Software Visualization

SWE 795, Spring 2017
Software Engineering Environments
Today

• Part 1 (Lecture)(~60 mins)
  • Software visualization

• Part 2 (In class activity)(~30 mins)
  • Sketch a software visualization

• Break!

• Part 2 (Discussion)(45 mins)
  • Discussion of readings
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.

### Increased Resources

| High-bandwidth hierarchical interaction | 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). |
| Parallel perceptual processing | Some attributes of visualizations can be processed in parallel compared to text, which is aerial. |
| Offload work from cognitive to perceptual system | Some cognitive inferences done symbolically can be recoded into inferences done with simple perceptual operations (Larkin and Simon, 1987). |
| Expanded working memory | Visualizations can expand the working memory available for solving a problem (Norman, 1993). |
| Expanded storage of information | Visualizations can be used to store massive amounts of information in a quickly accessible form (e.g., maps). |

### Reduced Search

| Locality of processing | Visualizations group information used together, reducing search (Larkin and Simon, 1987). |
| High data density | Visualizations can often represent a large amount of data in a small space (Tufte, 1983). |
| Spatially indexed addressing | By grouping data about an object, visualizations can avoid symbolic labels (Larkin and Simon, 1987). |

### Enhanced Recognition of Patterns

| Recognition instead of recall | Recognizing information generated by a visualization is easier than recalling that information by the user. |
| Abstraction and aggregation | 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). |
| Visual schemata for organization | Visually organizing data by structural relationships (e.g., by time) enhances patterns. |
| Value, relationship, trend | Visualizations can be constructed to enhance patterns at all three levels (Bertin, 1977/1981). |

### Perceptual Inference

| Visual representations make some problems obvious | Visualizations can support a large number of perceptual inferences that are extremely easy for humans (Larkin and Simon, 1987). |
| Graphical computations | Visualizations can enable complex specialized graphical computations (Hutchins, 1996). |

### Perceptual Monitoring

| Visualizations can 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. |

### Manipulable Medium

| Unlike static diagrams, visualizations can allow exploration of a space of parameter values and can amplify user operations. |

---

Designing an information visualization

Marks’ graphical properties

- Quantitative (Q),Ordinal (O), Nominal (N)
- Filled circle - good; open circle - bad
Effectiveness of graphical properties

- Quantitative (Q), Ordinal (O), Nominal (N)
- Filled circle - good; open circle - bad

<table>
<thead>
<tr>
<th>Extent</th>
<th>Spatial (Position)</th>
<th>Q</th>
<th>O</th>
<th>N</th>
<th>Object Grayscale</th>
<th>Q</th>
<th>O</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Differential</td>
<td>Orientation</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>Color</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Texture</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Shape</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
</tbody>
</table>
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)

- 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 iconagraphically
  • 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
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

https://www.youtube.com/watch?v=SJwEwA5gOkM
First to automatically create viz. of data structures

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

Goal: help with debugging

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

PECAN

Steven Reiss at Brown’s code & data visualization systems
Take advantage of new Apollo workstation capabilities
PECAN (1985) – automatic graphics about the program
Multiple views
Integrates Balsa data visualization
Syntax directed editing
Drag and drop
Flowcharts of code
Code highlighting while executing
Data viz. like Incense
Incremental compilation
Could handle up to 1000 LOC

Friendly Integrated Environment for Learning and Development (FIELD)

Field (1990) – IDE, wrappers for Unix tools
Code and data viz.
Message-based (control) integration
Basis for most other Unix IDEs
Widely used
Followed by DESERT, ...

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

Data Display Debugger

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

PythonTutor

http://pythontutor.com/

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
Fig. 3. A SHriMP View of a program which implements a Hangman game. The main subsystems (Control, Setup, Gameplay, Input, Output, GlobalVars and Dice) are shown in this view. A fisheye view of the Gameplay subsystem provides more detail since it is shown larger than the other subsystems. The maintainer can browse the source code by following hyperlinks within an architectural view of the entire program.

Lattix (Design Structure Matrices)

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, …

https://www.youtube.com/watch?v=FzMI4Zu2tps
Theseus

https://www.youtube.com/watch?v=qnwXX510E2Q

In Class Activity

• Form groups of 2
  • Sketch a software visualization
  • You should decide
    • What is the task you are supporting
    • What information do developers need for this task
    • How does your visualization help developers to obtain this information more easily
    • What context is (or is not) visualized? Why is the specific visualization chosen?
  • Illustrate your visualization with two or more examples of its output