9 Software Visualization

Real programmers code in binary.
9.1 Definitions

- „... thought is impossible without an image.“
  [Aristotle, 350 BC]

- „Imagination or visualization, and in particular the use of diagrams, has a crucial part to play in scientific investigation.“
  [Rene Descartes, 1637]

- „The understanding can intuit nothing, the senses can think nothing. Only through their union can knowledge arise.“
  [Emanuel Kant, 1781]
9 Software Visualization

• 9.1 Definitions
  – „Logicians may reason about abstractions. But the great mass of men must have images.“
    [Thomas Babington Macaulay, 1825]
  – Nowadays:
    • Computers are an important tool to create beautiful and efficient visualizations.
    • User has a better understanding of complex phenomena.
    • Consequently, visualization has become a subdiscipline of computer science.
9 Software Visualization

• 9.1 Definitions

  – “Visualization is the use of computers or techniques for comprehending data or to extract knowledge from the results of simulations, computations, or measurements.”

    [McCormick, DeFanti, Brown, 1987]

  – Definition above:
    • Visualization not restricted to rendering information visible
    • In general, visualization is set equal to perceptualization including
      – Sonification (Sound, Traffic light example)
      – Tactilization and Haptilization (Touch, Braille)
9 Software Visualization

• 9.1 Definitions
  – Visualization plays a major role in the use of computers to support human reasoning.
    ➔ Intelligence Amplification (IA)
  – In contrast: Artificial Intelligence (AI)
    • Building a human-like intelligence form
    • Making the computer itself intelligent
9 Software Visualization

9.1 Definitions

- Visualization frequently used in
  - Mechanical engineering
  - Chemistry
  - Physics
  - Medicine
  - Biology

- Only little use of visualization as a tool for
  - Designing Software
  - Implementing Software
  - Maintaining Software
9 Software Visualization

• 9.1 Definitions
  
  – Many software developers consider themselves as theoreticians and disregard visualization completely!

Programmers tend to adapt to the level of representation provided by the computer, instead of adapting the computer representations to their perceptual abilities!
9.1 Definitions

- Many formal and cryptical notations, but terminology of computer science is rich of visual metaphors
- Goal of the metaphors is to generate mental images, to better memorize ideas and concepts, and to exploit the analogies to understand structures and functions more easily
- Automata and machines are mathematical models of computation
- Tapes, trees, leaves, queues, files, folders, archives form mental images of the models
9 Software Visualization

9.1 Definitions

- For example: A Turing machine is a mathematical model consisting of sets, functions, and relations.
- Machine analogy helps to transport aspects from the physical world to mathematical and consequently, helps to better understand the mathematical model.
- Analogy: Gear wheels and how one drives the other once we start to turn one among them!
9 Software Visualization

• 9.1 Definitions
  – Goal of Software Visualization:
    • Not the generation of nice and aesthetically pleasing images and diagrams
    • BUT: The generation of images and diagrams with the goal to support software comprehension
    • Invention of novel visual metaphors will not only generate more effective and efficient visualizations, it definitely will improve the way software developers explore software systems.
9 Software Visualization

9.1 Definitions

- Narrow Definition
  - Visualization of algorithms and programs

- Wide Definition
  - Visualization of artifacts related to software and its development process
9 Software Visualization

9.1 Definitions

- Researchers in Software Visualization develop methods and apply visual metaphors on different aspects of software
  - Static structure of the software
    - Hierarchy, Call relations, Source code,...
  - Concrete and abstract execution of the software
    - Execution traces, Data during runtime,...
  - Evolution of software
    - How is the system built over time?
9 Software Visualization

9.1 Definitions

- Two disciplines of visualization
  - Scientific Visualization
    - Data with inherent 3D spatial structure
  - Information Visualization
    - Abstract data

- Software Visualization is a subdomain of Information Visualization because this kind of data has an abstract nature.

  (graphs, hierarchies, text, quantitative, ordered, categorical data)
• 9.2 Taxonomies and Surveys
  – Classification of Software Visualization research and tools
  – Taxonomy by Myers for program visualization
  – Distinguishing between data, code, and algorithm
  – Algorithm visualization on a higher level of abstraction than program code

![Diagram showing the classification of data, code, and algorithm into static and dynamic categories.](image)
9.2 Taxonomies and Surveys

– Price et al. suggested a more hierarchical taxonomy of software visualization.

– Distinguishing between program visualization consisting of code and data visualization and more abstract algorithm animation.
9.2 Taxonomies and Surveys

- Price et al. introduced in 1993 several aspects which may be used to classify software visualization tools
  - Scope: Range of programs used as inputs for the visualization
  - Content: What information about the software is visualized
  - Form: Characteristics of output of system (e.g. medium)
  - Method: How is the visualization specified?
  - Interaction: How does the user control the system?
  - Effectiveness: How well does the system communicate information to the user?
9.2 Taxonomies and Surveys

- Roman and Cox used in 1993 a very similar set of aspects to classify existing software visualization tools
  - Scope (code, data state, control state, behavior)
  - Abstraction
  - Specification method
  - Interface and presentation
9.2 Taxonomies and Surveys

- Oudshoorn et al. in 1996 introduced a quite different and more technical program visualization taxonomy
- Based on what kind of software on what kind of hardware is actually visualized
9.2 Taxonomies and Surveys

- By attempting to identify open research questions in software visualization, Diehl proposed a literature survey in 2002 and classified research papers in a 4 x 4 matrix.
- Matrix rows: hardware, virtual/abstract machine, program, and system.
- Matrix columns: static and dynamic phenomena.
9 Software Visualization

9.2 Taxonomies and Surveys

- Matrix incomplete because one could add additional layers e.g. operating system or project structure
- Shades of gray indicate how much published research exists in certain areas of software visualization
- Matrix gives a rough orientation about the activity of research in these areas
9.2 Taxonomies and Surveys

- Koschke in 2002 presented a survey based on questionnaires filled in by 111 researchers from
  - Software maintainance
  - Re-engineering
  - Reverse engineering

- Koschke reports that
  - 40% find software visualization absolutely necessary for their work
  - 42% find it important but not critical
• 9.2 Taxonomies and Surveys

  – Bassil and Keller in 2001 present a survey with 107 participants mostly from industry and found reasons why practitioners apply software visualization.

  – In decreasing importance the benefits of SoftVis tools are:
    • Savings in time and money
    • Better comprehension of software
    • Increase in productivity and quality
    • Management of complexity
    • To find errors
    • …
9   Software Visualization

• 9.2 Taxonomies and Surveys
  – Two issues were ranked highest on the participants wish list
    • Integration of software visualization tools into other (third-party) tools
    • Better import/export of data and visualization
9 Software Visualization

9.3 Examples of Software Visualization Tools

- Structure,
- Behavior, and
- Evolution of Software (Stephan Diehl)

- For these three we need:
  - Static Program Visualization,
  - Algorithm Animation, and
  - Software Evolution Visualization
Information Visualization WS 2013/14
9 Software Visualization

9.3 Examples of Software Visualization Tools

– aiCall: Static Program Visualization
  • AbsInt GmbH’s analysis tool
  • Produces visualizations of control flow graphs of embedded applications
  • In the graphs the results of a static analysis are shown
  • For each instruction and each function the analysis computes the stack usage
  • Useful for preventing runtime errors due to stack overflow
9.3 Examples of Software Visualization Tools

- aiCall: Static Program Visualization
9  Software Visualization

• 9.3 Examples of Software Visualization Tools
  – Static Program Visualization

Circular Bundle View
Information Visualization WS 2013/14

9  Software Visualization

• 9.3 Examples of Software Visualization Tools
  – Algorithm Animation
9.3 Examples of Software Visualization Tools

- Algorithm Animation

Radix Sort
9 Software Visualization

- 9.3 Examples of Software Visualization Tools
  - Algorithm Animation

Quicksort Algorithm
9.3 Examples of Software Visualization Tools
   - Algorithm Animation

**Merge Sort Algorithm**
9 Software Visualization

- 9.3 Examples of Software Visualization Tools
  - Algorithm Animation

**Shortest Path Algorithm**
9.3 Examples of Software Visualization Tools

- Software Evolution
  - SeeSoft tool developed at AT&T Bell Laboratories to visualize changes and metrics related to evolving large (several million lines of code) and complex software systems.
  - Files represented as rectangles
  - Within each rectangle colored pixels or lines show lines of code
  - Color indicates age of the last modification
  - Blue (cold) used for lines which have not been changed for a long time
  - Red (hot) used for recently changed lines
9.3 Examples of Software Visualization Tools

- Software Evolution


9 Software Visualization

- 9.3 Examples of Software Visualization Tools
  - Software Evolution
  - CVSScan: Visualization of Code Evolution. L. Voinea, A. Telea, and J.J. van Wijk
9.3 Examples of Software Visualization Tools

- Software Evolution
- Gevol: A System for Graph-Based Visualization of the Evolution of Software. C. Collberg, S. Kobourov, J. Nagra, J. Pitts, K. Wampler
9.3 Examples of Software Visualization Tools

- Software Evolution
- The Evolution Matrix: Recovering Software Evolution using Software Visualization Techniques. Michele Lanza
Information Visualization WS 2013/14

9  Software Visualization

- 9.3 Examples of Software Visualization Tools
  - Software Evolution
9 Software Visualization

- 9.4 Static Program Visualization
  - Program text given as a sequence of characters
  - Both text-based and diagrammatic visualization methods

```java
for (int i=0; i<GraphStudy.nodenumer; i++) {
    if (i!=1) {
        int target = (int)(GraphStudy.nodenumer*Math.random());
        while (target==i) {
            target = (int)(GraphStudy.nodenumer*Math.random());
        }
        GraphStudy.edges[i][target] = 1;
        edgecount++;
    }
}
for (int i=2; i<GraphStudy.edges.length; i++) {
    for (int j=0; j<i; j++) {
        int c = 0;
        for (int k=0; k<i; k++) {
            if (GraphStudy.edges[k][j]==1) {
                c++;
            }
        }
        if (GraphStudy.edges[i][j]==1) {
            double prob = 1.0*c/edgecount;
            if (Math.random()<prob) {
                //GraphStudy.edges[i][j] = 1; //edgecount++;
            }
        }
    }
}
```
9 Software Visualization

9.4 Static Program Visualization

- Pretty Printing

```java
for (int i=0; i<GraphStudy.nodenumbers; i++) {
    if (i!=1) {
        int target = (int)(GraphStudy.nodenumbers*Math.random());
        while (target==i) {
            target = (int)(GraphStudy.nodenumbers*Math.random());
        }
        GraphStudy.edges[i][target] = 1;
        edgecount++;
    }
}

for (int i=2; i<GraphStudy.edges.length; i++) {
    for (int j=0; j<i; j++) {
        int c = 0;
        for (int k=0; k<i; k++) {
            if (GraphStudy.edges[k][j] == 1) {
                c++;
            }
        }
        if (GraphStudy.edges[j][k] == 1) {
            c++;
        }
        double prob = 1.0*c/edgecount;
        if (Math.random()<prob) {
            if (GraphStudy.edges[j][i] == 1) {
                GraphStudy.edges[j][i] = 1;
                edgecount++;
            }
        }
    }
}
```
Information Visualization WS 2013/14

9 Software Visualization

• 9.4 Static Program Visualization
  – Pretty Printing
    • Making the structure of a program more explicit
    • Originally restricted to the use of indentation, spaces, and line breaks
    • Declarations can be aligned by tabbing
    • Operator precedence by using different widths of spaces more explicit
    • Advance of technology: Also fonts, font face, and color are used
      – For example: bold face for keywords, italics for comments
      – Different font sizes illustrate nesting
9 Software Visualization

• 9.4 Static Program Visualization
  – Pretty Printing

```
int i,c; while (i<100) if (i%2==0) c++; i++;
```

```
int i,c;
while (i<100)
  if (i%2==0) c++; i++;
```

```
int i,c;
while (i<100)
  if (i%2==0) c++; i++;
```
9.4 Static Program Visualization

- Pretty Printing

```c
int i, c;

while (i < 100)
    if (i % 2 == 0) c++;
    i++;

```  

- A wrong pretty printing can suggest wrong nesting. In pretty printed text on the right the statement `i++` seems to be part of the loop body!
9 Software Visualization

9.4 Static Program Visualization

- Program as Publication
  - In 1984 Donald Knuth (inventor of document typesetting system TEX) introduced the term literate programming
  - This means programs should be readable for the human user and not only for computers
  - Knuth considered programs as works of literature
  - Facilitation of well-documented and typeset programs, the Web tool was developed
9.4 Static Program Visualization

- Program as Publication
  - "I’m pleased that my work on typography, which began as an application of computers to another field, has come full circle and become an application of typography to the heart of computer science."

[Donald Knuth, 1984]
9.4 Static Program Visualization

- Program as Publication
  - Baecker and Marcus investigated the use of typography to increase program readability
  - They produced program books with the same care as other textbooks
  - A typical program book consists of
    - The frontmatter (cover page, title page, abstract, program history, information about the authors, table of contents)
    - First chapter (user documentation, as a tutorial on how to use the program)
    - Second chapter (overview of the program structure by a program map and hierarchy)
9.4 Static Program Visualization

- Program as Publication
  - Program map is a table with thumbnails of each program code page where major function names are emphasized
  - Each subsequent chapter contains pretty printed program code of a source file with comments
- Last chapter provides programmer documentation including installation and maintenance guides
- End of the book consists of several indices like cross-references, caller index, and callee index
- At the back cover page the highlights of the book content are summarized
9.4 Static Program Visualization

- Jackson Diagrams (1975)
  - Programs are decomposed hierarchically
  - Basic elements of Jackson diagrams are actions or operations
  - Actions decomposed into subactions or suboperations
9 Software Visualization

• 9.4 Static Program Visualization
  – Jackson Diagrams
    • A simple operation is drawn as a box

[Diagram: An operation]
9.4 Static Program Visualization

- Jackson Diagrams
  
  A sequence of operations is represented by boxes connected with lines. In the example below, operation A consists of the sequence of operations B, C and D.
9.4 Static Program Visualization

- Jackson Diagrams
  - An iteration is again represented with joined boxes. In addition the iterated operation has a star in the top right corner of its box. In the example below, operation A consists of an iteration of zero or more invocations of operation B.
9.4 Static Program Visualization

- Jackson Diagrams
  - Selection is similar to a sequence, but with a circle drawn in the top right hand corner of each optional operation. In the example, operation A consists of one and only one of operations B, C or D.
9.4 Static Program Visualization

- Jackson Diagrams

```c
int fact(n) {
    if (n>1) {
        nfact = 2;
        for (int i=3;i<=n;i++)
            nfact=nfact*i;
    }
    else {
        nfact = 1;
    }
    return nfact;
}
```
9.4 Static Program Visualization

– Jackson Diagrams
• 9.4 Static Program Visualization
  – Control-Flow Graph or Flowchart (1947)
    • Goldstine and von Neumann introduced control-flow graphs (CFG) to show the structure of programs
    • Rectangular nodes represent events, activities, processes, functions, or statements
    • Diamond nodes contain branch conditions and can have several exits (outgoing edges)
9.4 Static Program Visualization

- Control-Flow Graph or Flowchart (1947)
  - Edges in the graph are drawn as arrows and illustrate a transition from one statement to the next (the flow or control)
  - Nowadays, many new graphical elements have been added and have been standardized as DIN 66001 (flowcharts)
9.4 Static Program Visualization

- Control-Flow Graph or Flowchart (1947)
  - If-else statement

```plaintext
if (the conditional expression is true)
{
    perform these statements
}
else
{
    perform these statements if the conditional is false
}
```
9.4 Static Program Visualization

- Control-Flow Graph or Flowchart (1947)
  - While loop

```c
while (some conditional expression is true) {
    perform the statements located between the braces
}
```
9.4 Static Program Visualization

- Control-Flow Graph or Flowchart (1947)
  - Do while

```java
    do
    {
        perform the statements located between the braces
    } while (this conditional expression is true)
```
9.4 Static Program Visualization

– Control-Flow Graph or Flowchart (1947)
  • For loop

```plaintext
for ( variable initialized ; conditional expression ; variable modifier )
{
    perform these statements while the conditional expression is true
}
```
9.4 Static Program Visualization

- Control-Flow Graph or Flowchart (1947)

- Case/Switch

```java
switch ( variable of interest )
{
    case value1:
        code to be executed.
        break;
    case value2:
        code to be executed.
        break;
    case value3:
        code to be executed.
        break;
    .
    .
    default:
        code to be executed.
        break;
}
```
9 Software Visualization

• 9.4 Static Program Visualization
  – Control-Flow Graph or Flowchart (1947)
    • Exercise: First apply pretty printing and then draw the corresponding flowchart

```c
int f(n) { if (n>1) { v=2; int i=3; while (i<=n) { v=v*i; i=i+1; } } else { v = 1; } return v; }
```

What is the result of f(5) and f(6)? What for f(n)?
9.4 Static Program Visualization

- Nassi-Shneiderman Diagrams
  - Graphical design representation for structured programming
  - Developed in 1972 by Isaac Nassi and graduate student Ben Shneiderman
  - Also called structograms, as they show a program's structures
9.4 Static Program Visualization

- Nassi-Shneiderman Diagrams
  - Top-down design
  - Problem at hand is reduced into smaller and smaller subproblems, until only simple statements and control flow constructs remain
  - Top-down decomposition reflected in a straight-forward way, using nested boxes to represent subproblems.
  - Consistent with the philosophy of structured programming, Nassi–Shneiderman diagrams have no representation for a GOTO statement
Information Visualization WS 2013/14
9 Software Visualization

• 9.4 Static Program Visualization
  – Nassi-Shneiderman Diagrams
    • Nassi–Shneiderman diagrams are (almost) isomorphic with flowcharts.
    • Everything you can represent with a Nassi–Shneiderman diagram you can also represent with a flowchart.
    • For flowcharts of programs, almost everything you can represent with a flowchart you can also represent with a Nassi–Shneiderman diagram.
    • The exceptions are constructs like goto and the C programming language break and continue statements for loops
9 Software Visualization

9.4 Static Program Visualization

- Nassi-Shneiderman Diagrams

- “Not only does this notation help the programmer to think in an orderly manner, it forces him or her to do so... The absence of any representation of the GOTO or branch statement requires the programmer to work without it: a task which becomes increasingly easy with practice.”

[I. Nassi and B. Shneiderman, 1973]
9.4 Static Program Visualization

- Nassi-Shneiderman Diagrams
  - Each instruction in a rectangular structure block
  - Structure blocks are processed from top to bottom
  - Empty structure blocks only allowed in branches
9.4 Static Program Visualization

- Nassi-Shneiderman Diagrams
  - Single conditional (if-else)
  - Only if condition is true, block 1 is processed
  - One block can consist of one or more instructions
  - If condition evaluates to false no instruction is processed
9.4 Static Program Visualization

- Nassi-Shneiderman Diagrams
  - Alternative conditional (if-else)
  - If condition evaluates to true left part is processed
  - If condition evaluates to false right part is processed
Information Visualization WS 2013/14

9 Software Visualization

• 9.4 Static Program Visualization
  – Nassi-Shneiderman Diagrams
    • Nested conditionals (if-else)

```plaintext
if (Condition) {
    "THEN logic";
}
else {
    if (Condition 2) {
        "THEN logic 2";
    }
    else {
        "ELSE logic 2";
    }
}
```
9 Software Visualization

9.4 Static Program Visualization

- Nassi-Shneiderman Diagrams
  - Case/Switch
  - If more than three conditions have to be checked
  - Too many if-else branches were needed otherwise
  - Value of a variable can be checked for its corresponding interval
  - Corresponding case will be processed (switch, select)
  - Case can always be transformed into nested conditionals
9.4 Static Program Visualization
   - Nassi-Shneiderman Diagrams
     - Case/Switch
9.4 Static Program Visualization

- Nassi-Shneiderman Diagrams
  - Do While
  - First test condition evaluated
  - If evaluated to true the instructions are performed one-by-one
9.4 Static Program Visualization

- Nassi-Shneiderman Diagrams
  - Do Until
  - Instructions are executed first
  - After that the exit condition is tested.
9.4 Static Program Visualization

- Nassi-Shneiderman Diagrams

  Exercise:
  - What is computed in this example?
9.4 Static Program Visualization

- Nassi-Shneiderman Diagrams
  - Example for computing an average

```
Accum = 0
Count = 0
READ X
DO WHILE X >= 0
  Count = Count + 1
  Accum = Accum + X
  READ X
END
Average = Accum / Count
WRITE Count, Accum, Average
```
9 Software Visualization

9.4 Static Program Visualization

– Nassi-Shneiderman Diagrams

• Exercise: Draw a Nassi-Shneiderman diagram

```c
int f(n) { if (n>1) { v=2; int i=3; while (i<=n) { v=v*i; i=i+1; } } else { v = 1; } return v; }
```
9.4 Static Program Visualization

- Call relation visualization
  - Which software artifacts are related to each other/call each other?
    - Static Call Graph, Project Hierarchy, Inheritance Hierarchy
Information Visualization WS 2013/14

9 Software Visualization

• 9.4 Static Program Visualization
  – Project Hierarchy Visualization