Chalmers University of Technology
University of Gothenburg

Data Structure and Algorithm Design Visualization Tool

DATX02-16-23

Planning Report

Johan Gerdin | Ivar Josefsson | Dennis Jönsson
Jonathan Skårstedt | Simon Smith | Richard Sundqvist

February 25, 2016

Contents

1 Background **1**

2 Purpose **1**

3 Problem **2**

3.1 Implementation & Translation 2

3.2 Interpreter & Grammar 3

3.3 Log File 3

3.4 Visualisation 3

4 Delimitations **4**

5 Method **4**

5.1 Research Stage 4

5.2 Development Stage 5

6 Schedule **5**

References **6**

1 Background

The usefulness of animations of data structures and algorithmic operations on them is not agreed upon. One of the principal weaknesses appears to be that students do not remember the material when it is simply played out before them, without interaction [1].

There exists a large number of AV-systems, freely available on the internet today. A common characteristic for most of these systems is that they are of low quality in terms of usability and overall utility. The majority of Systems have been produced by small-scale research efforts and have been short lived in terms of development [2]. AV-systems are usually older Java applets dated back to the mid 1990's [3]. Newer AV-systems has emerged since then, however, few of them have had any success and most are rather specialized which makes broader applications difficult [2].

After the advent of HTML5, newer and more powerful AV systems such as VisuAlgo and PyViz have been developed [4]. However, none of the systems which are in use today can be regarded as mainstream in CS education [5]. Some believe that the field is lacking systems which are offering required visualizations in packages with a unified interface [6, 5].

We believe there is a need for a visualization tool which supports easy-to-use integration with several languages that assists the programmer in their work by visualising the effects of written code in development.

2 Purpose

We want to create a modular AV system capable of being integrated with any general-purpose programming language. Our purpose is not to develop for all existing languages, but rather develop a well-defined grammar that the interpreter for our visualization module can understand. As a proof of concept, we wish to implement a solution for a single programming language which alters source code to output a stream adhering to the grammar. Third party developers should then be able to make their own translators for whichever host language they prefer, and extend the AV system with custom visualizations.

Our hope is that beginners, students and professionals will find this system useful when implementing algorithms in a familiar host language. Our primary target group is beginners and students who may find it difficult to debug abstract structures such as trees by simply looking at the tables of variables in values which debuggers typically provide. Our system will instead visualize data structures similarly to how they are drawn in course literature. By recording and drawing each operation we believe the user will more easily be able to detect how the data structure is affected by the program and especially where logical errors occurs in the algorithm.

Because our system should be usable by those with little experience in software development, considerable effort must be put into making both the translator and the AV module user friendly.

3 Problem

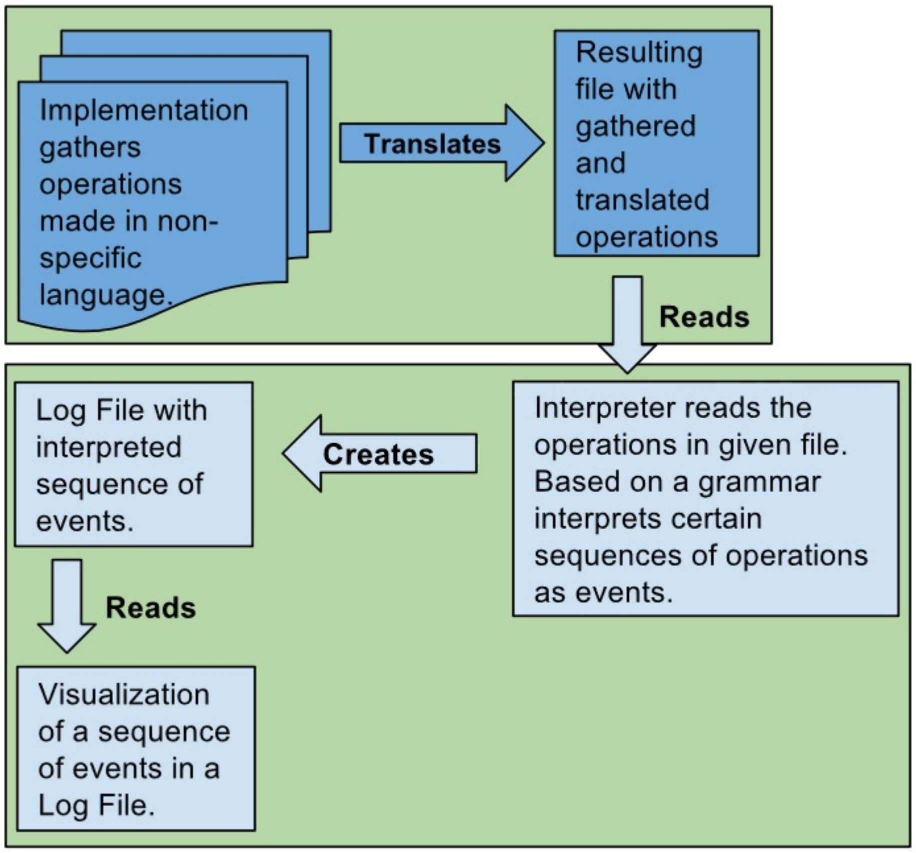

Implementation gathers operations made in non-specific language.
Translates
Resulting file with gathered and translated operations
Reads
Log File with interpreted sequence of events.
Creates
Interpreter reads the operations in given file. Based on a grammar interprets certain sequences of operations as events.
Reads
Visualization of a sequence of events in a Log File.

Figure 1: An overview of the system.

Fig 1. gives an overview of the entire system. The system can be broken down into these separate parts:

- Implementation & Translation
- Interpreter & Grammar
- Log File
- Visualization

3.1 Implementation & Translation

An implementation in a non-specific language that determines what sequence of basic operations are being performed on a given data structure. The implementation will translate this into a file that contains this sequence on a pre-specified format. Some issues are:

- How do we decouple the specific language from the resulting file?
- How do we make sure the resulting file is created on the same format in any language?
- How do we determine how and where the operations are used in the source code?

3.2 Interpreter & Grammar

An interpreter that will read the file created by the implementation. The system will interpret the sequence based on a given set of rules (a grammar) and match these operations against events defined in the grammar of higher abstraction than for example "read/write" such as "swap element 1 and 2". Some issues are:

- How smart should the interpreter be, i.e. what level of abstraction are we looking for?
- How should the grammar handle ambiguities, i.e. a sequence that can be interpreted in multiple ways?
- How do we make the grammar flexible and extendable?

3.3 Log File

A resulting log file with the interpreted sequence of operations.

- In what format should this file be stored?
- How should the file represent the sequence?

3.4 Visualisation

A visualization of a sequence given by a log file. The visualization should be varied in colour and shape and be animated. One should be able to replay the visualization and be able to go through the steps in the sequence one by one.

- How do we set the contract for how the visualization should handle the sequence in the log file?
- How do we represent the various operations in the sequence?

4 Delimitations

The visualization will be done primarily in two dimensions. However, we intend to use an environment which allows for drawing of three dimensions in the future.

Communication between the source program and our solution is strictly unidirectional, with our solution as the receiver. You will not, for example, be able to halt execution of the algorithm from the visual interface (though the visualisation itself may be played at any pace in any direction).

5 Method

We aim to incrementally build our system in iterations using agile methods. The development process will be based on SCRUM and XP (eXtreme Programing) where we will make use of main concepts described by Leffingwell such as product features, project backlog, user stories, tasks, spikes and sprints [7]. The project will be divided into three major stages, the initial research stage, development stage and lastly a third stage for finishing the documentation.

Our main channels for communications are the web and two to three weekly meetings. During the weekly meetings we will discuss our progress and handle Scrum related activities, planning and workload distribution. Extra workshops will be set up to facilitate sharing of knowledge, collaborative work and handling of development related issues.

5.1 Research Stage

Research and initial requirements analysis will make up the majority of the first stage of the project. The goal of this stage is to identify user needs and gain an overall understanding of the current state of the field. The results of this research will serve as a base for our initial vision and main high level features of the system.

User stories are the elementary units of functionality and main building blocks of this project. User stories are derived from high level product features and are brief and concrete statements that describes something the system must be able to accomplish. They have to be defined in a way that is understandable for both the developer and a future user of the system. User stories are usually on the form: “As a <Role>, I can <Activity> so that <Result>.”

Since the stories are derived from product features, they may initially be too high level and will have to be broken down into smaller pieces. This is done repeatedly until a story reaches a proper level of detail. A proper level is where a user story can be regarded as independent, time estimable and testable. After a story of proper size has been defined, it will finally be broken down into separate tasks of coding, defining tests, testing, and finally implementing the described functionality.

User stories are stored in a document called the project backlog. In the backlog each story will get a status on whether it has been started and what tasks have been performed. Some user stories may necessitate research of technical approaches or some level of prototyping before work may begin. For these types of concerns, technical and functional tasks called spikes are defined. Spikes are stored in the backlog along with the user stories as dependencies. If there are spikes present, they must be completed before work on relevant tasks begin.

5.2 Development Stage

After an initial backlog has been set up, we'll divide user stories into separate sets according to their dependencies and priorities. Each set will represent a separate version of the system and will be implemented and tested in sprints. Sprints are time periods of 1 - 2 weeks where we incrementally extend the previous version of the system. During a sprint user stories and tasks will be divided over the team and tasks will be completed in order for each user story. At the end of a sprint we'll evaluate our progress and refactor the requirements and project backlog based on the result of the evaluation. All sprints may however not contain tasks of implementing new parts of the system. This is because a spike may be concerning core functionalities or other things which may have an impact on the success of the project. In this case it can be wise to take enough time investigating different paths.

The sprints will be repeated up until the point where the system is finished or the time for the project is up. Since every incremental version of the system is concerned with its own tasks of defining, coding, testing, implementing, and refactoring, no other stages will be required. The overall idea is to always have some version of the system ready for demonstration.

6 Schedule

Since we will follow an agile development cycle we are trying to avoid hard deadlines, however there is a need to make sure that the project continues as planned and making sure we end the project at an adequate time, allowing us to produce required and recommended documentation and prepare adequately for our presentations. We estimate that we have 340 hours per person to dedicate to the project given that each person should spend about 400 hours total on the project. With this in mind we estimate to have 14 development cycles, where each cycle lasts for a week.

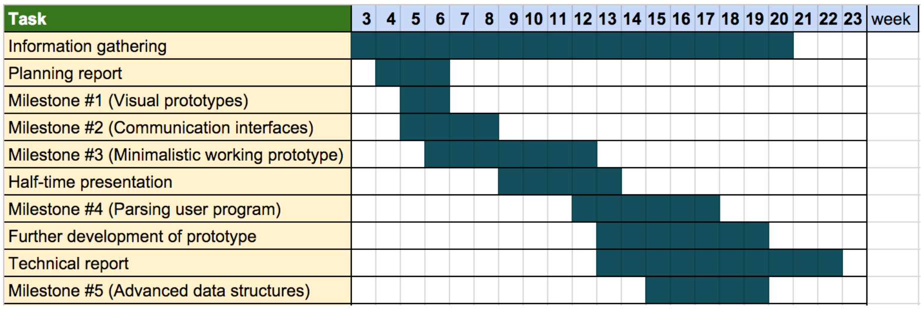
Given the definition of the system and the established vision we defined several milestones:

1. Prototype and test different visualization tools/libraries
2. Define contract between modules and definitions of communication between modules

3. Basic functionality and structure of software project, excluding modification of user written program, starting directly from a created OI file. Every module of the system should be included in this step excluding the parser for the user written program. At this point a basic demo should be able to be run given our produced system. This milestone does not include support for more advanced data structures or instructions.

4. Going from a user written program to producing a readable OI file

5. From user defined program to visualisation, advanced data structures and support for user written modules to be used by the program.



Task	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	week
Information gathering	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■				
Planning report		■	■	■																		
Milestone #1 (Visual prototypes)			■	■																		
Milestone #2 (Communication interfaces)			■	■	■	■																
Milestone #3 (Minimalistic working prototype)				■	■	■	■	■	■	■												
Half-time presentation							■	■	■	■	■											
Milestone #4 (Parsing user program)										■	■	■	■	■	■							
Further development of prototype											■	■	■	■	■	■	■					
Technical report											■	■	■	■	■	■	■	■	■	■		
Milestone #5 (Advanced data structures)													■	■	■	■	■					

Figure 2: A Gantt chart detailing the expected active tasks per week.

We will begin with prototyping and testing different tools/libraries that we might be interested in using. This part of the project is estimated to be done for two weeks. Not only should we become more familiar with the tools that we might be using but we want to try to encounter as many problems as possible at an early stage, to avoid painting ourselves into a corner later.

After we are done prototyping we will focus on developing and producing standards for how the communication between modules will work and look. These definitions will then be continually developed and expanded upon as the project progresses but the brunt of the work should be done during the beginning of the development cycle of the project.

References

[1] C. Hundhausen, S. Douglas, and J. Stasko. (2002) A meta-study of algorithm visualization effectiveness. [Online]. Available: http://www.cc.gatech.edu/ stasko/papers/jvlc02.pdf

[2] M. L. Cooper, C. A. Shaffer, S. H. Edwards, and S. P. Ponce, "Open source software and the algorithm visualization community," *Science of Computer Programming*, vol. 88, pp. 82–91, 2014.

[3] C. A. Shaffer, M. L. Cooper, A. J. D. Alon, M. Akbar, M. Stewart, S. Ponce, and S. H. Edwards, “Algorithm visualization: The state of the field,” *ACM Transactions on Computing Education (TOCE)*, vol. 10, no. 3, p. 9, 2010. [Online]. Available: http://www.cs.utep.edu/makbar/papers/TOCE.pdf

[4] S. Šimoňák, “Using algorithm visualizations in computer science education,” *Open Computer Science*, vol. 4, no. 3, pp. 183–190, 2014.

[5] J. Sorva, V. Karavirta, and L. Malmi, “A review of generic program visualization systems for introductory programming education,” *ACM Transactions on Computing Education (TOCE)*, vol. 13, no. 4, p. 15, 2013. [Online]. Available: http://demo.villekaravirta.com/PVreview.pdf

[6] V. Karavirta and C. Shaffer, “Creating engaging online learning material with the jsav javascript algorithm visualization library.” [Online]. Available: https://people.cs.vt.edu/ shaffer/Papers/p159-karavirta.pdf

[7] D. Leffingwell, *Agile software requirements: lean requirements practices for teams, programs, and the enterprise.* Addison-Wesley Professional, 2010.