Software Visualization

CS 695 / SWE 699: Programming Tools
Fall 2023
Today

• Part 1 Lecture (~45 mins)
  • 10 min break
• Part 2: Tech Talks (30 mins)
  • Two tech talks
• Part 3: In-Class Activity (1 hour)
Logistics

• HW 4 due 11/29
Overview

• How can software visualization help?
• Different types of software visualization
Why a diagram is (sometimes) worth ten thousand words

- Diagrams can group together all information that is used together, thus avoiding large amounts of search for the elements needed to make a problem-solving inference.
- Diagrams typically use location to group information about a single element, avoiding the need to match symbolic labels.
- Diagrams automatically support a large number of perceptual inferences, which are extremely easy for humans.

### How information visualization amplifies cognition.

<table>
<thead>
<tr>
<th>Increased Resources</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>High-bandwidth hierarchical interaction</strong></td>
<td>The human moving gaze system partitions limited channel capacity so that it combines high spatial resolution and wide aperture in sensing visual environments (Resnikoff, 1987).</td>
</tr>
<tr>
<td><strong>Parallel perceptual processing</strong></td>
<td>Some attributes of visualizations can be processed in parallel compared to text, which is aerial.</td>
</tr>
<tr>
<td><strong>Offload work from cognitive to perceptual system</strong></td>
<td>Some cognitive inferences done symbolically can be recoded into inferences done with simple perceptual operations (Larkin and Simon, 1987).</td>
</tr>
<tr>
<td><strong>Expanded working memory</strong></td>
<td>Visualizations can expand the working memory available for solving a problem (Norman, 1993).</td>
</tr>
<tr>
<td><strong>Expanded storage of information</strong></td>
<td>Visualizations can be used to store massive amounts of information in a quickly accessible form (e.g., maps).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reduced Search</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Locality of processing</strong></td>
<td>Visualizations group information used together, reducing search (Larkin and Simon, 1987).</td>
</tr>
<tr>
<td><strong>High data density</strong></td>
<td>Visualizations can often represent a large amount of data in a small space (Tufte, 1983).</td>
</tr>
<tr>
<td><strong>Spatially indexed addressing</strong></td>
<td>By grouping data about an object, visualizations can avoid symbolic labels (Larkin and Simon, 1987).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enhanced Recognition of Patterns</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Recognition instead of recall</strong></td>
<td>Recognizing information generated by a visualization is easier than recalling that information by the user.</td>
</tr>
<tr>
<td><strong>Abstraction and aggregation</strong></td>
<td>Visualizations simplify and organize information, supplying higher centers with aggregated forms of information through abstraction and selective omission (Card, Robertson, and Mackinlay, 1991; Resnikoff, 1987).</td>
</tr>
<tr>
<td><strong>Visual schemata for organization</strong></td>
<td>Visually organizing data by structural relationships (e.g., by time) enhances patterns.</td>
</tr>
<tr>
<td><strong>Value, relationship, trend</strong></td>
<td>Visualizations can be constructed to enhance patterns at all three levels (Berin, 1977/1981).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Perceptual Inference</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Visual representations make some problems obvious</strong></td>
<td>Visualizations can support a large number of perceptual inferences that are extremely easy for humans (Larkin and Simon, 1987).</td>
</tr>
<tr>
<td><strong>Graphical computations</strong></td>
<td>Visualizations can enable complex specialized graphical computations (Hutchins, 1996).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Perceptual Monitoring</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Visualizations allow for the monitoring of a large number of potential events if the display is organized so that these stand out by appearance or motion.</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Manipulable Medium</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unlike static diagrams, visualizations can allow exploration of a space of parameter values and can amplify user operations.</strong></td>
<td></td>
</tr>
</tbody>
</table>

Designing an information visualization

Tufte’s principles of graphical excellence

- show the **data**
- induce the viewer to think about the substance rather than the methodology
- avoid distorting what the data have to say
- present **many** numbers in a small space
- make large data sets **coherent**
- encourage the eye to **compare** different pieces of data
- reveal data at several levels of detail, from overview to fine structure
- serve reasonable clear **purpose**: description, exploration, tabulation, decoration
Interactive visualizations

• Users often use iterative process of making sense of the data
• Answers lead to new questions
• Interactivity helps user constantly change display of information to answer new questions
• Should offer visualization that offers best view of data moment to moment as desired view changes
How software visualizations may help

• Offer information that helps developers to answer questions

• Facilitate easier navigation between artifacts containing relevant information
Key questions for software visualization design

• Do you *really* need a visualization?
  • If you know the developer’s question, can you answer it more simply *without* a visualization?

• Anti-pattern: show all the information, let user find patterns
  • In other domains (e.g., data analytics), visualization is a tool for data exploration and understanding dataset.
  • **Not true for SE:** developers want to complete tasks, finding patterns often not relevant

• How much context do you need?
  • More context —> more information to sort through
  • Less context —> more direct
Some popular forms of software visualizations

- Code
  - Iconographic representation of code text

- Algorithm & object structure visualizations
  - Depictions of data value changes over time
  - Runtime snapshots of object reference structure

- Module structure
  - Static views of module properties & dependencies (e.g., calls, references)

- Runtime software structure
  - Run time view of software elements that exist at runtime and messages sent (e.g., HTTP requests)

- Function calls
  - Dynamic and static depictions of function calls
Code visualizations

• Offer overview of source code

• Identify relevant sources lines matching some property
  • e.g., changed in a commit, passing a test, with a compiler warning

• Represent lines of iconographically
  • e.g., colored lines
AT&T Bell Labs [Eick, 1992]
Visualization for performance
“Hot spots” in red
Large volumes of code
Image is of 15,255 LOC
Up to 50,000 LOC
Can indent like original source files
Also, recently changed,
Version control systems
Static, dynamic analyses
Interactive investigation

Tarantula

Color – code coverage
- Red – failed test case
- Green – past test case
- Yellow – hue is % of test cases passing

Industry Use: Eclipse Markers
Industry use: Visual Studio Code Minimap

```javascript
/*
 * Copyright (c) Microsoft Corporation. All rights reserved.
 * Licensed under the MIT License. See License.txt in the project root for license information.
 */

'use strict';

import 'vs/css!./minimap';
import { ViewPart } from 'vs/editor/browser/view/viewPart';
import { ViewContext } from 'vs/editor/common/view/viewContext';
import { IRenderingContext, IRestrictedRenderingContext } from 'vs/editor/common/
import { getOrCreateMinimapCharRenderer } from 'vs/editor/common/view/runtimeMini
import * as browser from 'vs/base/browser/browser';
import * as dom from 'vs/base/browser/dom';
import { MinimapCharRenderer, MinimapTokensColorTracker, Constants } from 'vs/edit
import * as editorCommon from 'vs/editor/common/editorCommon';
import { CharCode } from 'vs/base/common/charCode';
import { IViewLayout, ViewLineData } from 'vs/editor/common/viewModel/viewModel';
import { ColorId } from 'vs/editor/common/modes';
import { FastDomNode, createFastDomNode } from 'vs/base/browser/fastDomNode';
import { IDisposable } from 'vs/base/common/lifecycle';
import { EditorScrollbar } from 'vs/editor/browser/viewParts/editorScrollbar/edit
```
Algorithm & object structure visualizations

- Depict runtime state at a snapshot or over time
  - e.g., elements in a collection, numeric values

- Often focused on teaching basic algorithms (e.g., sorting algorithms, linked list insertion)

(Section adapted from Software Visualization, Lecture by Brad A. Myers, Spring 2011)
Sorting out Sorting
Incense

First to automatically create viz. of data structures

Produce pictures “like you might drawn them on a blackboard”

Goal: help with debugging

Figure 14. ARRAY [1..4] OF POINTER with two POINTERS referring to the same value.

Figure 15. This erroneous tree structure demonstrates that a pointer to previously displayed object does not generate a new copy. The second arrow is drawn to the first occurrence.

Figure 16. Pointer to value inside a record (a) does not get confused with a pointer to the record itself (b).

Figure 17. Incense display for RECORD [int: INTEGER, p1: POINTER TO CARDINAL].

Figure 18. Artist hierarchy that would be created for: rec: RECORD [p1: POINTER TO CARDINAL, int: INTEGER]; (This figure was not created by Incense).

Figure 19. Demonstration of the advantage of curved lines used in Incense (a) over straight lines (b). The control points used to specify the spline are shown as black squares in (a).

Brown University Algorithm Simulator and Animator (BALSA)

Major interactive integrated system
Extensively used for teaching at Brown Univ.
Lots of algorithms visualized
Architecture for attaching the graphics with code
Still required significant programming for each viz.
Marc followed up with Zeus ('91) at DEC SRC

Transition-based Animation Generation
(TANGO)

Smooth animations between states
Paths & transitions
Make it easier to author algorithm visualizations
Events inserted into the code tied to animations

Figure 9. Superimposed sequence of frames from the bin-packing animation.

Figure 2. Tango animation of a producer-consumer ring buffer.

Data Display Debugger

https://www.gnu.org/software/ddd/

Over 2.5 million people in over 180 countries have used Python Tutor to visualize over 20 million pieces of code.
Module Views

• Depict static structure of modules (e.g., files, folders, packages)

• Often depicts dependencies between modules

• Focus on reverse engineering tasks, refactoring tasks, other architecture related tasks
SHriMP Demonstration

Visualizing a Java Program
Code Crawler (Polymetric Views)

GitHub Repo-Visualization

https://githubnext.com/projects/repo-visualization
Runtime Structure Views
Function calls

- Depict function invocations

- Could be runtime view (specific execution) or static view (all possible executions)

- Many decisions about what to show & how to show it
  - Code centric? Timeline centric?
  - Show all functions? Show some functions? Which ones?
  - What information about functions to depict? Order, time, asynchronicity, …
Reacher
10 min break
Tech Talks
In-Class Activity

- In groups of 2 or 3, try out CodeSee.io
  - Go to CodeSee.io and sign up (use the free Community Plan)
  - Find a codebase that you can run CodeSee on (e.g., your 695 project)
  - Build a CodeMap for your codebase
  - Write a reflection on your experiences using CodeSee:
    - Is it helpful in understanding your project? What tasks would it help with? What questions does it help you answer?
    - What's hardest to use about the tool? What questions does it not help answer? What information would you like to see that it doesn't currently provide?
- Submission
  - Submit a pdf with your reflection through Blackboard. 1 submission per group. Due 7:10pm today.