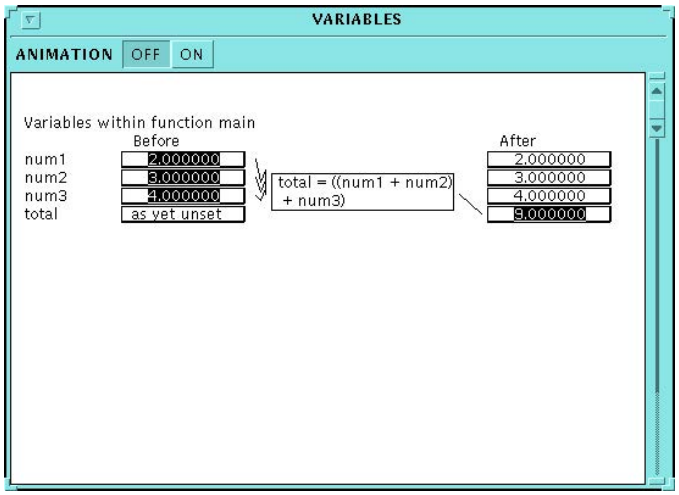


Figure 2-2: The variables window

The explanation window contains textual information about the meaning of each statement. It tells the user how the statement is causing the changes reflected in the code window and in the variables window. The input/output window is mainly straightforward, it gives the user the possibility to insert the necessary input and observe the output of the program. This window makes visible to the novice programmer the input buffer, which is normally invisible, helping him to understand the input process better. An example when this is useful is when the user enters more input than the program is ready for.

We saw how the Bradman tool reinforced the model of how the program is executed, more broadly how the computer produces an output given an input. This is presented as a dynamic process which produces the results by changing the program states. The efficiency of the Bradman tool was also demonstrated in an experiment [39] and the result were favourable. The experiment provided evidence that novice programmers can benefit from the use of such a tool.

As positive aspects of the Bradman tool we found the good visualisation of the execution flow and the display of variable changes. These aspects are very important in creating an effective mental model of the program execution. As negative aspects we found the lack of support for other programming languages or support for other kind of visualisations such as recursion.

### 2.3.2 Jeliot 3

Jeliot 3 [32] is a program visualisation tool that was built with the purpose of helping novice students to learn procedural and object-oriented programming languages. The development process of Jeliot has been research oriented, and it offers a semi-automatic visualisation of the data and control flow of the program. The purpose of this tool is to encourage students to write their own programs and at the same time examine the visual representation of the programs execution. The mental model of the computation is acquired during this process and helps students to better understand how the program works and gain new knowledge.

Jeliot 3 was built for Java, which is an object oriented programming language that is often used as the first language to teach programming. The tool offers the possibility to visualise object oriented concepts such as objects and inheritance.

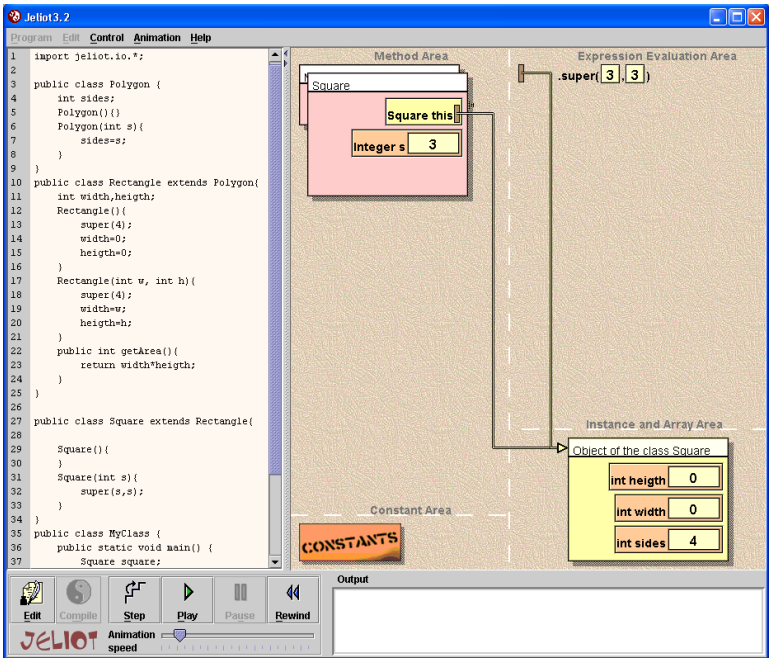


Figure 2-3: User interface of Jeliot 3

The user interface of Jeliot 3 is illustrated in Figure 2-3 and it was built to be simple and usable for novices. The menu bar is used during editing but during visualisation it is taken away to create more space for the source code viewer. The visualisation is controlled by a VCR-like menu and is displayed in the right-hand side frame. The output of the program is printed in the output console and the input requests are shown in the visualisation frame. If an error occurs during execution the error viewer displays the reason and the line of code that produced the error is highlighted.

The visualisation frame was built to be as explicit and consistent as possible and reduce the cognitive load of the student. As presented in Figure 2-4 the visualisation area is divided in four areas, each visualised component always appear in its specific area. All expressions are evaluated and all the values are displayed so that the student does not have to guess where the values are coming from. The cause and effect are easily identified by the link between visualisation and code highlighting. The elements in the visualisation form are shown in an UML-like notation. The objects are displayed as boxes that contain attributes and their values and the references are displayed as lines connecting the object with the corresponding variable. An object can have multiple references at any moment.

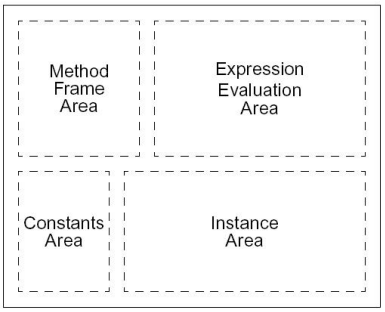


Figure 2-4: The structure of the animation frame in Jeliot 3

Jeliot 3 is a tool that provides clear semantics and engages students into the learning process. It helps students to create a mental model of the programs execution. After the evaluation of the tool, the results were favourable and students get benefits form using Jeliot 3 [31]. However this evaluation concludes that this tool is not flexible enough to support the different students skills and the task they are performing (e.g. comprehending or debugging), and that would imply the need to model the student by helping them with extra learning material.

To conclude, we found Jeliot 3 as a good tool to visualise the execution of object-oriented programs. It displays well the execution flow and the ob-

ject instances. Also it was built with extensibility in mind, so it can be extended to support other visualisations such as recursion. Besides these positive aspects we found as negative aspects the lack of support for other programming languages and the tight flexibility of the tool to support different student skills.

### 2.3.3 Notational Machine

In this section we will present a tool that uses the notional machine [4] [5] and was built with the purpose of teaching introductory programming and helping novice programmers to understand programming and its dynamics. The implementation has been done as an extension of the BlueJ [24] environment and visualise the execution of Java programs in real time. The model introduced does not work at the level of simple statements, but only at the level of objects, classes, methods and the call chain between them. It also shows objects state.

Figure 2-5 shows a diagram for a simple program. The peach coloured rectangles are classes and dark red rectangles are objects. The references are represented by arrows. This is a simple representation but clicking on the object the user will get an expanded view, which shows the state of its fields and references. The expanded view is presented in Figure 2-6. Also a use case is the visualisation of recursion. The arrow will point back to itself and a number showing the count of recursive calls is displayed near. We find this kind of visualisation for recursion ineffective because strengthens the loop model of recursion, which is an inaccurate model.

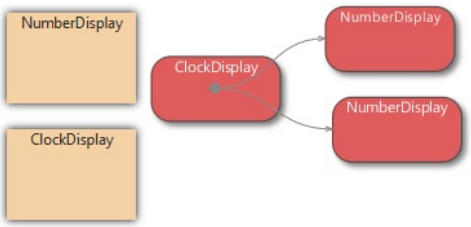


Figure 2-5: Diagram of a simple program [4]

To invoke a method the user right clicks an object and selects the method from the pop-up menu. During execution just the currently active methods are shown in the diagram. The call sequence is animated, and the user can control the animation speed. The currently active method is highlighted so the user can visually follow the execution. Such a call chain is presented in Figure 2-6.

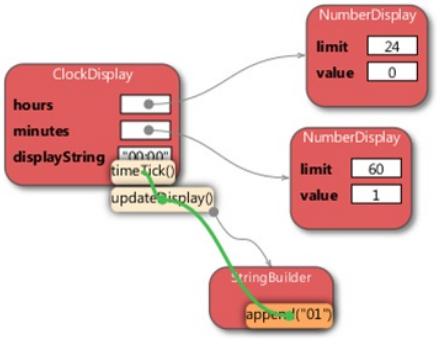


Figure 2-6: An object diagram with overlaid call sequence showing a chain of method invocations [4]

The user interface of this tool is presented in Figure 2-7. Speed and stepping granularity can be determined by the speed slider. Also the level of detail can be specified by the detail slider. Depending on user's intents a range of details can be included or excluded. Another important component of this tool is the heat map view, which was optimised for large programs with hundreds of objects. Heat map view provides the least level of detail. As the program executes the object color changes, going warmer as more methods are invoked and cooling as methods finished executing. Using this view the user can observe object creation and destruction and activity hotspots.

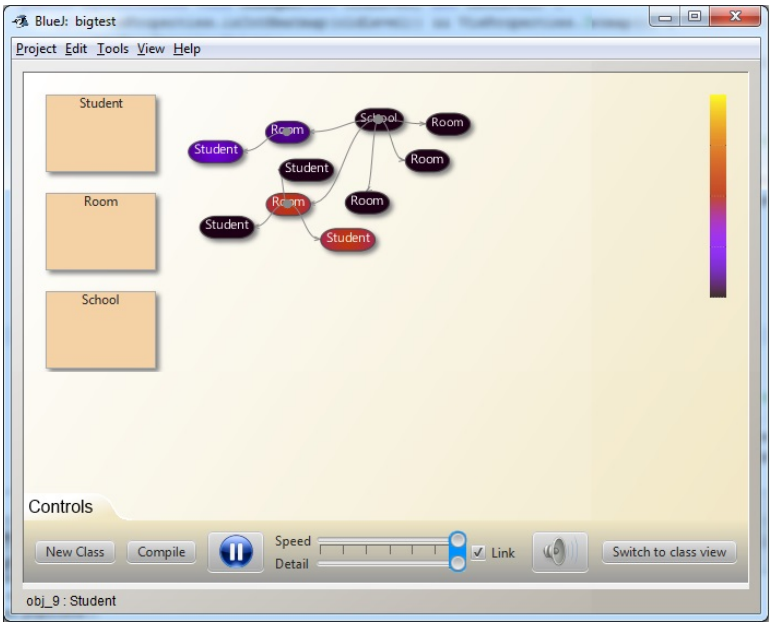


Figure 2-7: The heat map view while the program is executing [4].

This tool is very recent and an evaluation regarding the efficiency and usability of the tool has not been done. The authors are planning to do that. We presented this work because it is very recent and it demonstrates that this is an active field of research that still needs solutions and innovation. A positive aspect of this tool is that it gives a good overview of the object relations and their method invocation. Also another positive aspect is the heat map display. There are also negative aspects such as the lack of source code visualisation, the inability to work at the level of simple statements and the lack of support for other programming languages. Also another negative aspect is the limited interaction with the execution flow, the program can just be executed forward by an automatic animation or paused and it does not allow the user to easily go to a certain step in the execution flow. Another negative point is the recursion visualisation which strengthens the loop model of recursion, which an inaccurate model.

### 2.3.4 RGraph

The RGraph tool [37] uses recursion graphs to visualise recursion for Java programs. Recursion graph representation is based on the recursion trees. The purpose of this tool is to help novice programmers to understand recursion. RGraph can be used to generate the complete or partial recursion graph of a program. A complete recursion graph displays the entire execution flow and a partial graph displays the recursion process at a certain time in the execution flow. Partial graphs are used to test the students ability to comprehend the recursion by making them to think what is missing in the graph. Another feature of RGraph is the traceability, the call sequences are traceable, which means that recursion graphs displayed are directed graphs.

The user interface of RGraph is presented in Figure 2-8. We can observe that the user must specify the directory where the program is found and the methods to be traced. The user also must specify the percentage of missing labels, 0 percent meaning a complete graph. Clicking the 'Generate Graphs!' button, the graph is displayed. In Figure 2-9 can be observed a generated recursion graph. The oval nodes represent a recursion call and the square nodes represent a pre-processing or a post-processing statement before or after the recursion call.


Figure 2-8: RGraph User Interface [37].

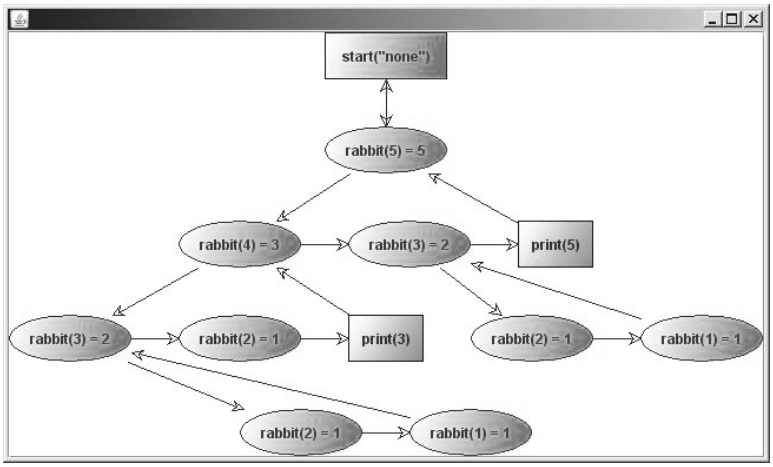


Figure 2-9: Recursion graph representation [37].

An evaluation of the tool was made by surveying students before and after using the RGraph tool. The students found the tool useful to understand recursion and the results of the survey were in favour of RGraph.

A positive aspect of this tool is the good representation of the recursion through recursion graphs which strengthens the copies model of recursion. This tool is specifically built for recursion visualisation and does not support any other features. We consider the lack of interactivity, the visualisation of source code and variable changes as negative points of the tool.

### 2.3.5 VIP

The VIP [43] tool is a visual interpreter for C++ used to teach introductory programming. It is part of the Codewitz [25] project, an international project with the purpose of creating free visualisation tools for teaching programming. Even though is was developed for C++ it lacks the features of object oriented programming and the programming language is restricted to a subset of C++.

The user interface of VIP is divided in different sections and it can be observed in Figure 2-10. The sections are the following: instructions, control, source code, evaluation, variables and output\input.

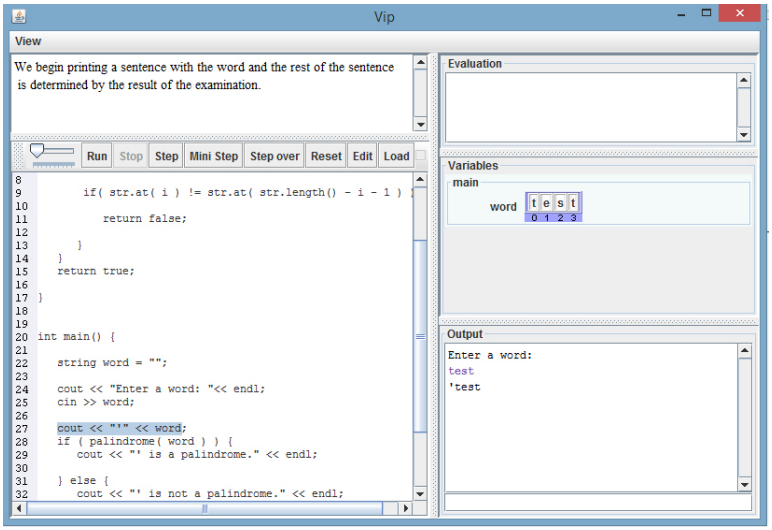


Figure 2-10: VIP user interface.

The instructions section will display the instructions of the program. For each line of the program another instruction can be displayed. The instructions section also can render and display HTML code.

The control section includes some buttons that allow interaction with the execution of the program. The program can be executed step by step or continuously at a certain speed specified by the user. The user can step forward but also backwards in the execution flow of the program. The program source code is displayed in the source code section. The current line that is executed is highlighted. The evaluation section displays details about the operators evaluation in the current statement. Thus the operator precedence can be observed. The user can see the evaluation of the statement not just its result. The variables and stack can be observed in the variables section. The variables are displayed dependent to their subroutine. To display pointers and references are used arrows that point to the relevant variable. Variables are displayed in a coloured box, the color indicating if the memory was read or written at that point of execution. The user interface also includes a section that displays the output of the program while it is executed. Also includes a section where the user can give input to the program.

A positive aspect of this tool is the representation of memory pointers. They succeed at giving a good representation of the pointers and references. Another positive aspect is that the tool provides visualisations of the statement evaluation. Although it was built for an object-oriented programming language it does not support all its features moreover just a subset of the C++ language. We consider this as a negative point. Another negative points can be the lack of a graphical representation for recursion and support for other languages.

### 2.3.6 Current presentation tools

The current presentation tools such as Microsoft PowerPoint or Apple's Keynote do not have an easy way to present source code. The content must be divided over different slides and formatted text cannot be displayed using a bounded box with scrollbars. The syntax highlighting of the code must be done manually because there is no support for such action. Moreover if we want to have interaction we must create an animation in which we have to manually create each frame, all divided on different slides. To create a visualisation that allows you to present source code, interact with the program execution, view variables and display a graphical representation of a recursive algorithm, is simply impossible in the current presentation tools. The current presentation tools are very limited regarding the presentation of specialised content.

### 2.3.7 Overview

In this section we have seen a review of some related tools used for teaching programming to novices with the purpose of strengthening the mental model

of the program execution. We saw the positive and negative aspects of each tool and to recapitulate we present an overview by comparing the tools based on the following criteria:

- Source Code: Does the tool displays the source code as it executes?
- Variable Changes: Does the tool displays the variable changes?
- Interactivity: Does the tool support interactivity with the execution flow?
- Procedural: Does the tool support procedural programming?
- Object-oriented: Does the tool support object-oriented programming?
- Recursion: Does the tool support an effective recursion visualisation?
- Multiple Languages: Does the tool support multiple programming languages?
- Extensibility: Can the tool be extended to support different programming languages?

We can observe in Table 2-1 that each of these tool focuses only on specific aspects and each one has some limitations.

| | Source Code | Variable Changes | Interactivity | Procedural |
|---|---|---|---|---|
| Bradman | ✓ | ✓ | ✓ | ✓ |
| Jeliot 3 | ✓ | ✓ | ✓ | |
| Notational Machine | | ✓ | | |
| RGraph | | | | |
| VIP | ✓ | ✓ | ✓ | ✓ |
| | Object-oriented | Recursion | Multiple Languages | Extensibility |
| Bradman | | | | |
| Jeliot 3 | ✓ | | | |
| Notational Machine | ✓ | | | |
| RGraph | ✓ | ✓ | | |
| VIP | | | | |

Table 2-1: Comparison between different tools.

## 2.4 Better Visualisations

As this thesis focuses on presentations, we also conducted a literature study regarding presentations and information visualisation to enhance the display and the outcome of our plug-in. Nowadays teachers are using interactive presentation tools. Research has been made to measure the influence of these interactive presentation tools on teachers pedagogy [23]. This is influenced by many factors but one of them is the resources that are available to the teacher. We want our plug-in to be an efficient resource. Using traditional presentation software, such as Microsoft PowerPoint or Keynote, it is extremely difficult, if not impossible, to create such a representation that we propose. Also the drawback of such slidewares are well known [40].

Our plug-in must help learners to enhance their mental model about how the presented program works and we think that use of animation and interactivity is necessary. Dynamic visualisations are becoming more and more frequent in education. Lowe shows that interrogation of a dynamic visualisation during learning [27] can be effective. Animation, as a dynamic visualisation, has the capability to help learners to build consistent mental models of complex processes.The use of animation can facilitate [41] in many situations such as showing change over time. A real change in a process should be natural to transpose it in a metaphoric change during a presentation. There are many forms of animation, which are determined by the information to be transmitted. Moreover interactive animations offer the possibility for learners to have a selective view of information that way avoiding excess of information. So from animation we are moving to interactivity.

The difficulties of understanding and comprehending a process can be exceeded by using interactivity, which is known to facilitate learning. Being able to start, stop and replay an animation can allow reviewing or focusing on certain steps within animation. Using close-ups, zooming, alternative perspectives, and control of speed are even more likely to facilitate perception and comprehension. Interactivity may be the key to overcome the drawbacks of animation as well as enhance its advantages. [41]

We previously mentioned that the role of a teacher should be more than one of a presenter. His role should be one of a motivator. Kelleher and Pausch showed that storytelling can be used to motivate programming [21]. Also Gershon and Nahum showed that storytelling can have an important role in information visualisation [12]. Stories are used to transmit information, cultural values or experiences. From the invention of writing to the printing press until today, technology has always provided new means to tell stories. Now computer having an important role in our lives, storytelling is adapting to our computerised world. They are stating that "*a story is worth a*

*thousand pictures*" and you can present information in a story-like fashion. To do that, they present some key steps such as building the big picture, animating the events, resolving conflict and ambiguity. Also they introduce the 'comic metaphor' in which information is presented side by side, in a time line, such as in a comic book.

## 2.5 Requirements

Having this literature study as basis, we define some requirements for our plug-in. Requirements are conditions that are needed to be accomplished in order to achieve an objective or to solve a problem [22]. From our literature study we have concluded that for novice programmers the most important thing, towards better comprehension, is having a good mental model of how program works. We also concluded that an efficient mental model is one that uses visuals and shows the user the changes in the source code, such as variable changes, as the program runs. Moreover we have seen that recursion can be a difficult concept to grasp, and that it is better comprehended when visualisation is used. Based on that, we define the following requirements:

**R1. Mandatory:** The plug-in must support multiple programming languages [45]

**R2. Mandatory:** The display of the source code must be done in an efficient way and not be affected by its length [40]

**R4. Mandatory:** The plug-in must be able to display the flow of the program [6, 28, 32, 38]

**R5. Mandatory:** The plug-in must be able to display changes of variables values [6, 28, 32, 38]

**R6. Mandatory:** The plug-in must support interactivity, the user must be able to go at any step in the execution flow. [23, 41]

**R7. Mandatory:** The plug-in must respect the MindXpres guidelines and be content oriented. The visualisation must be automatically generated. [36]

**R8. Recommended:** The plug-in must be able to graphically display recursion [7, 11, 37]

**R9. Optional:** The plug-in must allow navigation within the code [41]

**R10. Optional:** The plug-in must be able to show the program output [6, 28, 32, 38]

We have defined some mandatory requirements that we think are absolute necessary for our plug-in. They are essential in helping the audience to understand how the presented program works and to create a mental model of its execution. The recommended requirements will enhance the plug-in and will increase its capabilities. The optional requirements are not vital for our plug-in but certainly they will give a plus. In the scope of this thesis, since we are time constrained, the optional requirements will be implemented if the time will allow.

## 2.6 Conclusion

In the previous sections we presented the findings of our literature study. This form the basis for the requirements that we defined for our plug-in. We have seen that teaching and learning programming is not such an easy task. For novice programmers the most important thing, towards better comprehension, is having a good mental model of how computer works. Specific to our problem we have seen that an effective mental model is one that uses dynamic hints that make transparent to the user all the changes in the variable values, source code and output as the program executes. We presented some related work and their efficiency, that gives us support for our work. Also we showed that using visualisation techniques such as animation and interactivity we can enhance the outcome of our plug-in.

In the next chapter we introduce the MindXpres presentation tool. We describe the plug-in architecture, which is important for the development of our own plug-in.

# 3 MindXpres

In the previous chapter we presented our literature study, through which we identified some of the problems that novice programmers are facing. Based on this study, we extend the MindXpres presentation tool. In this chapter we introduce the MindXpres [36] presentation tool, which will be used to implement our plug-in.

MindXpres is a modern presentation tool that steps away from the classical slide concept. It changes the way we interact with presentations bringing new features that enhance the flexibility of the tool and are solving some of the problems of the common tools such as Microsoft PowerPoint, Apple Keynote or OpenOffice Impress. MindXpres tries to resolve problems such as the lack of overview, distinction between level of detail or optimal use of spatial reasoning. The navigation between slides shifts from a linear to a non-linear order. Using a zoomable user interface and psychological concepts that enforce spatial reasoning, the information is not spread over several slides and retains it's original form. With MindXpres the presentation is becoming more audience-oriented, while the common sildeware are presenter-oriented by facilitating the content creation and the whole aesthetic part of it, increasing the ease of use for the presenter. MindXpres is an extensible framework. Its architecture makes it very different from other presentation tools. Based on a plug-in architecture it can be extended very easily, giving the users the power to create specific plug-ins to present special content types, without any loss of ease of use offered by the common slide-ware.

Within the architecture of MindXpres the separation between content and visualisation is made very clear, similar to a LaTeXdocument. The creation of the domain-specific language that focuses on content is handled by a graph-

ical editor and the visualisation is done by the compiler tool based on a template. The output is an HTML5 document which makes portability and distributability possible across different devices.
The plug-in mechanism of MindXpres offers users the possibility to present special content types by adding new components which facilitate that. Advanced features such as non-linear traversal of the presentation, hyperlinks, transclusion, semantic linking and navigation of information, multimodal input, dynamic interaction with the content and the import of external presentations now are available with MindXpres.
In the following sections we will have a closer look to MindXpres architecture and mention the features that are helping us in the development of the plug-in.

## 3.1 MindXpres Architecture

In the following paragraphs we will briefly describe how MindXpres is built. The architecture of MindXpres consists of three components as is presented in Figure 3-1. First component is the XML Container Format which is used to define the content to be used in the presentation. The second one is the Compiler which transforms the XML document into an HTML document. The last one is the output format with the visualisation layer.
To create a presentation a user has two options. He can write the content of the presentation directly in XML or using a WYSIWYG Editor that could generate the XML for him. The compiler is processing the XML file and outputs the presentation in the selected format.

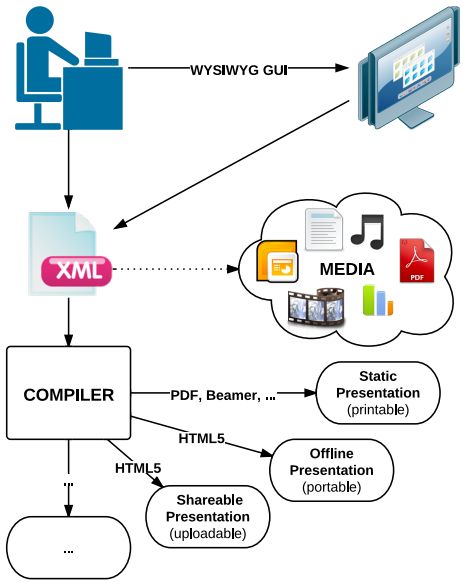


Figure 3-1: MindXpres architecture (Source: R. Roels, B. Signer [36])

### 3.1.1 XML Container Format

To shift the focus from visualisation to content, MindXpress uses the XML language, which provides a semantic interface to define presentations. An XML schema is used to validate the XML document. We can see in the following an example of an XML document:

```
<presentation theme="vub">
  <slide title="foobar">
    <image src="foobar.jpg">
    <bulletlist>
      <item>item1</item>
      <item>item2</item>
    </bulletlist>
  </slide
</presentation>
```

This XML file will generate a presentation which contains one slide. The slide has a title and includes an image and a bullet list with two items.

### 3.1.2 Compiler

The role of the compiler is to validate the XML document based on the XML schema, to parse it and transform it into valid HMTL5 document. The XML document used in the previous example will be transformed by the compiler in the following HTML code:

```
<div data-type="presentation" data-theme="vub">
  <div id="element_1" data-type="slide" data-title="foobar">
    <img width="300px" src="foobar.jpg">
    <ul>
      <li>item1</li>
      <li>item2</li>
    </ul>
  </div>
<div>
```

## 3.2 Plug-in Mechanism

MindXpres was built to support extensibility. This is done through the plug-in mechanism. Most of the presentation tool are very limited when it comes to expandability. If you want some feature that the tool does not support it is often difficult or even impossible to add that feature by yourself. MindXpres changes that, because it does not have a core with hardcoded components and aesthetics. The core presentation engine of MindXpres is seen as a plug-in framework. Nothing is hidden internally and most of the components are developed as plug-ins. If you want to add new features, it offers a mechanism to do that by plugging in any feature in a user-friendly way. You have access on a very low level, being able to modify or ad functionality. Plug-ins are developed in JavaScript and are interacting with the presentation content through a specific interface. To add a new plug-in is as simple as placing it in a specific folder.
This plug-in based architecture of MindXpres is most important to us because it allows us to expand the presentation tool by creating a plug-in to present source code. The plug-in will automatically recognize the programming language, highlight the syntax and adding scroll bars for longer code. Moreover its functionality will be much more complex by providing to the presenter the possibility to interact with the code by showing an execution flow and variables states. All this with the purpose to better illustrate its execution so that the audience can have a better mental model of that. MindXpres is based on three major types of plug-ins such as components, containers and structures. They will be discussed in the following subsections.

### 3.2.1 Components

Plug-ins which provide visualisations and functionality for a specific content type for different content containers are called components. Examples of such components include images, video or source code in our case. In MindXpres every content container is a plug-in. The role of the plug-in is to decide how

to display its content and how the presenter can interact with it. We can give a good example with a video plug-in. For instance, a plug-in component that provides video uses flash player. However, one want to use the HTML5 component and so he can modify the current plug-in or create a new one. Our plug-in will be a component plug-in which will decide how to display the source code and how the presenter will interact with it, also providing the means to do that.

### 3.2.2 Containers

Containers provide functionality to visually organize components. Shortly, containers are elements that contain components. The best example for a container is a slide. Each slide can contain different content but usually they contain some reoccurring elements such as slide title, slide number or author's name. The Simplification of users work is done by abstracting these common elements to a higher level. A container can help user to lay out the content by defining presets or allowing definitions of layouts. Also a container can also contain other containers, not just components.

### 3.2.3 Structures

Structures allow to lay out components on a larger scale. For instance, in a Zoomable User Interface, to display the element in a grid. The difference between structure and container plug-ins is that structure plug-ins have ties to the XML language that defines the presentation. For example as LaTeXprovides structure in documents such as chapters, sections and subsections, the structure plug-ins can define structures that can be used from the XML language. This is mostly necessary for complex visualisations. Structures are more complex and on a higher level than containers.

## 3.3 How to Use

As we previously stated to add a new plug-in is as simple as placing it in a specific folder. To keep things clean all the plug-ins are contained into individual folders. The plug-ins folder includes all JavaScript and CSS files, images and other relevant resources. In this way the management of the plug-ins is done very easily, simply by adding or removing plug-in folders in the plug-in directories. MindXpres also has some specific naming conventions, that enables the core to know what to load and how to access the loaded information. Every plug-in must contain a file called `plugin_info.js` which contains some plug-in-specific information, such as the tags that it provides for the XML authoring language but also the name of the main plug-in object, which is initiated by calling its `init()` method. In Figure 3-2, the structure of a video plug-in folder is presented.

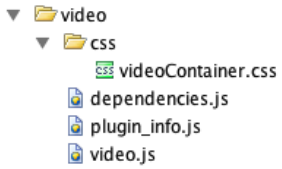


Figure 3-2: The plug-in structure (Source: R. Roels, S. Beat [36])

## 3.4 Conclusion

This chapter was an introduction to MindXpres. We have seen how MindXpress solves some problems of the traditional presentation tools and described its architecture. The most important part for us is the plug-in mechanism that gives the possibility to add, modify, and remove features in an easy way. This mechanism will allow us to extend MindXpres by creating a component plug-in.

# 4 Source Code Plug-in

In the previous chapter we have seen the plug-in architecture of the MindXpres presentation tool. MindXpres in its basic version does not support complex visualisations but plug-ins are used to extend the basic functionality of MindXpres to support such complex visualisations. This thesis will consist of extending the capabilities of MindXpres with a plug-in that supports the visualisation of source code and program's execution. Using this plug-in, a user will be able to present source code and moreover a program's execution with little effort.

In this chapter we will describe our plug-in. Having our literature study and the requirements we defined as the basis, we establish the specifications for our plug-in. Requirements are telling what the plug-in should do and the specifications are telling how the requirements will be accomplished in the implementation.

## 4.1 Specifications

A specification describes how the implementation must be done in order to achieve a requirement [22]. In the following paragraphs we will describe the specifications for our plug-in.

The plug-in must support multiple programming languages (requirement R1). This is an essential requirement of our plug-in because of multiple existing programming languages. In order to achieve this requirement the plug-in must support extensibility for any programming language. One should be

able to easily write an extension, that will extend the support for another programming language. Thus in the implementation we must take this into consideration and provide support for the future extension of the plug-in.

The current slideware tools are not supporting the presentation of source code. Large amounts of text are not well displayed and must be scattered between different slides. Our plug-in must allow the display of the source code in an efficient way and not be affected by it's length (requirement R2). This requirement can be achieved by the following specifications. If the length of the source code is larger than the current view, then scrollbars must be used. Also to have an efficient and more natural display of the source code, this must be displayed in a formatted way and with the syntax highlighted. In Figure 4.1 we can see a possible implementation of these specifications.

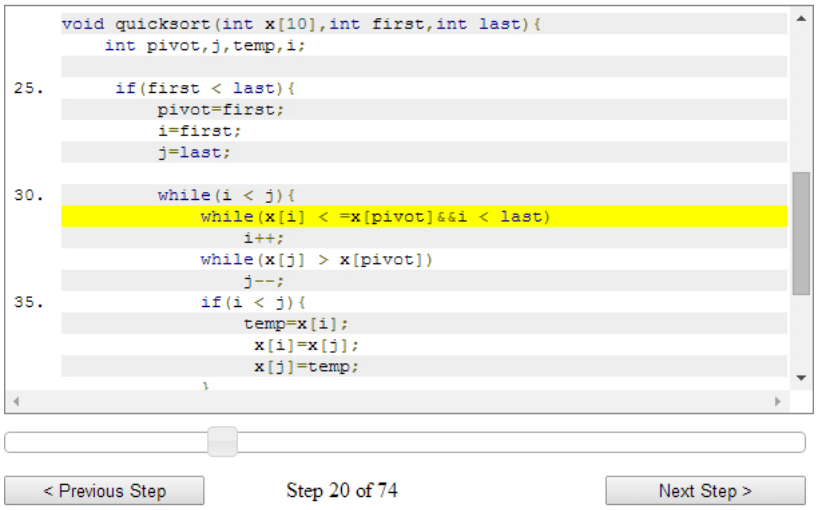


Figure 4-1: A possible view of the source code

To create an effective mental model we must use dynamic hints as the program executes, thus we must be able to display the flow of the program (requirement R3). To achieve this requirement we can use animation. During the animation the current line which is executed must be selected and emphasized by using some visual clues. The number of the current line should be also displayed. An example of this specifications can also be seen in Figure 4-1. We observe the current line which is in execution is highlighted in yellow, on the left side we have the line numbers displayed and at the bottom we can see the step number in the execution flow.

As in an effective mental model, while the program is executed we must make transparent to the user the changes in variables value (requirement R5). Thus while the animation of the program flow is running, for each line that is executed we must display the variables values. For each variable we display the old value that it had before the current statement was executed and the new value that the variable got after the current statement was executed. In this way the effect of each statement on variables can be observed. An example of such visualisation can be observed in Figure 4-2. In the first column we have the variables with their names. In the second column we have the old values of the variables, the values that the variables had before the execution of the current line. In the third column we have the new values of the variables, the values that the variables have after the execution of the current line.

| Var Name | Before | After |
|---|---|---|
| x | {2, 0, 1, 4} | {1, 0, 1, 4} |
| i | 1 | 2 |
| j | 3 | 2 |

Figure 4-2: A possible view of the variables values

The plug-in must support interactivity, the user must be able to go to any step in the execution flow (requirement R6). The animation of the execution should not be continuous but rather giving the possibility to the user to interact with it. Thus the user will have control over the execution by being able to go forward or backward step by step and observe the program states. Moreover the navigation must be easy, and the user must have the possibility to quickly go to a certain state in the execution flow. In Figure 4-1 can be observed an interface that will give such possibilities. In the bottom part of the figure we can see two buttons labelled 'Previous Step' and 'Next Step' that will allow the user to interact with the execution and go step by step forward or backward. Also we can observe a slider, that will allow the user to go faster at a specific step in the execution flow using just a drag action. This is very useful when users want to skip certain parts of the code and go directly to the interested part of the programs execution. Displaying the step number will also help remembering where exactly the interested part is.

In MindXpres the separation between content and visualisation is made very clear. The plug-in must also reflect that (requirement R7). The user should

be focused on content and the visualisation should be automatically generated by the plug-in. Thus the plug-in must have a mechanism to retrieve the necessary data to generate the visualisation. This mechanism will be clearly explained in the next chapter.

Visualising recursion can improve the ability to comprehend recursive subroutines, that is why it is recommended that our plug-in should be able to graphically display recursion (requirement R8). As we saw in our literature study the visualisation must enhance the copies model. This model defines a recursive method as a method capable of creating new instances of itself passing the control forward to new instances and back to the suspended ones. A representation of recursion can be done using recursion trees [42]. In Figure 4-3 we can see such an example. Each node in the tree represents a copy of the recursive method. The root represents the first method that creates new instances (its children) of itself and it passes the control forward and so on until a leaf is reached that means the recursive call has ended and the control is passed back to its parent and so on until the root is back reached. Also for each node the method call is displayed together with its input parameters.

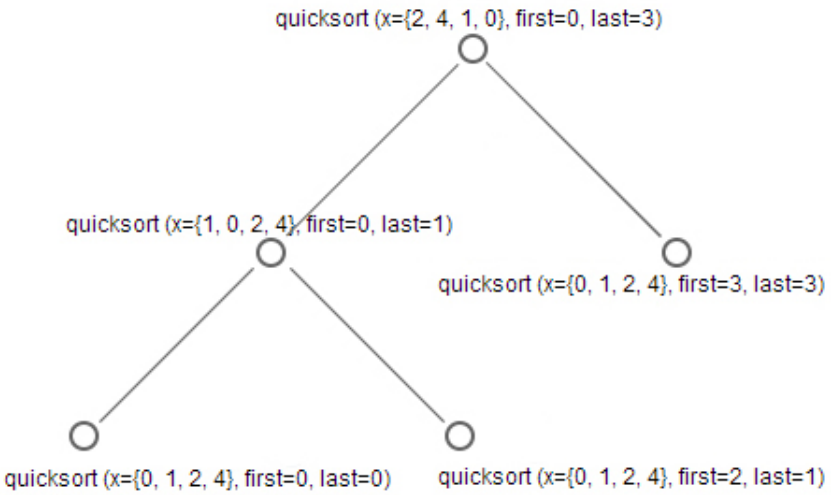


Figure 4-3: A representation of a recursion tree

For more ease of navigation and comprehension of the program the plug-in must allow navigation within the code (requirement R9). This requirement can be achieved by creating links between variable or methods and their def-

inition. The user can easily go to a variable or method definition by clicking on it.

It is also beneficial that the user can observe the output of the program (requirement R10). Even if the user can follow for each statement how the variables are changing and observe for an output operation the value, an explicit display of the programs output will be beneficial.

## 4.2 Conclusion

In the previous sections we defined the requirements and the specifications for our plug-in. All the requirements are based on the literature study conducted in Chapter 2. By building a plug-in that is accomplishing the requirements we will have a MindXpres plug-in, which will help people to better present source code and to visualise program execution, with a very good applicability in education. In the next chapter we go in detail with the actual implementation of the plug-in. We give technical details, documenting how the specifications suggested in this chapter are made into a functional implementation.

# 5 Implementation

In Chapter 4 we defined the requirements and the specifications for our plug-in. Based on that we will implement our plug-in. In this chapter we will present in detail the implementation of the plug-in. First we will describe the architecture, with its components, and describe for each component how it is implemented. Secondly we will describe the graphical user interface of the plug-in. Lastly the integration with MindXpres will be presented.

## 5.1 Architecture

In this section we will describe the architecture of the plug-in. Since our plug-in has multiple functionalities like displaying source code, view variables values, displaying graphically the recursion process and allowing interactivity with the program flow we divide the architecture by components, each component will represent a functionality. Thus we have the following components: The Core, Source Code, Variables, Recursion and Control. We will describe the implementation of each component separately in the following sections.

### 5.1.1 The Core

The most important part of our plug-in is the core. Here we store all the data that we need for our plug-in to work and all our components will use this data.

Let us remember one important requirement for our plug-in, that the visualisation must be automatically generated. The user must be concerned about

the content and let the plug-in create the visualisation. That can mean the user must insert the source code, and the plug-in will do the rest. To do such thing we need an interpreter that could interpret the program. It is not feasible to do that and it will be hard to extend the plug-in for any programming language. Thus we came up with a mechanism to retrieve the necessary data to generate the visualisation. We request to the user to input a file which contains the data or the information from which the data can be extracted. A simple example of such file is a debugger log file. The user runs the program in a debugger and generate a log file. Such file can be easily generated and can contain the data we need. Log files always have a certain structure so they are easy to parse. By creating this mechanism we can extend the plug-in for any programming language using any format for the input file. Before going into detail of this mechanism of data retrieval, let us see the structure of the core. In the following listing we can see the definition of the `CodeVis` class.

```
1  function CodeVis() {
2    this.flow ;
3    this.vars;
4    this.line ;
5    this.data;
6    this.recursivity ;
7    this.recursive_function ;
8    this.recursive_calls ;
9    this.recObj;
10
11   this.init =  function(data) {
12     this.data = data;
13     this.flow = new Array();
14     this.vars = new Array();
15     this.line = new Array();
16
17     this.recursivity = false;
18     this.recursive_function = null;
19     this.recursive_calls = null;
20
21     this.recObj = new Array();
22   }
23 }
```

This class stands at the core of the plug-in and it contains the data needed for the visualisation. The `flow` property is an array which contains the program flow, to be more specific for each element in the array the key represents the step of the flow and the value represents the line number at which the execution is found. The `vars` property is also an array which contains the variables and their values. We will go later into detail explaining the structure of this array. `line` property is also an array which contains the lines of code.

For every value of the array, the key represents the flow step and the value is a string which contains the line of code that is executed at that step. The properties `recursivity`, `recursive_function`, `recursive_calls` and `recObj` are needed to construct the recursion visualisation and will be explained later. The `data` property will be initialised with the content of the input file.

Now that we have an overview of the core structure let us describe the data retrieval mechanism. As we said before we want our plug-in to be extensible for any programming language and by using this mechanism we made it possible. The user must import a file that contains the data or the information that we need to extract the data. We said that such a file can be a debugger log file. For the plug-in to be able to read a certain file, the `CodeVis` class must be extended with a method that parse the file and extracts the data. To exemplify how this is done, for the purpose of this thesis, we extended the core to be able to present source code written in C and Java. For the C language we use as input a GDB: The GNU Project Debugger[1] log file. We chose GDB because it is widely used, has support for Windows and Unix systems, also on many Linux distributions is installed as a default package. For the Java language we use as input Java Debugger (JDB)[2] [3] log files. Both are command line debuggers, they can provide the information we need and are easy to use. To extend our class we simply define the following methods:

```
1  CodeVis.prototype.gdb = function(params) {
2      var recursive_function;
3      var recursive_calls;
4    recursive_function = params[0];
5    recursive_calls = params[1];
6
7        if (typeof recursive_function !== 'undefined' && typeof
             recursive_calls !== 'undefined' ) {
8        this.recursivity = true;
9        this.recursive_function = recursive_function;
10       this.recursive_calls = recursive_calls;
11     }
12       /* here is be the code that parse this.data and initialise
             this.flow, this.vars, this.line and this.recObj */
13 }
14
15 CodeVis.prototype.jdb = function(params) {
16     var recursive_function;
17     var recursive_calls;
```

[1] http://www.sourceware.org/gdb/

[2] http://docs.oracle.com/javase/7/docs/technotes/tools/windows/jdb.html

[3] http://docs.oracle.com/javase/8/docs/technotes/tools/unix/jdb.html

```
18    recursive_function = params[0];
19    recursive_calls = params[1];
20
21       if (typeof recursive_function !== 'undefined' && typeof
             recursive_calls !== 'undefined' ) {
22        this.recursivity = true;
23        this.recursive_function = recursive_function;
24        this.recursive_calls = recursive_calls;
25       }
26        /* here is be the code that parse this.data and initialise
              this.flow, this.vars, this.line and this.recObj */
27  }
```

By calling one of this method we will extract the data from our file and we will have all we need for our visualisation. An example of a simple initialisation of the plug-in is the following:

```
1     codevis = new CodeVis();
2     codevis.init(reader.result);
3     codevis.gdb();
```

In this way the plug-in will have access to `codevis.flow`, `codevis.vars`, `codevis.line`, `codevis.recObj` and it will be able to display the visualisation. Also to be noted that in the latest example we send no parameters to the `codevis.gdb()` constructor, that means that the visualisation will not include the display of a recursion tree.

One can easily extend the plug-in to support a new programming language by extending the `codeVis` class.

### 5.1.2 Source Code

Our next step in the implementation is to display the source code. For a good comprehension and a good presentation the source code must be easy to read, thus the syntax must be formatted and highlighted. There are plenty[1] of Javascript libraries that we can use to accomplish that. For our plug-in we chose Prettify [2]. This is a Javascript library and a CSS file that allows automatic syntax highlighting of source code within an HTML page. As stated on their website the library has the following features:

- Works with code that includes embedded links or line numbers.
- Has a simple API.

[1] http://www.1stwebdesigner.com/css/16-free-javascript-code-syntax-highlighters-for-better-programming/

[2] https://code.google.com/p/google-code-prettify/

- Small size: the loading of the page is not affected.
- Supports CSS styling.
- Automatic language detection. Supports all C, Bash, and XML-like languages.
- Can be extended for other languages.
- Supported by major browsers. Used by code.google.com and stackoverflow.com

### 5.1.3 Variables

Another functionality of our plug-in is the display of variables values, so that one can observe, as the program runs, how the values are changing. All the data about variables is stored in the `vars` array. For each element of the array the key represents the step of the execution flow and the value must be an object which contains the name of the variable, the old value and the new value of the variable at that specific step in the execution flow. The structure of this object can be observed in the following example, which represents a variable named `i` with an old value `0` and a new value `1`.

```
1  Object {name: "i", old_value: "0", new_value: "1"}
```

### 5.1.4 Recursion

We chose for the graphical representation of recursion to use recursion trees. To display such trees we opted for a JavaScript library instead of writing our own graphical generator. We chose to use D3 (Data Driven Documents)[1] library, which is a JavaScript library mostly used in the domain of information visualisation to create and control interactive graphical displays of data. Using D3 we will be able to create a graphical representation of the recursion tree and interact with it.

To be able to build a recursion tree we must have the necessary data in a specific format required by D3. Each node in the tree must have a name, this name will be displayed near the node. In our case the name can be a string representing the method call and its parameters. Also a node can have children, so we need to specify it's children. The children property of the node is an array which contains other nodes. An example of such format is the following:

[1] http://d3js.org/

```
var treeData = [{
    "name": "quicksort(x={2, 4, 1, 0}, first=0, last=3)",
    "children": [
      {
        "name": "quicksort (x={1, 0, 2, 4}, first=0, last=1)",
        "children": [
          {
            "name": "quicksort (x={0, 1, 2, 4}, first=0, last=0)",
            "children": [],
          },
          {
            "name": "quicksort (x={0, 1, 2, 4}, first=2, last=1)",
            "children": [],
          }
        ]
      },
      {
        "name": "quicksort (x={0, 1, 2, 4}, first=3, last=3)",
         "children": [],
      }
    ]
  }];
```

A graphical representation of the structure presented above, using D3 to generate it, can be observed in Figure 5-3. The `recObj` property is an array which contains the recursion tree representation. For each element in the array the key represents the step of the flow and the value represents the tree object corresponding to that flow step. The tree object must have the format previously described.

One important thing in our implementation of GDB and JDB extensions is that the constructor must be initialised with a parameter. Parameter `params` is an array which contains two elements. First element is a string and represents the name of the recursive method, second element is an integer that represents the number of recursive calls within the method. We need this value to know the maximum number of children that a node can have, in order to automatically generate an n-ary tree. In this way the properties `recursive_function`, `recursive_calls` are initialised and we are able to display the recursion tree. When the parameter is sent, `recursivity` receives the value `true` which means the recursion tree will be displayed. A sample initialisation of the visualisation that will display a recursion tree is the following:

```
    //construct  the parser
    parser = new CodeVis();
    parser.init(reader.result);
    //call the specific  the parser
    window["parser"][extension](params);
```

In the last statement of this example the `extension` variable is a string which contains the name of the extension, for example `'gdb'` or `'jdb'`. The value of `extension` is read from the `data-extension` attribute of the container. The value of `params` variable is read from `data-parameters` attribute of the container. Also to be noted that this implementation is specific to our GDB and JDB extensions. With the available input data, from the log files, we needed that additional information in order to successfully create the recursion tree. One can extend the plug-in in another way, by requesting more or even less input parameters.

### 5.1.5 Control

The control of the visualisation is easy to be implemented since we have all the data we need related to the execution flow. If we use our initialisation example, a simple call like:

```
1 codevis.flow[i];
```

will return the line number which is executed at step `i` in the execution flow. A call like:

```
1 codevis.vars[i];
```

will return the variable with its old and new value at the step `i` in the execution flow. A call like:

```
1 codevis.recObj[i];
```

will return an object which contains the structure of the recursion tree at the execution step `i`.
If we need to control our visualisation and make interaction possible a simple manipulation of the execution flow steps is enough, in our example of the parameter `i`.

## 5.2 Graphical User Interface

In the previous section we described the implementation of each functionality of our plug-in. In the following section we will make a presentation of our user interface, and how the visualisation is displayed to the users. We separated the user interface into multiple views, depending on the functionalities that are accomplished by each view.

### 5.2.1 Source Code View

The source code will be displayed in the Source Code View. This view is presented in Figure 5-1. We can see the formatted and highlighted source code. Scrollbars are used when long source code is displayed. Also the

current line that is executed is highlighted. In the figure we can observe line 32 highlighted in yellow. When the user interacts with the visualisation and the current line is changed, the scrolling of this view is done automatically so that the current line is always visible.

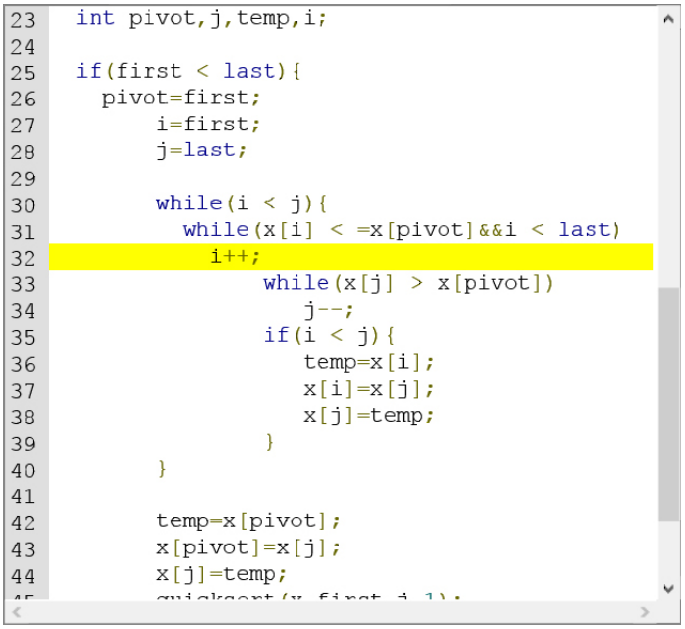


Figure 5-1: Source code view, with current line selected

### 5.2.2 Variables View

In Figure 5-2 we can observe the Variables View. This view is used, as suggests its name, to display the variables value as the program executes. It includes three columns. First column displays the variable name. Second column displays the old value of the variable, the value that it had before the execution of the current line. Third column displays the new value of the variable, the value that it got after the execution of the current line.

| Var Name | Before | After |
|---|---|---|
| x | {2, 4, 4195584, 0} | {2, 4, 1, 0} |
| i | 0 | 1 |
| j | -136403616 | 3 |

Figure 5-2: Variables view

### 5.2.3 Recursion View

As previously stated we will use D3 library to generate the recursion tree. When program entered into a recursive method for which we want to display the recursion tree, the root of the tree is displayed. When a recursive call is met, a new node is added into the tree and so on. This view of the recursion tree can be observed in Figure 5-3.

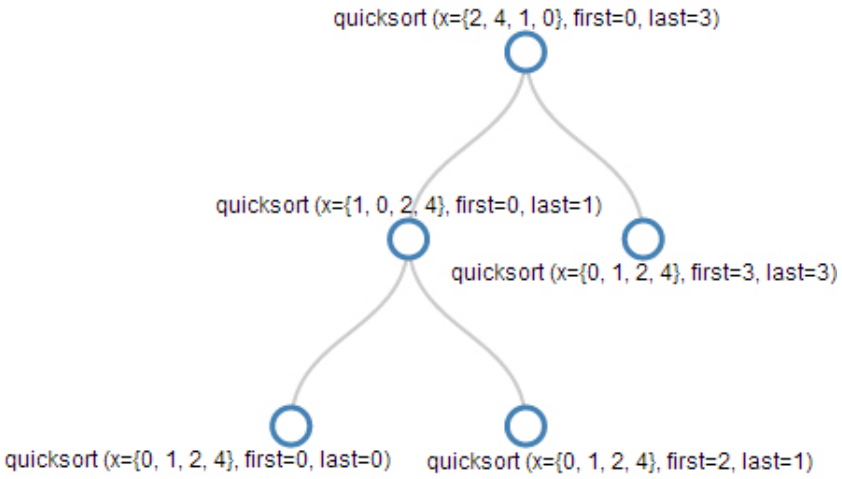


Figure 5-3: A graphical representation of a recursion tree using D3

### 5.2.4 Control View

Control view contains the input elements that allows the user to interact with the visualisation. We can observe in Figure 5-4 that the user can go forward or backward into the execution flow either by using the two buttons "Next Step" and "Previous Step" or the slider. Using the buttons he can go step by step and easily observe the execution. But if the interesting part that he wants to present is at the end of the execution flow it will take a lot of time to go trough every step to arrive at that part. Thus we implemented the slider, which the user can easily drag and go very fast to a specified step in the execution flow and then use the buttons to go step by step.

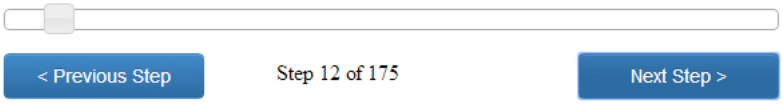


Figure 5-4: Control view

## 5.3 MindXpres Integration

In this section, we describe how the integration of our plug-in into MindXpres is done. We presented the plug-in mechanism of MindXpres in section 3.2. We will use this mechanism to integrate our plug-in into MindXpres.

First we create a new folder in the `components` folder called `codevis`. Here we will create the `plugin_info.js` file, which is the most important file in a plug-in. This file will the first that will be loaded. This file contains the plug-in information, for example the tags that it provides for the XML authoring language. The content of this file is the presented in the following listing:

```
1  var codevis_plugin_info = {
2    types: ["codevis"],
3    path: ""
4  };
```

Next we will create the `codevis.js` file, which contains the plug-in object. The plug-in will be initiated by calling the `init` method and the detected DOM elements that the plug-in handles will be passed by to the plug-in via its `process` method. A shorter version of this file is listed below:

```
1  DI.register_include("codevis_dependencies", codevis_plugin_info.
       path +"dependencies.js");
2
```

```
3  var codevis_plugin = new function(){
4      this.init = function(){ };
5      this.process = function(type, elList){
6       elList.each( function(index, container){
7        //plug-in code goes here
8          });
9     };
10 };
```

The third file that we want to create is `dependencies.js` which was included in the `codevis.js` using the `DI.register_include` call of the dependency injector. This file will include our JavaScript libraries and CSS files. This is also done using the dependency injector. Its content is listed below:

```
1 DI.register_include("d3", codevis_plugin_info.path +"js/d3.v3.min.
    js");
2 DI.register_include("jquery-ui", codevis_plugin_info.path +"js/
    jquery-ui.min.js");
3 DI.register_include("jquery-ui-css", codevis_plugin_info.path +"
    css/jquery-ui.min.css");
4 DI.register_include("style", codevis_plugin_info.path +"css/style.
    css");
5 DI.register_include("parser", codevis_plugin_info.path +"js/parser
    .js");
6 DI.register_include("pretiffy_js", codevis_plugin_info.path +"js/
    prettify.js");
7 DI.register_include("pretiffy_css", codevis_plugin_info.path +"css
    /prettify.css");
8 DI.register_include("script", codevis_plugin_info.path +"js/script
    .js");
```

We separated our files by creating two different folders within the plug-in folder. We created a `js` folder where all our JavaScript libraries are stored and a `css` folder where the CSS files are stored. We included the D3 library needed to display the recursion tree, the jQueryUI[1] library needed for the slider control, and the Pretiffy library needed for the code syntax highlighting. At this point we also include other files which contain the source code of our plug-in.

The XML representation of the plug-in is very simple. We will use just one tag, specifically `<codevis>` tag, that will contain the source code to be presented. This tag has two attributes which are specifying the name of the plug-in extension that is used to parse the input data and the parameters the plug-in receives. The attribute `extension` is mandatory and `parameters`

[1] http://jqueryui.com/

is optional, if it is not used there will be no display of the recursion tree. If we want to send multiple parameters we separate the values by `"|"`. The plug-in will split the string and send parameters as an array. An example of the XML representation is listed below:

```
1 <codevis extension="gdb" parameters="quicksort|2" >
2  source code goes here
3 </codevis>
```

## 5.4 Challenges

One challenge was to make a comprehensive literature study. We started from a general problem and we needed to identify specific problems and find correct answers. We studied the related work and learned from their experiences. We also defined and prioritized a set of requirement that form the basis of our plug-in.

During the implementation we also faced some challenges. One major challenge was how to get the data needed for our visualisation. One solution could be to have an interpreter that would interpret the written program and give us the data. This was not feasible for multiple reasons. Firstly it would get a lot amount of time to build one and exceeds the purpose of this thesis. Secondly to extend the plug-in for multiple languages would imply a new interpreter for each language. The solution we adopted was to use external data files that contain all the information we need. An example of such file is a debugger log file. The main reason we chose this method is because most debuggers offer the possibility to save their output into a file and using a debugger you can watch variables or get the execution flow. Secondly these log files have a well defined structure what makes them easy to read and parse. Another challenge related to previous one was to find the best debuggers for our purpose. They had to be able to provide all the data we need but also to be accessible and easy to use. We wanted to use standard debuggers that are coming with the different development environments so the users do not need to install extra programs. The amount of work to write such parser is highly lower than to write an interpreter, but this method also comes with a shortcoming in usability. To use a debugger and generate a log file one might have to follow certain steps or to write some certain commands, but we foresee in our future work to improve the usability by having some methods to automatically generate such log files.

Another challenge we faced was the creation of the recursion tree. One method to create a tree is to create it form the pre-order traversal array. Be-

sides the pre-order traversal we must also know how many children a node can have and the height of the tree. Having this data, a tree can be easily built, but its branches must always be complete starting from the leftmost one. A recursion tree is not always complete so this method could not be used. We adopted another method which uses an array that contains the method calls and method exits in the order that they happen. Having a method entrance and a method exit as a node delimiter, we know that the other calls between these two represents the node children. To create the recursion tree we also used a recursive algorithm which receives as input the specified array.

We had to develop everything with extensibility in mind so creating a format of the data was also a challenge. We created a core class whose properties retain all the data about the variables, source code and recursion tree, all related to the execution flow.

## 5.5 Conclusion

In this chapter we described the implementation of our plug-in. We described the architecture of the plug-in presenting the implementation of each component. We saw how the core of the plug-in is implemented and how one can extend the plug-in for any programming language. We also presented how the source code, variables changes, recursion and the interaction control are implemented and displayed. At the end of the chapter we described how the integration of the plug-in is made into MindXpres. In the next chapter we will present a scenario of how the plug-in can be used to interactively present source code.

# 6 Use Case

In this chapter we present a use case of the plug-in by means of two scenarios. We show how the plug-in works from a practical standpoint and we document the scenario with code listings and images.

## 6.1 C language

### 6.1.1 Scenario

Now let us define the scenario that we use to present a use case of the plug-in. We create a presentation to present the quicksort algorithm. The program is written in C language and we use GDB to generate the input file from which the plug-in extracts the data for the visualisation. Quicksort is a recursive algorithm, thus we have also recursion visualisation.

### 6.1.2 The presentation

We create a simple presentation containing one slide on which the visualisation of our plug-in is displayed. To define the presentation we must use the XML authoring language. We define the root node `presentation` which contains the `slide` element. Within the `slide` element we insert the `codevis` tag which contains the source code. Next step is to set the attributes of the `codevis` tag. We set the `extension` attribute with the value 'gdb', the attribute `parameters` with the value 'quicksort|2'. The XML representation of the presentation is listed in Listing 6.1 and the output is presented in Figure 6-1.

```
<presentation>
  <slide>
    <codevis extension="gdb" parameters="quicksort|2" >
    <![CDATA[
#include< stdio.h >
void quicksort(int [10],int,int);

int main(){
  int size,i;
  size=4;
  int x[] = {2,4,1,0};

  quicksort(x,0,size-1);

  printf("Sorted elements: ");
  for(i=0;i<size;i++)
    printf(" %d",x[i]);

  return 0;
}

void quicksort(int x[10],int first,int last){
     int pivot,j,temp,i;
     if(first<last){
         pivot=first;
         i=first;
         j=last;
         while(i<j){
             while(x[i]<=x[pivot]&&i<last)
                 i++;
             while(x[j]>x[pivot])
                 j--;
             if(i<j){
                 temp=x[i];
                  x[i]=x[j];
                  x[j]=temp;
             }
         }
         temp=x[pivot];
         x[pivot]=x[j];
         x[j]=temp;
         quicksort(x,first,j-1);
         quicksort(x,j+1,last);
    }
}
    ]]>  </codevis>
  </slide>
</presentation>
```

Listing 6.1: The presentation

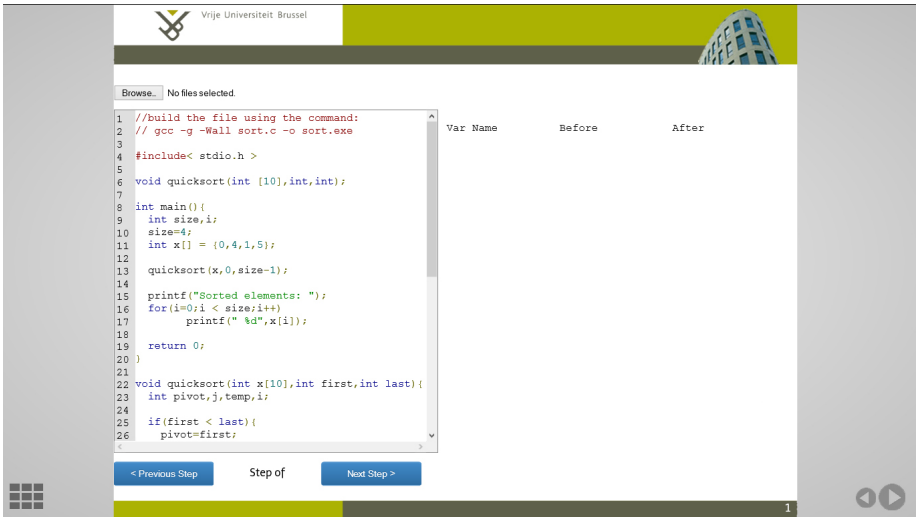


Figure 6-1: Slide view

The next step is to generate the input file using GDB. In order to be able to use the C program with GDB, it must be compiled using the `-g` option. Lets say the source of the program is found in `sort.c` file, the command to compile the program is the following:

```
gcc -g -Wall sort.c -o sort.exe
```

After the program was compiled we can use it with GDB. We wrote a short GDB script to make the generation of the file easier. The GDB script was saved in a file called `sort.gdb` and the content is listed below:

```
# set up breakpoints for sort.exe:
set can-use-hw-watchpoints 0
b main
b quicksort

set logging on
set logging file gdb.txt
set logging redirect off
set logging overwrite on

# go to main breakpoint
run
watch x
```

The script creates a breakpoint on the `main` method and `quicksort` method, and saves the output in the `gdb.txt`. We also want in our visualisation to view how variable `x` is changing, so we also set a watchpoint on `x`. We execute the GDB script using the command bellow:

```
gdb -x sort.gdb -se sort.exe
```

This command starts the GDB tool and stops the execution of the program at the first breakpoint. By entering the command `next` it will execute further. At the next breakpoint stops again, this is when it enters the `quicksort` method. Here we can also set to watch local variables, so we give the following commands to watch `i` and `j`:

```
watch i
watch j
next
```

After the execution is finished the output was saved to the `gdb.txt` file that we can import it to our presentation. Immediately after the import was done, the visualisation is ready. We can see the result in Figure 6-2. The slider is visible, the current line is highlighted and changes of variable `x` are displayed.

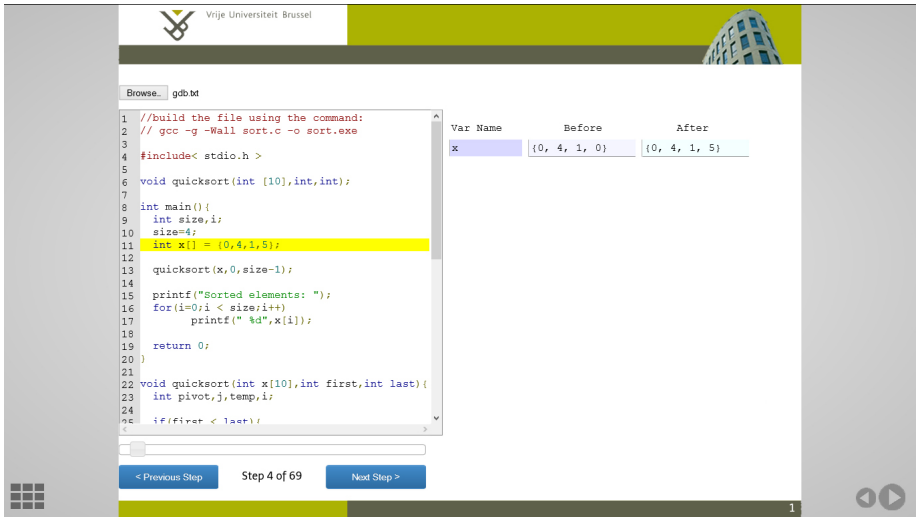


Figure 6-2: Visualisation is ready

In Figure 6-3 we can also observe the creation of the recursion tree as we step forward into the programs execution.

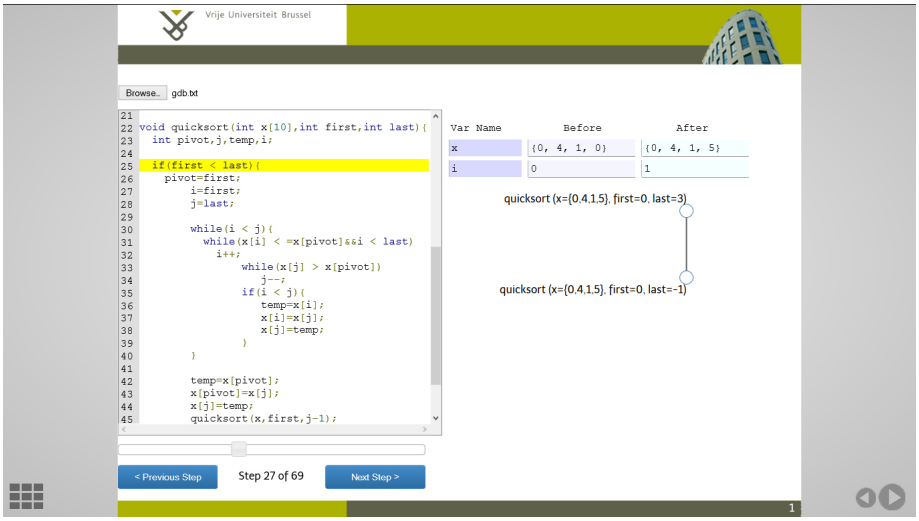


Figure 6-3: Visualisation is ready

## 6.2 Java language

### 6.2.1 Scenario

In this second scenario we create a presentation to present the quicksort algorithm written in Java language. We use JDB to generate the input file from which the plug-in extracts the data for the visualisation. Our visualisation also includes the recursion tree of the algorithm.

### 6.2.2 The presentation

The process of creating the presentation is similar to the one presented in the previous section. The XML representation of the presentation is listed in Listing 6.2. We can observe that the `extension` attribute of the `codevis` element now has the value 'jdb'. That is because we use the `jdb` extension of the plugin.

```
<presentation>
  <slide>
    <codevis extension="jdb" parameters="quicksort|2">
    <![CDATA[
class Quicksort  {
  private static int[] numbers;
  private static int number;
  private static int i,j;

  public static void main(String []args) {
    int c;
    numbers = new int[]{1,56,3,45};
    number = numbers.length;
```

```
14      quicksort(0, number - 1);
15
16      System.out.println("Sorted list of numbers");
17
18      for (c = 0; c < numbers.length; c++)
19        System.out.println(numbers[c]);
20    }
21
22    private static void quicksort(int low, int high) {
23      i = low;
24      j = high;
25      int pivot = numbers[low + (high-low)/2];
26
27      while (i <= j) {
28        while (numbers[i] < pivot) {
29          i++;
30        }
31        while (numbers[j] > pivot) {
32          j--;
33        }
34
35        if (i <= j) {
36          int temp = numbers[i];
37          numbers[i] = numbers[j];
38          numbers[j] = temp;
39          i++;
40          j--;
41        }
42      }
43
44      if (low < j)
45        quicksort(low, j);
46      if (i < high)
47        quicksort(i, high);
48    }
49
50
51  }
52      ]]>  </codevis>
53    </slide>
54  </presentation>
```

Listing 6.2: The presentation

In the case of a Java program we use JDB to generate the log file needed for the plug-in. To be able to use a program with JDB it must be compiled using the `-g` option. An example is given in the following listing:

```
1  javac -g Quicksort.java
```

Next we write a simple JDB script that generates the data we need. We save this file as `quicksort.jdb` and its content is presented in the following listing:

```
stop in Quicksort.main
stop in Quicksort.quicksort
watch Quicksort.numbers
watch Quicksort.i
watch Quicksort.j
run
trace methods
monitor step
step
```

The `stop` command creates breakpoints. We use `watch` command to get the variables values. The `trace` command will output a specific message, when execution enters a method, that is used to generate the recursion tree. To execute the script we use the following commands:

```
jdb Quicksort
read quicksort.jdb
```

Before running the `jdb` we have to start writing the output of the terminal in a file and we use the `script` command to do that. The following command writes the output in the `log.txt` file:

```
script log.txt
```

To close the output stream and save the file we use the command:

```
script exit
```

Having this file generated we can use it to generate our visualisation.

## 6.3 Conclusion

In this chapter we presented the use of the plug-in by use of two scenarios. We created two presentations of the quicksort algorithm. One presentation for the algorithm written in C that uses a GDB log file as input to generate the visualisation and a second one for the algorithm written in Java that uses JDB log file as input. We showed how a presentation can be built and also the ease of generating the log file.

# 7 Conclusions

In this chapter we want to recapitulate our work and contributions this thesis is bringing. Also we will discuss some possible future work for the plug-in.

While presentations are becoming more and more important nowadays, most presentation tools did not evolved much since their apparition. To present special content can become an impossible task. MindXpres is a presentation tool that brings a shift of paradigms and changes the way people create, share and deliver presentation. Having an extensible architecture, we extended MindXpres to present special content such as source code.

The purpose of this thesis was to extend MindXpres with an intelligent plug-in that will give the presenter the ability to present source code. Moreover the presenter will be able to create a visualisation about the program execution.

We started our work with a literature study to identify the most important difficulties in presenting source code, more specifically during the process of learning and teaching programming. Through this study we gained a good insight about the matter in hand, and we found that there are real difficulties for novice programmers to comprehend some specific concepts. The comprehension of programs can be influenced by the choice of language, for a procedural programming language a strong program model and for an object-oriented programming language a strong domain model are facilitating the comprehension of the programs. We also presented studies that are showing the concepts which are found difficult by novices. The conclusions of this studies were that these topics are difficult just because the novices do

not have the ability to understand what happens to program while is executing. This is because they were incapable of creating a mental model of the programs execution. From this study we concluded that an efficient mental model is one that uses visuals and shows the user the changes in the source code, such as variable changes, as the program runs. as the program runs. We also found that another concept found difficult by novices is recursion. Novices are usually seeing recursion as an iterative process, which is wrong. Recursion is better understood when visualisations are used.

There is active research going on in the domain and we have presented some of the related work. These tools are used to help students create effective mental models and we took a look at their approach and implementation but also at their efficiency.

Having as basis our literature study we defined the requirements of our plug-in and presented its implementation. We created a MindXpres plug-in that is able to simulate the execution flow of a program, it shows variables changes and can generate a visualisation of recursive methods. Besides that, the visualisation of the plug-in allows interactivity, the author is able to go backward and forward through the execution flow. The plug-in accomplishes the conditions of an effective mental model.

We succeeded to bring a good contribution with this thesis. We made an analysis of the existing tools used to teach programming and identified their main features and characteristics. Based on the study we made we defined some requirements that such a tool must have. The current slideware are very limited but with the power of MindXpres we succeeded to create a tool to be used during presentations. This tool help novices to gain an effective mental model of the programs execution by using an interactive visualisation of the execution flow and variable changes. Moreover this tool is not limited to one programming language but can be extended to support any programming language. This makes it different from any other tool, which are specifically built for one programming language. We also made it easy to extend the tool by using input files which contain the data needed for visualisation. Also we succeeded to integrate the recursion visualisation and successfully create a copies model of the recursion execution. Most of the other tools which are dealing with recursion visualisation are specifically built for this purpose and do not offer many other features.

For the future work the real effectiveness of the plug-in can be tested and improved through an evaluation. Such an evaluation can be done through a questionnaire for both students and teachers. Through this evaluation we can test if the students are better understanding some core concepts that

are hard to grasp and for teachers test the usability to know how that can be improved. In the future work the plug-in can be improved by adding new visualisations. Understanding pointers can be sometimes difficult and adding a visualisation for pointers would improve our plug-in. Because with our plug-in we can observe the variable values at any time we did not create a visualisation for the program output but we propose this to be done in the future work. In this way the program output would be better observed. The recursion visualisation can be also improved by adding the visualisation of the stack. Also in the future work we propose an increase of usability in generating the log files. In our implementations we use GDB and JDB which are command line debuggers and the user must write some specific commands to create the log files. This process can be automated by creating a user interface that calls specific debugger commands in the background. In this way the usability will increase.

# Bibliography


[1] Slideshare. http://www.slideshare.net/about. Accessed: 2014-02-10.

[2] J. R. Anderson, P. Pirolli, and R. Farrell. Learning to Program Recursive Functions. *The nature of expertise*, pages 153–184, 1988.

[3] J. Bennedsen and M. E. Caspersen. Failure Rates in Introductory Programming. *ACM SIGCSE Bulletin*, 39(2):32–36, 2007.

[4] M. Berry and M. Kölling. The Design and Implementation of a Notional Machine for Teaching Introductory Programming. 2013.

[5] B. D. Boulay. Some Difficulties of Learning to Program. *Journal of Educational Computing Research*, 2(1):57–73, 1986.

[6] J. J. Cañas, M. T. Bajo, and P. Gonzalvo. Mental Models and Computer Programming. *International Journal of Human-Computer Studies*, 40(5):795–811, 1994.

[7] W. Dann, S. Cooper, and R. Pausch. Using Visualization to Teach Novices Recursion. In *ACM SIGCSE Bulletin*, volume 33, pages 109–112. ACM, 2001.

[8] J. Edgington. Teaching and Viewing Recursion as Delegation. *Journal of Computing Sciences in Colleges*, 23(1):241–246, 2007.

[9] B. S. Elenbogen and M. R. O'Kennon. Teaching Recursion Using Fractals in Prolog. *ACM SIGCSE Bulletin*, 20(1):263–266, 1988.

[10] G. Ford. An Implementation-Independent Approach to Teaching Recursion. In *ACM SIGCSE Bulletin*, volume 16, pages 213–216. ACM, 1984.

[11] C. E. George. Experiences with Novices: The Importance of Graphical Representations in Supporting Mental Models. In *12th Annual Workshop of the Psychology of Programming Interest Group*, pages 33–44, 2000.

[12] N. Gershon and W. Page. What Storytelling can do for Information Visualization. *Communications of the ACM*, (8):31–37, August 2001.

[13] D. Ginat. Do Senior CS Students Capitalize on Recursion? *ACM SIGCSE Bulletin*, 36(3):82–86, 2004.

[14] D. Ginat and E. Shifroni. Teaching Recursion in a Procedural Environment — How Much Should We Emphasize the Computing Model? In *ACM SIGCSE Bulletin*, volume 31, pages 127–131. ACM, 1999.

[15] M. Goldwasser and D. Letscher. Teaching Strategies for Reinforcing Structural Recursion With Lists. In *Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*, pages 889–896. ACM, 2007.

[16] A. Gomes and A. J. Mendes. Learning to Program - Difficulties and Solutions. In *International Conference on Engineering Education–ICEE*, volume 2007, 2007.

[17] T. Jenkins. Teaching Programming - A Journey From Teacher to Motivator. In *The 2nd Annual Conference of the LSTN Center for Information and Computer Science*, 2001.

[18] T. Jenkins. The Motivation of Students of Programming. In *ACM SIGCSE Bulletin*, volume 33, pages 53–56. ACM, 2001.

[19] T. Jenkins. On the Difficulty of Learning to Program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, volume 4, pages 53–58, 2002.

[20] H. Kahney. What do Novice Programmers Know About Recursion. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 235–239. ACM, 1983.

[21] C. Kelleher and R. Pausch. Using Storytelling to Motivate Programming. *Communications of the ACM*, 50(7):58–64, 2007.

[22] A. Kelly. Requirements and Specifications.

[23] S. Kennewell. Researching the Influence of Interactive Presentation Tools on Teachers' Pedagogy. February 2006.

[24] M. Kölling, B. Quig, A. Patterson, and J. Rosenberg. The BlueJ System and its Pedagogy. *Computer Science Education*, 13(4):249–268, 2003.

[25] E. Kujansuu and T. Tapio. Codewitz–an international project for better programming skills. In *World Conference on Educational Multimedia, Hypermedia and Telecommunications*, volume 2004, pages 2237–2239, 2004.

[26] E. Lahtinen, K. Ala-Mutka, and H.-M. Järvinen. A Study of the Difficulties of Novice Programmers. In *ACM SIGCSE Bulletin*, volume 37, pages 14–18. ACM, 2005.

[27] R. Lowe. Interrogation of a Dynamic Visualization During Learning. *Learning and Instruction*, 14(3):257–274, 2004.

[28] L. Ma, J. Ferguson, M. Roper, and M. Wood. Improving the Viability of Mental Models Held by Novice Programmers. In *Eleventh Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts. ECOOP Workshops 2007*, 2007.

[29] R. E. Mayer. The Psychology of How Novices Learn Computer Programming. *ACM Computing Surveys (CSUR)*, 13(1):121–141, 1981.

[30] I. Milne and G. Rowe. Difficulties in Learning and Teaching Programming—Views of Students and Tutors. *Education and Information Technologies*, 7(1):55–66, March 2002.

[31] A. Moreno and M. S. Joy. Jeliot 3 in a Demanding Educational Setting. *Electronic Notes in Theoretical Computer Science*, 178:51–59, 2007.

[32] A. Moreno, N. Myller, E. Sutinen, and M. Ben-Ari. Visualizing Programs With Jeliot 3. In *Proceedings of the working conference on Advanced visual interfaces*, pages 373–376. ACM, 2004.

[33] I. Parker. Absolute PowerPoint: Can a Software Package Edit Our Thoughts? *The New Yorker*, pages 76–87, May 2001.

[34] T. Rajan. Principles for the Design of Dynamic Tracing Environments for Novice Programmers. *Instructional Science*, 19(4-5):377–406, 1990.

[35] V. Ramalingam and S. Wiedenbeck. An Empirical Study of Novice Program Comprehension in the Imperative and Object-Oriented Styles. In *Papers presented at the seventh workshop on Empirical studies of programmers*, pages 124–139. ACM, 1997.

[36] R. Roels and B. Signer. MindXpres - An Extensible Content-driven Cross-Media Presentation Tool. In *In Proceedings of the 27th BCS Conference on Human Computer Interaction (HCI 2013)*, London,United Kingdom, 2013.

[37] L. Sa and W.-J. Hsin. Traceable Recursion with Graphical Illustration for Novice Programmers. *InSight: A Journal of Scholarly Teaching*, 5, 2010.

[38] P. A. Smith and G. I. Webb. Reinforcing a Generic Computer Model for Novice Programmers. *ASCILITE'95*, 1995.

[39] P. A. Smith and G. I. Webb. The Efficacy of a Low-level Program Visualization Tool for Teaching Programming Concepts to Novice C Programmers. *Journal of Educational Computing Research*, 22(2):187–216, 2000.

[40] E. R. Tufte. *The Cognitive Style of PowerPoint: Pitching Out Corrupts Within, Second Edition*. Graphics Press, 2006.

[41] B. Tversky, J. B. Morrison, and M. Betrancourt. Animation: Can it Facilitate? *International Journal of Human-Computer Studies*, 57(4):247–262, October 2002.

[42] J. Á. Velázquez-Iturbide and A. Pérez-Carrasco. InfoVis Interaction Techniques in Animation of Recursive Programs. *Algorithms*, 3(1):76–91, 2010.

[43] A. T. Virtanen, E. Lahtinen, and H.-M. Järvinen. Vip, a visual interpreter for learning introductory programming with c++. In *Proceedings of The Fifth Koli Calling Conference on Computer Science Education*, pages 125–130, 2005.

[44] S. Wiedenbeck. Learning Iteration and Recursion From Examples. *International journal of man-machine studies*, 30(1):1–22, 1989.

[45] S. Wiedenbeck, V. Ramalingam, S. Sarasamma, and C. Corritore. A Comparison of the Comprehension of Object-Oriented and Procedural Programs by Novice Programmers. *Interacting with Computers*, 11(3):255–282, 1999.