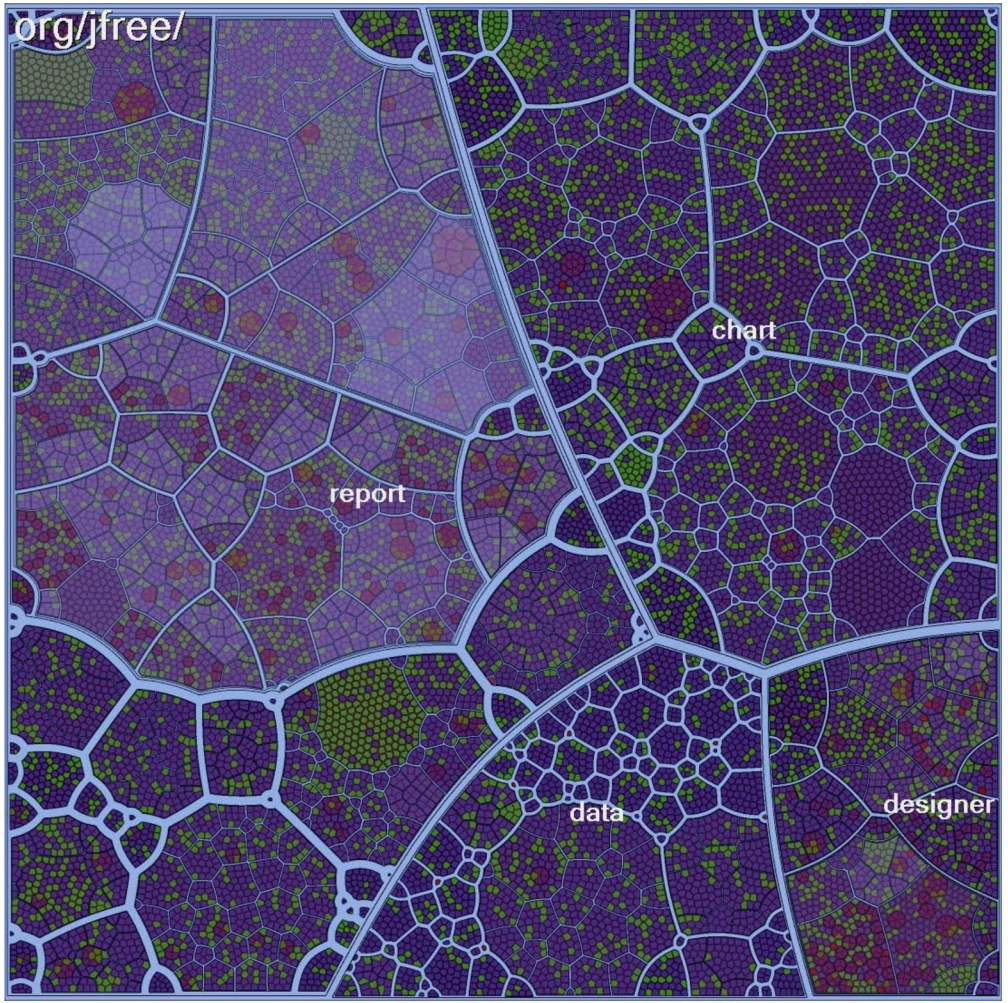
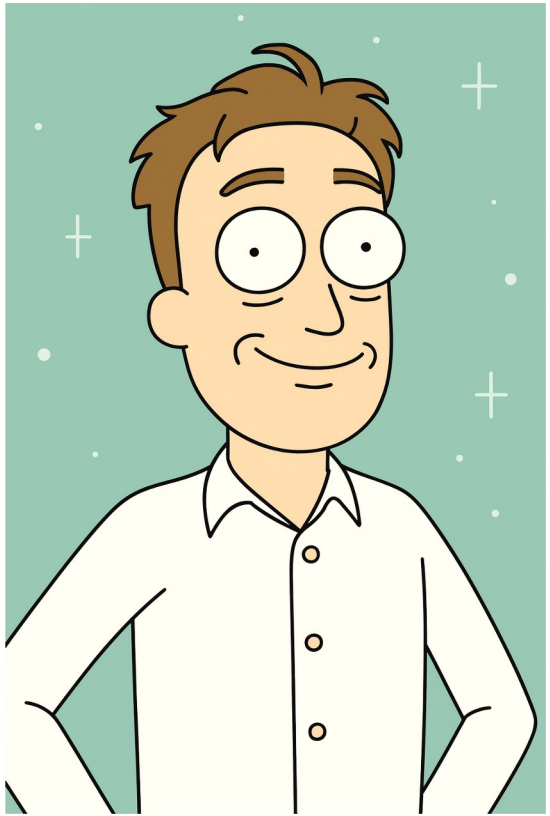
Software Visualization

Marius Irgens, PhD Candidate



About Me

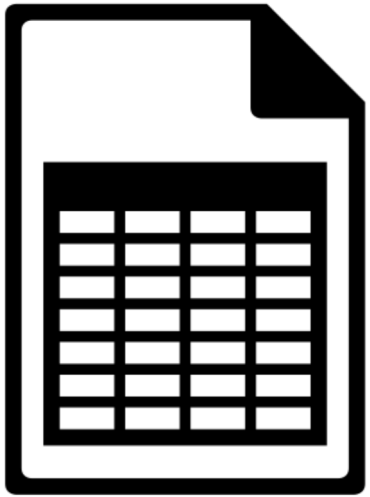
- **Name:** Marius Irgens
- **Role:** Ph.D. candidate at the University of Oslo
- **Supervisor:** Antonio Martini
- **Education:** Master’s degree in Visualization
- **Research:** Visualization of Technical Debt

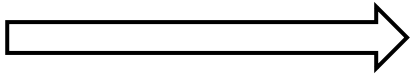


What is Visualization?

Visual representation of data

Mapping data to graphics



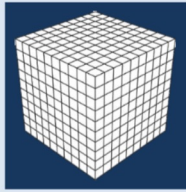




Main Areas of Visualization

VOLUME VISUALIZATION

FLOW VISUALIZATION



inherent spatial reference

no clear categorical difference
distinction increasingly blurry

2D/3D/4D

nD

INFORMATION VISUALIZATION

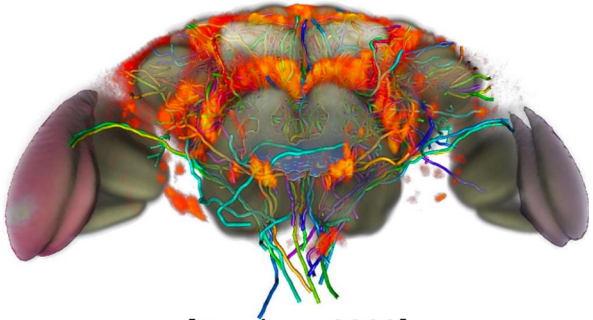
VISUAL ANALYTICS



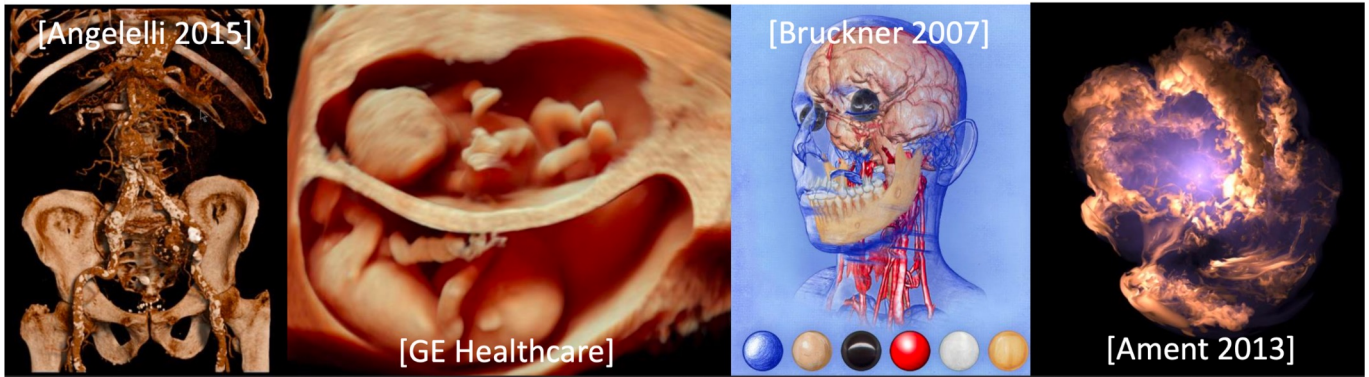
no preferred spatial reference

Volume Visualization

- Visualization of scalar fields
- Important in medicine, biology, geoscience, engineering, …

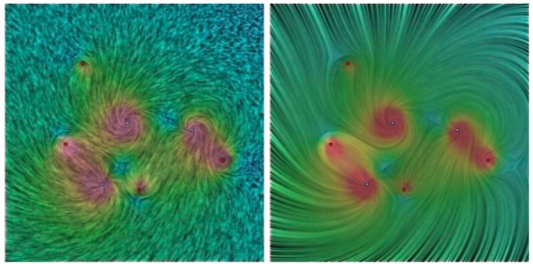


[Bruckner 2009]



Flow Visualization

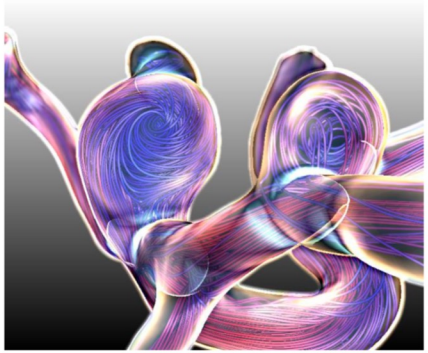
- Visualization of vector fields
- Data typically from computational fluid dynamics (CFD) simulations



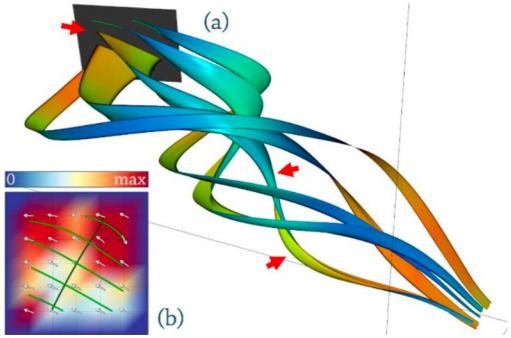
[van Wijk 2000]



[Laramee 2008]



[Lawonn 2014]

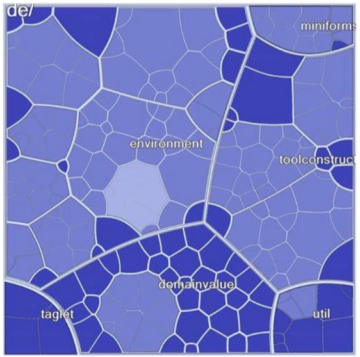


[Brambilla 2015]

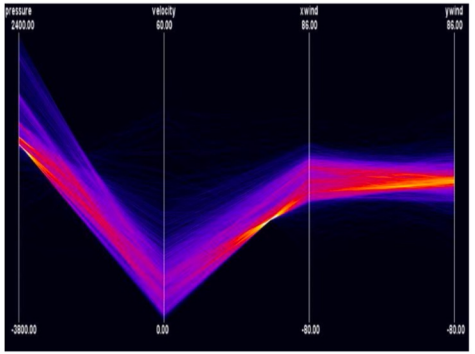
Information Visualization

- High-dimensional data, hierarchical data, textual data, …
- Many different techniques

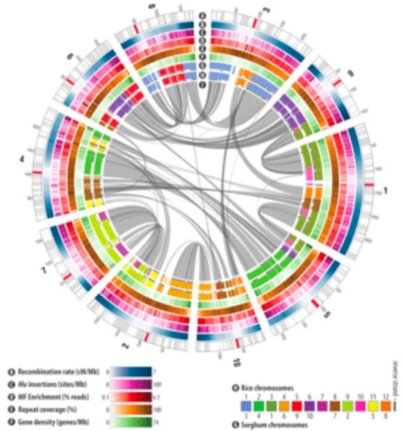




[Balzer 2005]



[Heinrich 2013]



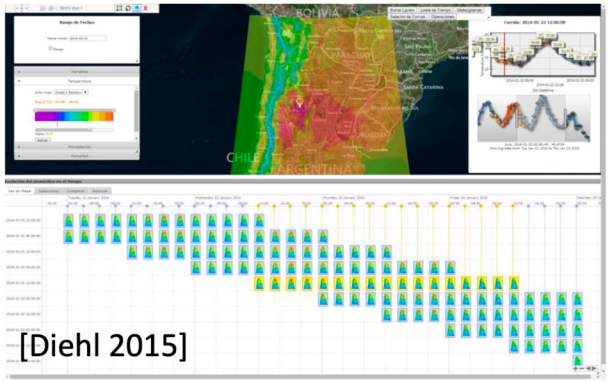
[Shnable 2009]

Visual Analytics

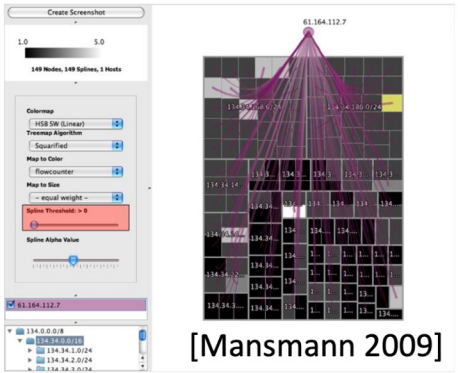
- Emphasizes the combination of computational and interactive visual techniques



[SAS Visual Analytics]



[Diehl 2015]



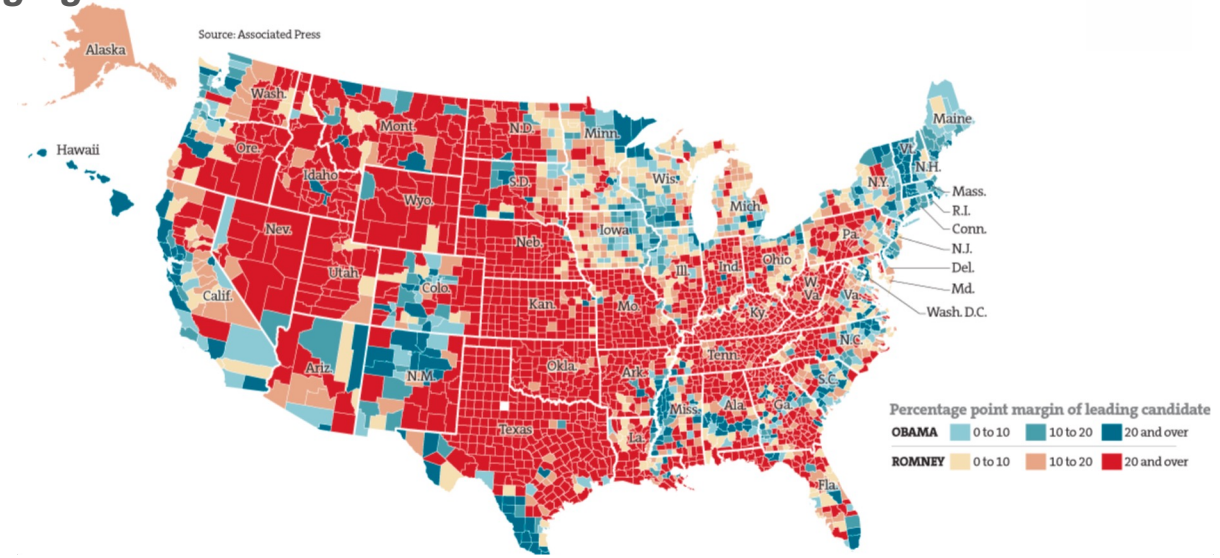
[Mansmann 2009]

Why Visualize?

Leverage human perception: Our visual system evolved to detect patterns and anomalies quickly

High bandwidth: High amounts of text and numbers can be compressed to expressive visual channels

Visualization accelerates knowledge gain



Van Wijk (2005): The value of visualization

Example

How many times does the letter V occur in this string?

MTHIVLWYADCEQGHKILKMTWYN
ARDCAIREQGHLVKMFPSTWYARN
GFPSVCEILQGKMFPSNDRCEQDIFP
SGHLMFHKMVPSTWYACEQTWRN

Example

How many times does the letter V occur in this string?

MTHIVLWYADCEQGHKILKMTWYN
ARDCAIREQGHLVKMFPSTWYARN
GFPSVCEILQGKMFPSNDRCEQDIFP
SGHLMFHKMVPSTWYACEQTWRN

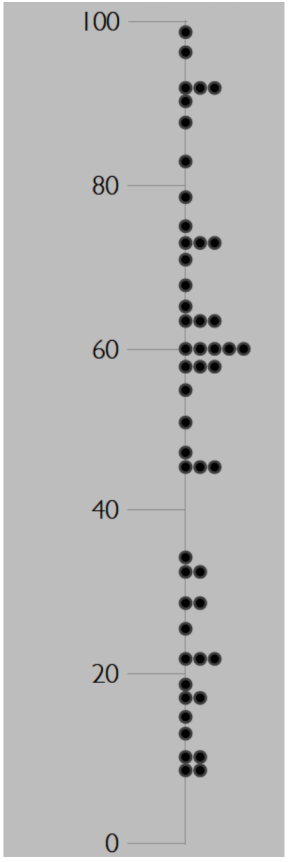
Another example

Which of these 50 numbers occurs most often?

15	19	60
33	11	75
57	34	79
18	51	92
73	22	13
71	60	22
17	10	68
73	18	55
65	46	29
60	73	22
46	92	97
10	58	46
57	17	83
26	99	33
88	92	60
91	29	57
96	12	47

Another example

Which of these 50 numbers occurs most often?



Alternatives

AI methods

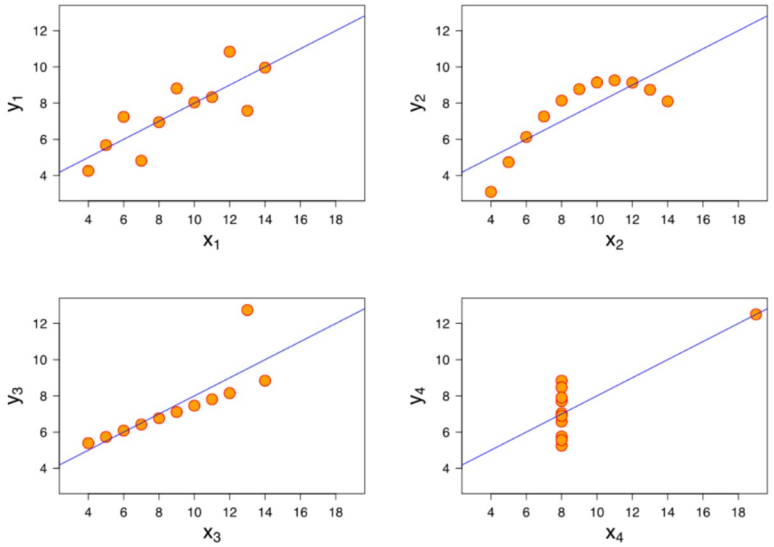
AI methods are often inaccurate and hard to control

Statistical analysis

Descriptive statistics will abstract the data, removing details

Anscombe's quartet

x_1	y_1	x_2	y_2	x_3	y_3	x_4	y_4
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89



same **mean** of x and y (9 and 7.5)
same **variance** of x and y (11 and 4.12)
same **correlation** between x and y (0.816)
same **linear regression line** (y = 3 + 0.5x)

Anscombe (1973): Graphs in Statistical Analysis

Part 1

Visualization Design

*“computer-based visualization systems provide **visual representations of datasets** designed to help people **carry out tasks more effectively**”*

Tamara Munzner



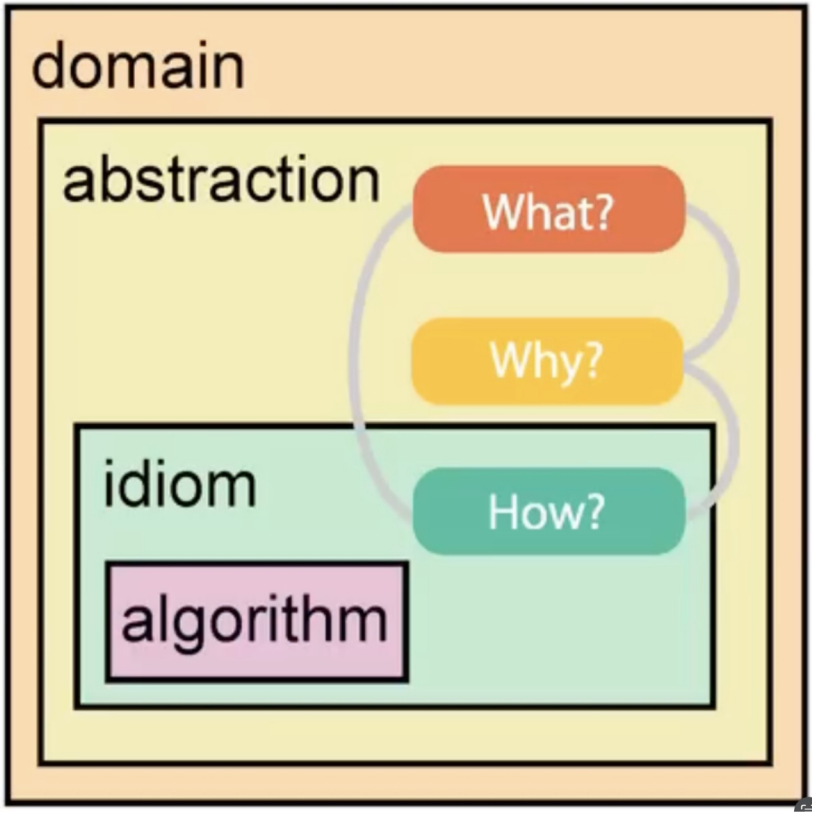
Problem-Driven Visualization Design

Domain Problem Characterization
Who are the target users, and what do they need?

Data and Task Abstraction
Translate from domain to visualization vocabulary:
What is shown? Data abstraction.
Why is the user looking at it? Task abstraction.

Visual Encoding and Interaction Techniques
How is it shown?
Visual encoding idiom: How to draw.
Interaction idiom: How to manipulate.

Algorithm
How are the visual encodings computed?



Munzner (2009): A Nested Model of Visualization Design and Validation



Data Abstraction: What is Shown

Focusing on those that are important for Software Visualization

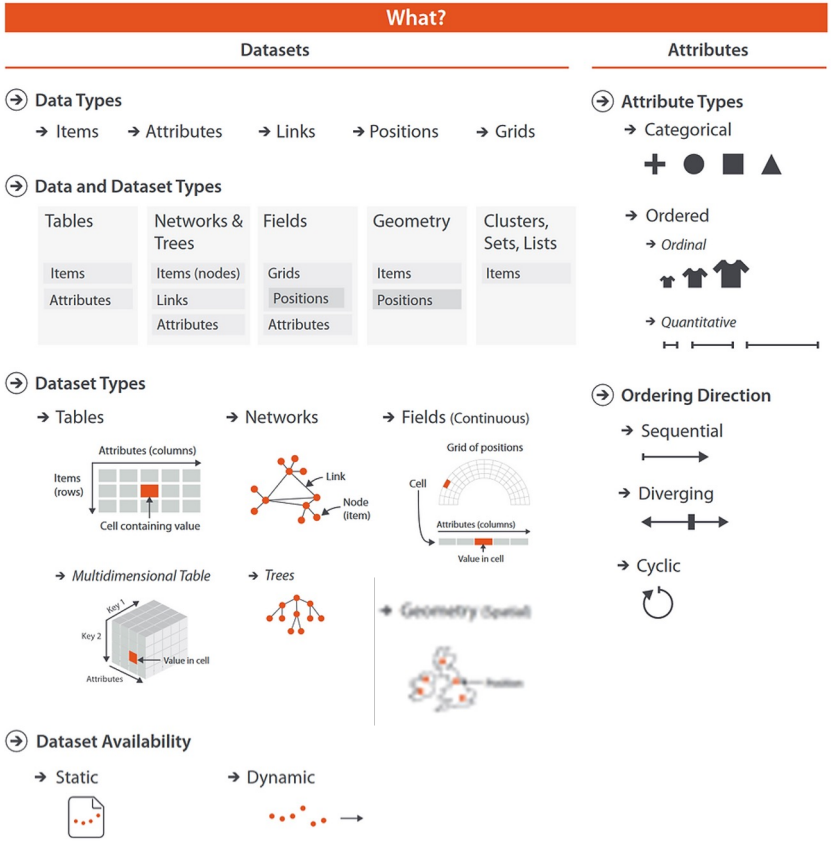
Data Types
Items and Attributes

Dataset Types
Tables, Networks and Trees

Attribute Types
Categorical, Ordinal and Quantitative

Ordering Direction
Sequential and Diverging

*An item **has** attributes, and there are different types of attributes, with different ordering directions.*

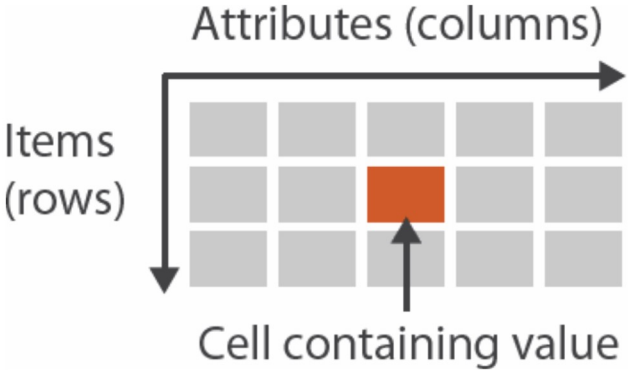


Tables

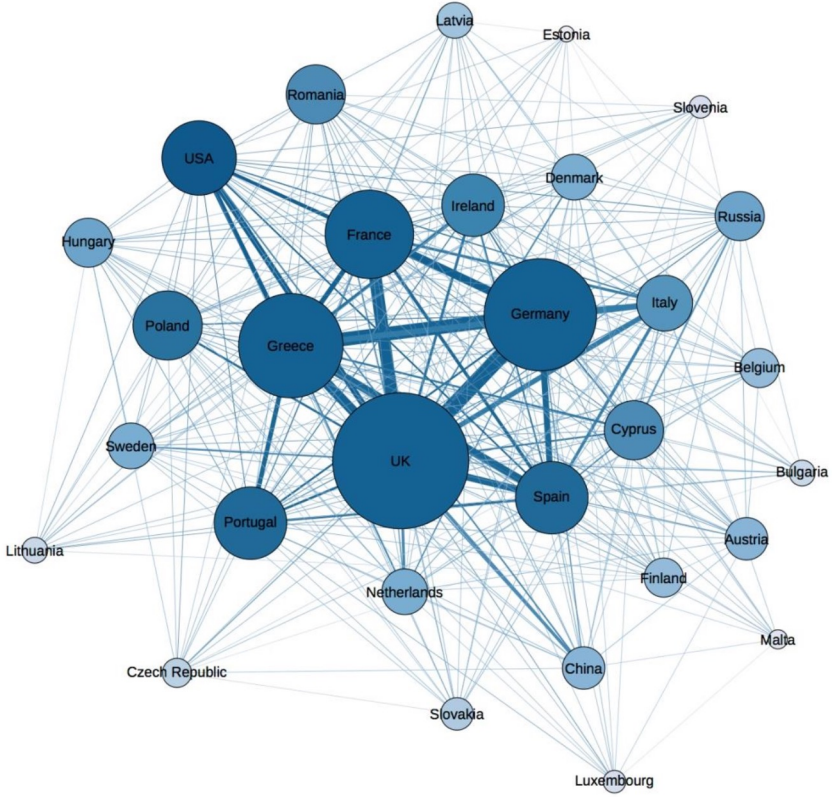
Attributes

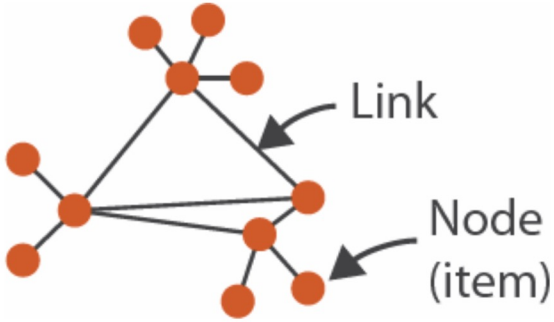
	1	2	3	4	5	6	7	8
1	OBS.	TOWN	TOWN#	LON	LAT	MEDV	RAD	CRIM
2	1	Nahant	0	-70.955	42.255	24	1	0.00632
3	2	Swampscc	1	-70.95	42.2875	21.6	2	0.02731
4	3	Swampscc	1	-70.936	42.283	34.7	2	0.02729
5	4	Marblehe	2	-70.928	42.293	33.4	3	0.03237
6	5	Marblehe	2	-70.922	42.298	36.2	3	0.06905
7	6	Marblehe	2	-70.9165	42.304	28.7	3	0.02985
8	7	Salem	3	-70.936	42.297	22.9	5	0.08829
9	8	Salem	3	-70.9375	42.31	27.1	5	0.14455
10	9	Salem	3	-70.933	42.312	16.5	5	0.21124
11	10	Salem	3	-70.929	42.316	18.9	5	0.17004
12	11	Salem	3	-70.935	42.316	15	5	0.22489
13	12	Salem	3	-70.944	42.317	18.9	5	0.11747
14	13	Salem	3	-70.951	42.306	21.7	5	0.09378
15	14	Lynn	4	-70.9645	42.292	20.4	4	0.62976
16	15	Lynn	4	-70.972	42.287	18.2	4	0.63796
17	16	Lynn	4	-70.9765	42.294	19.9	4	0.62739
18	17	Lynn	4	-70.987	42.2985	23.1	4	1.05393
19	18	Lynn	4	-70.978	42.285	17.5	4	0.7842
20	19	Lynn	4	-70.9925	42.2825	20.2	4	0.80271
21	20	Lynn	4	-70.988	42.2776	18.2	4	0.7258
22	21	Lynn	4	-70.9835	42.277	13.6	4	1.25179

Items



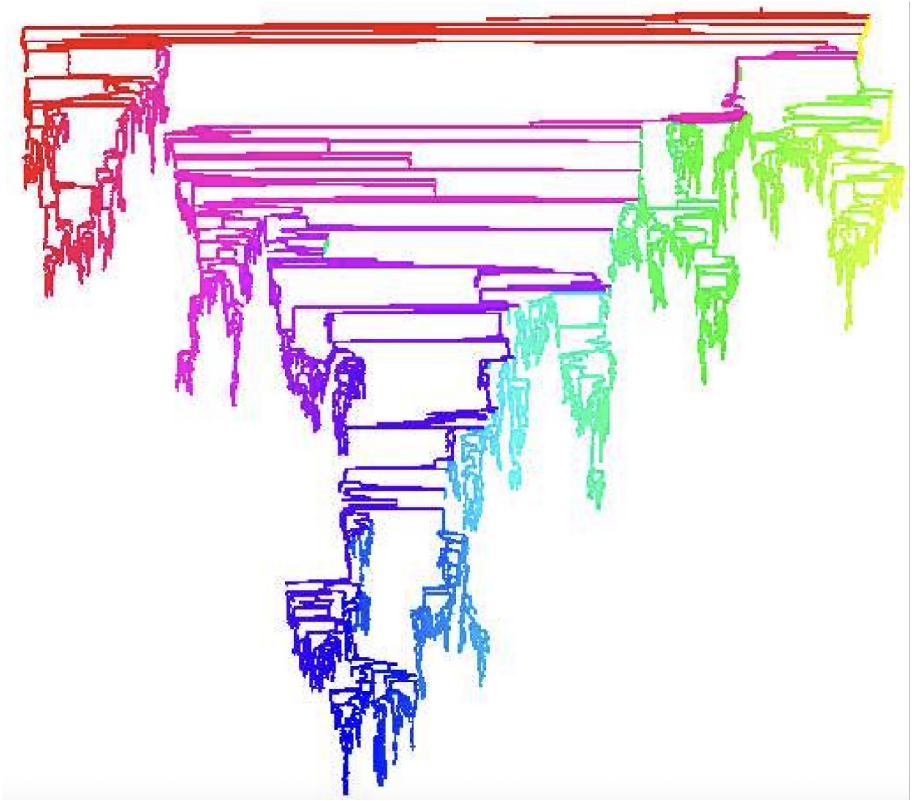
Networks

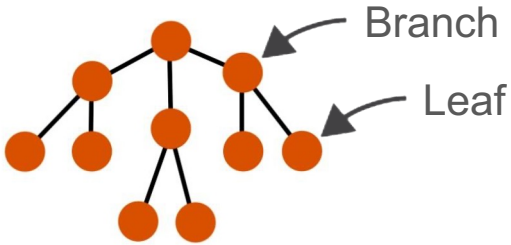

Latvia
Estonia
Romania
Slovenia
USA
Denmark
Ireland
Russia
France
Hungary
Poland
Germany
Italy
Greece
Belgium
Cyprus
Sweden
UK
Bulgaria
Spain
Portugal
Austria
Lithuania
Finland
Netherlands
Malta
China
Czech Republic
Slovakia
Luxembourg


Link
Node
(item)



Trees





Task Abstraction: Why is the user looking at it

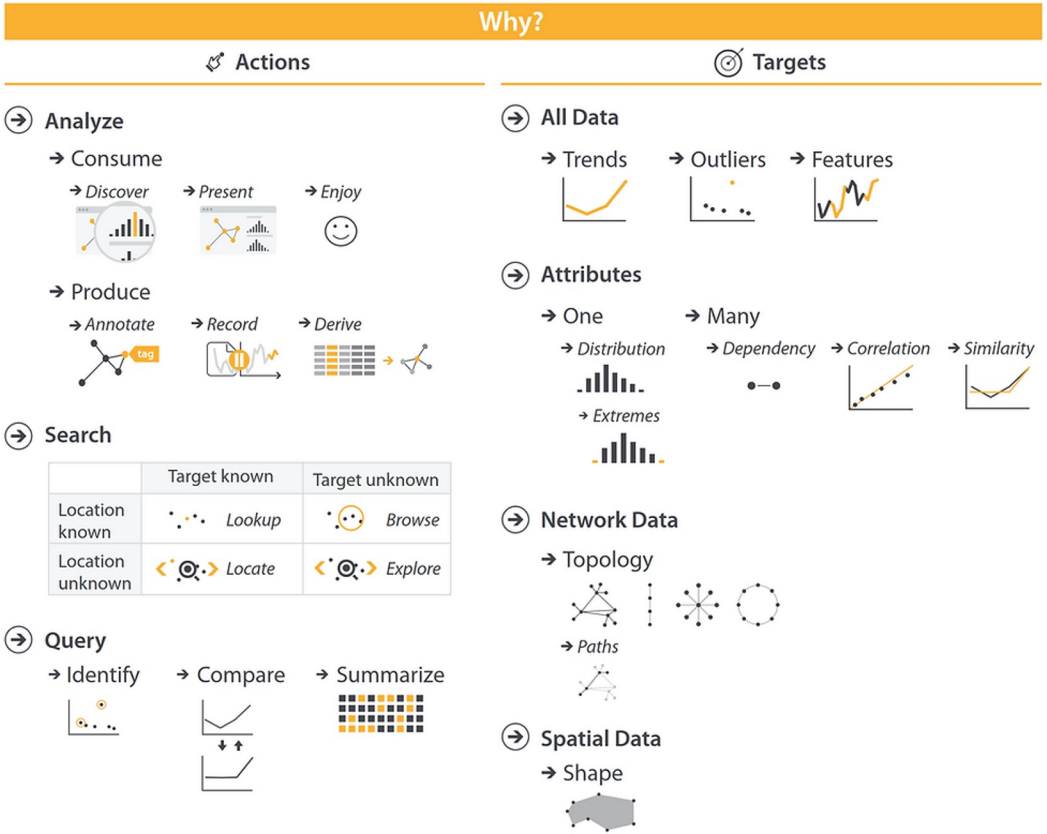
Domain problem characterization => Visualization specific tasks

Action-Target Pairs

Defines core visualization tasks

Examples:

- Discover Outliers
- Compare Features
- Present Trends
- Summarize Distributions
- Compare Topologies
- Identify Paths



Munzner et al. (2013): A Multi-Level Typology of Abstract Visualization Tasks

How: Visual Encoding and Interaction

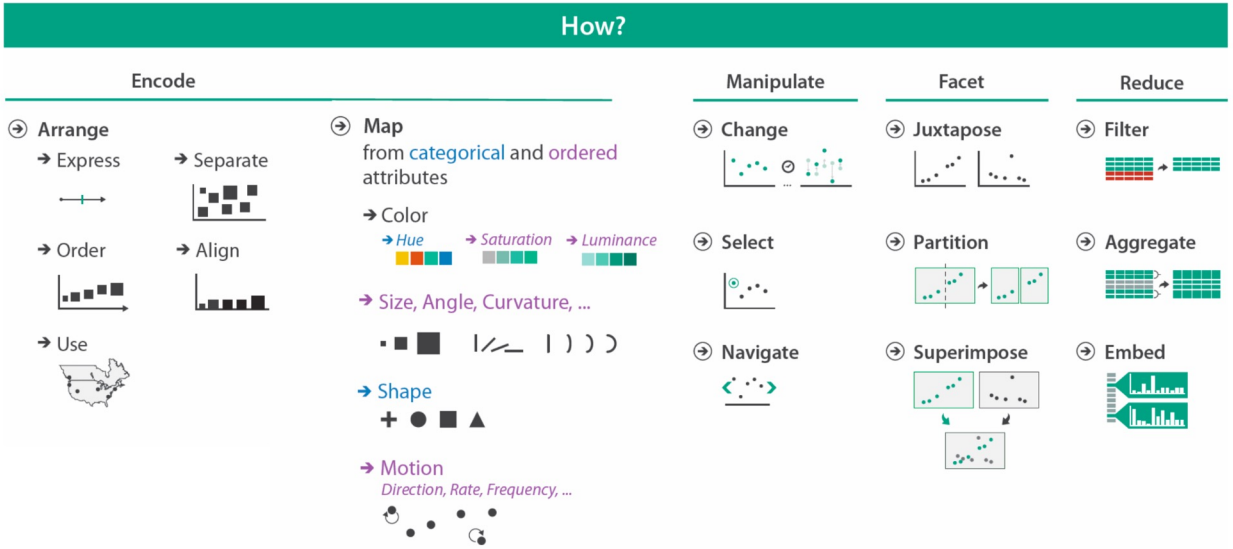
Big Topic!

To big to cover in a short introduction

We will focus on encoding



If you are interested in learning more, get this book!



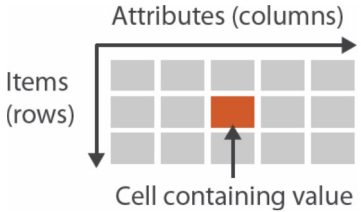
Marks and Channels

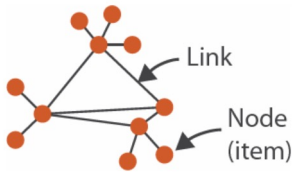
Marks

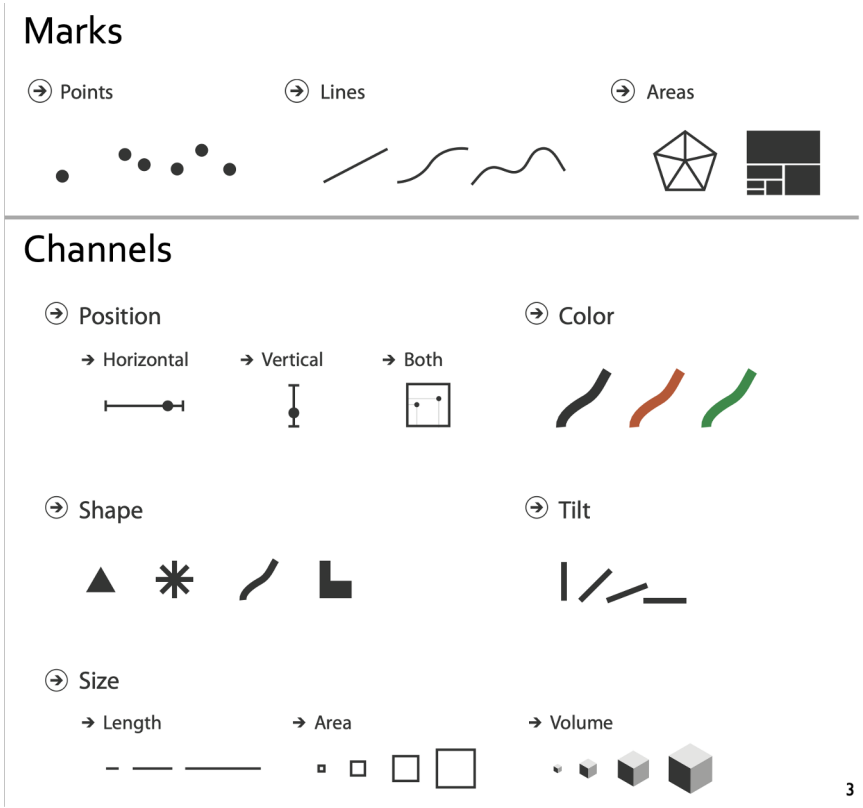
Represents *items* and *links*

Channels

Controls the appearance of marks
Express its *attributes*

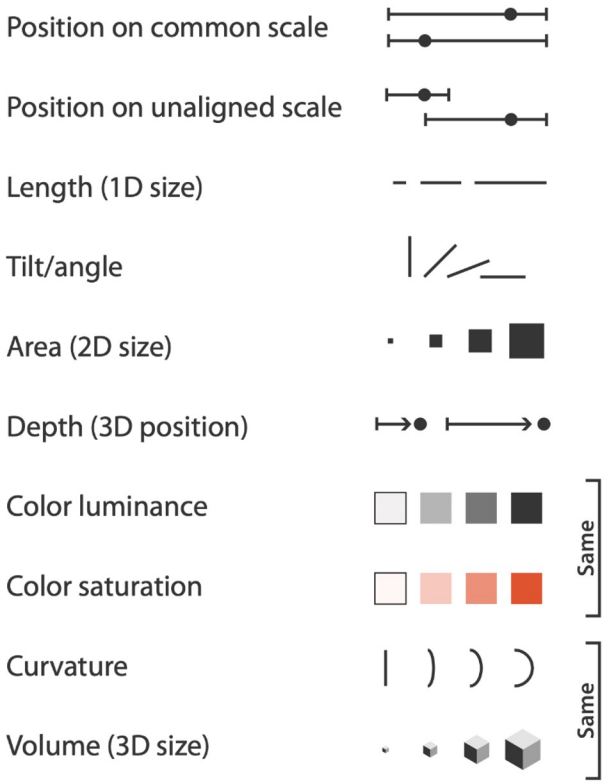






Channels Ranked

Magnitude Channels: Ordered Attributes



Identity Channels: Categorical Attributes

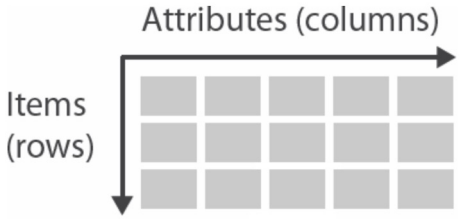


Expressiveness
Match channel types with attribute types

Effectiveness
Use the best channels for the most important attributes

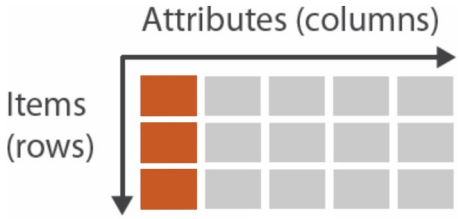


Channels Ranked





Channels Ranked



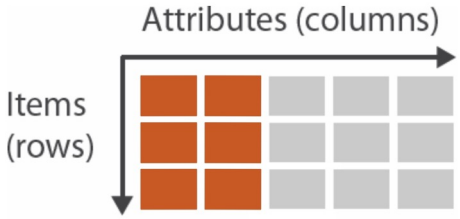
idiom: bar chart



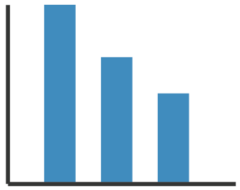
1 channel:
vertical position

mark: line

Channels Ranked



idiom: bar chart



1 channel:
vertical position

idiom: scatterplot



2 channels:
vertical position
horizontal position

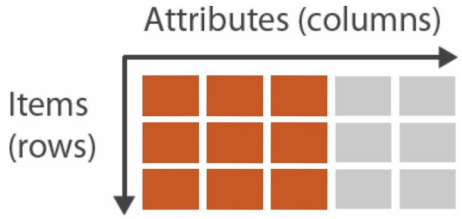




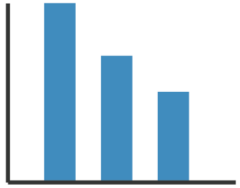
mark: line

mark: point

Channels Ranked



idiom: bar chart



1 channel:
vertical position

mark: line

idiom: scatterplot



2 channels:
vertical position
horizontal position

mark: point

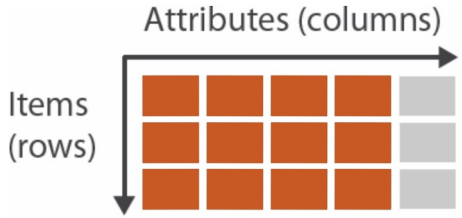


3 channels:
vertical position
horizontal position
color hue

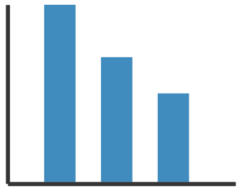
mark: point



Channels Ranked



idiom: bar chart



1 channel:
vertical position

mark: line

idiom: scatterplot



2 channels:
vertical position
horizontal position

mark: point



3 channels:
vertical position
horizontal position
color hue

mark: point



4 channels:
vertical position
horizontal position
color hue
size (area)

mark: point

Bar Charts

Marks

Lines

Channels

Position on a common scale (height/length)

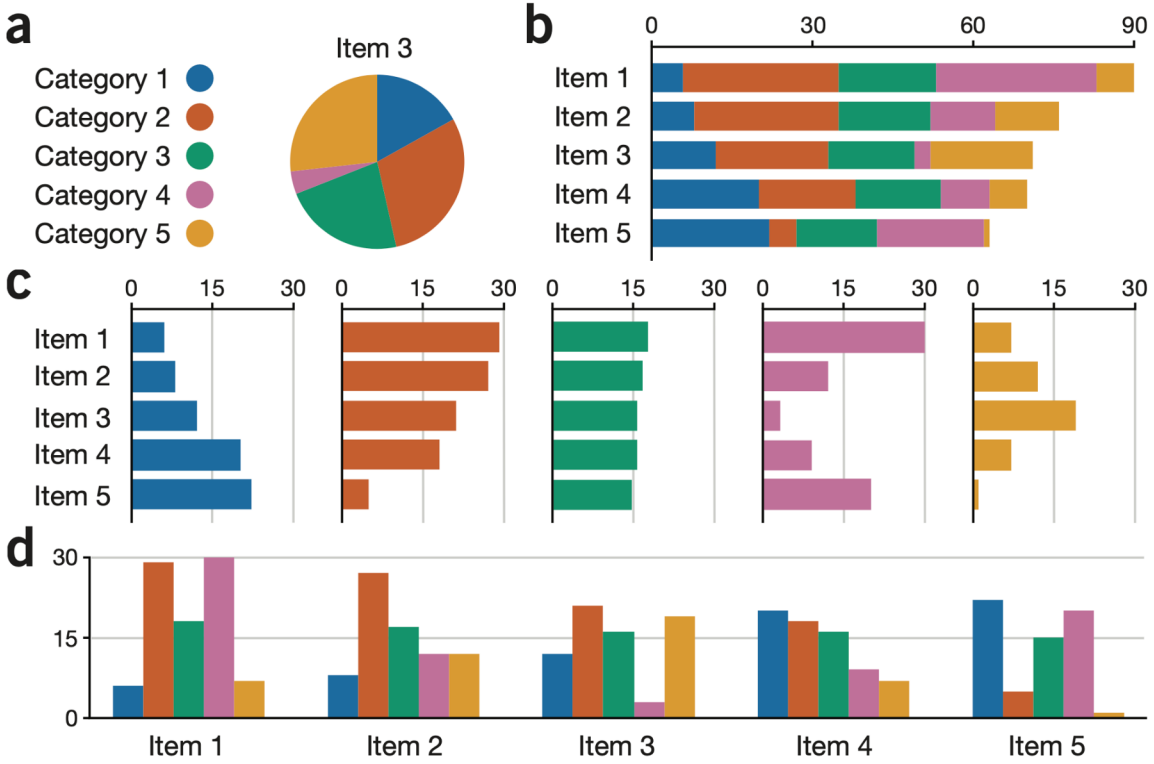
Bar Charts can express categorical attributes in many ways by smart use of colors and groupings

a) Values in different categories are difficult to compare in pie charts

b) Stacked bar charts enable comparison of overall values across items

c) Layered bar charts support comparison of values within categories

d) Grouped bar charts allow comparison of values across categories



Streit & Gehlenborg (2014): Bar charts and box plots

Box Plots

Marks

Lines and points

Channels

Box Plots combine multiple marks that all utilize positions on a common scale

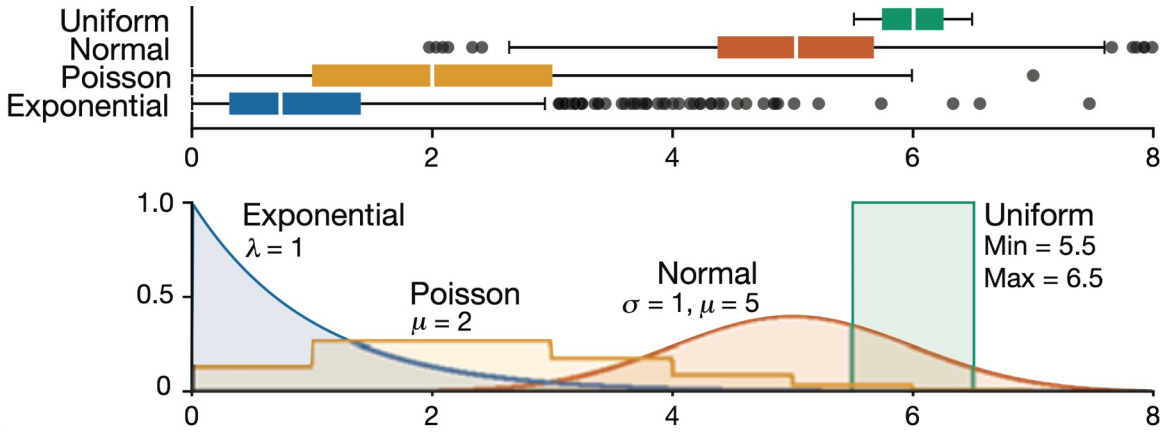
Bar: interquartile Range

Vertical line: Median or Mean value

Whiskers: Minimum and maximum values

Dots: Outliers

So, multiple marks that exploits the same channel, juxtaposed on top of each other



Streit & Gehlenborg (2014): Bar charts and box plots

Line Charts

Marks

Points, Lines

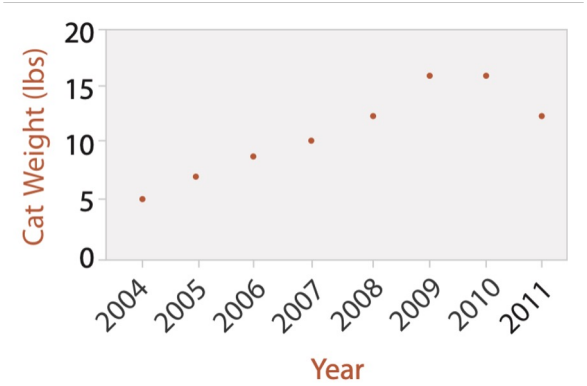
Channels

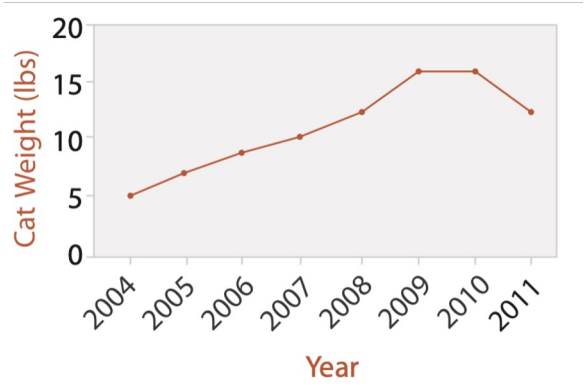
Position on a common scale, angle

Line charts adds a line mark to represent a continuous relationship between sample points

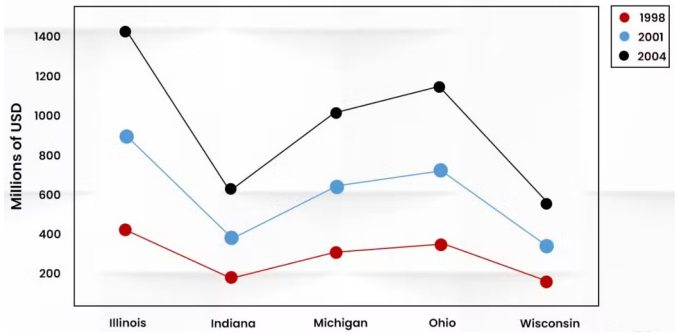
The line's angle expresses the velocity of change

When using line charts, both attributes should be continuous. If one attribute is categorical, you probably want a bar chart!





Wrong (Probably). Is "0.5 Illinois, 0.5 Indiana" an expected value?



Scatterplots

Marks

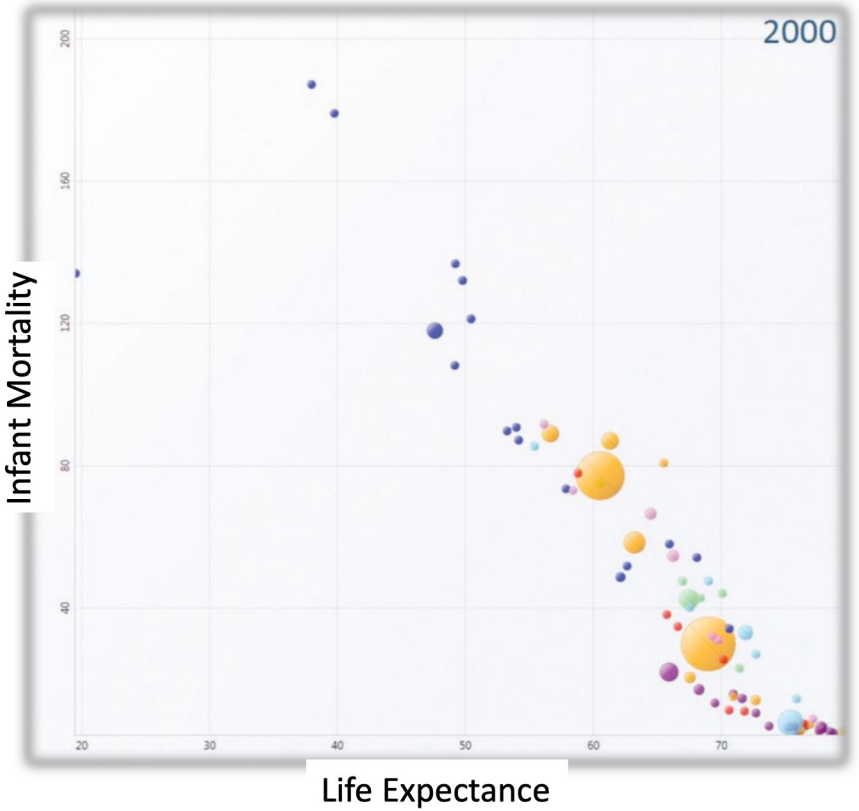
Points

Channels

Horizontal and vertical position on a common scale

Point's size, color, shape, etc.

Useful for a broad range of cases: trends, outliers, distribution, correlation, clusters, etc.



Networks - Node/Link Diagram

Marks

Nodes represents items (points)

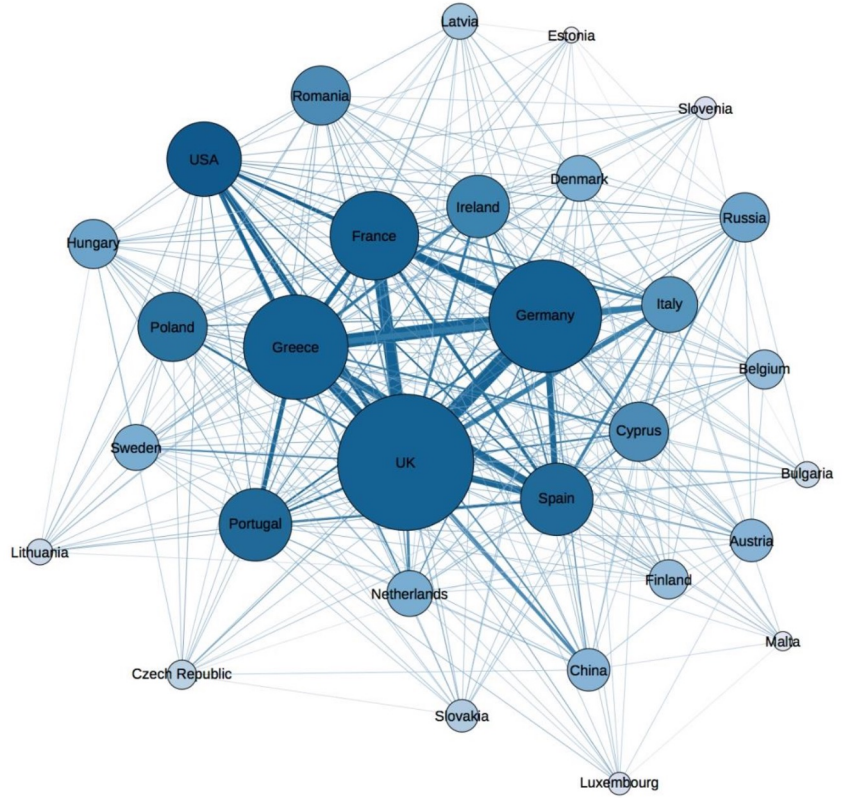
Links represents relationships between the items (lines)

Channels

In this example:

Nodes use color saturation and size to express item attributes

Links use thickness and color saturation to express relational attributes (aka weights)



Networks - Matrix

Marks

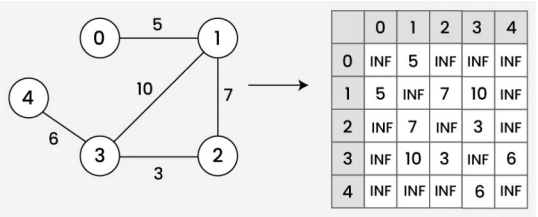
Nodes (items) are represented as rows *and* columns of a matrix

Links (relationships) are represented as cells in the matrix

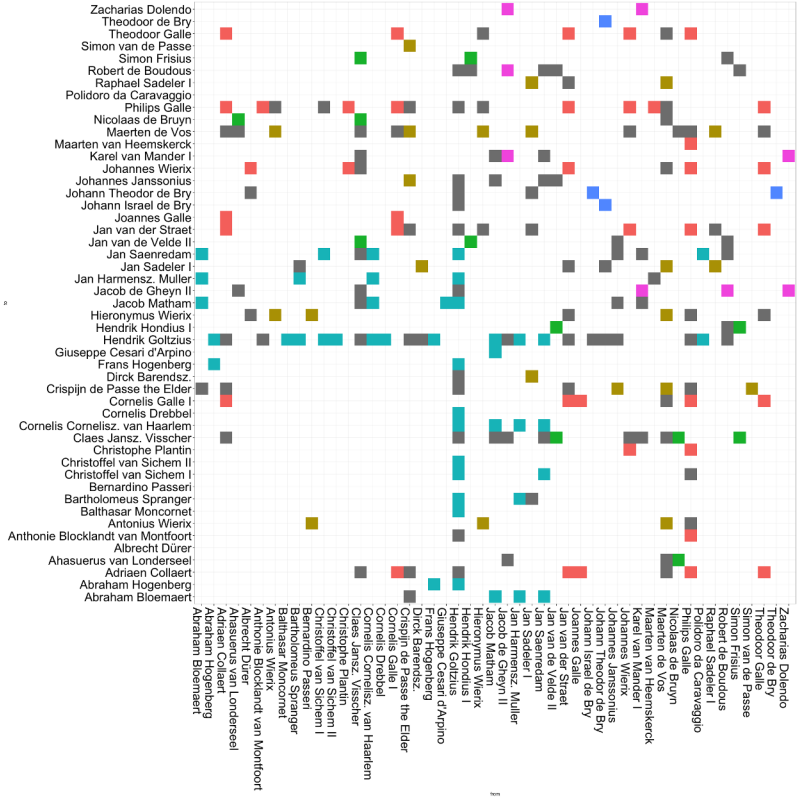
Channels

In this example:

Links use color to express its attribute (aka weight)



	0	1	2	3	4
0	INF	5	INF	INF	INF
1	5	INF	7	10	INF
2	INF	7	INF	3	INF
3	INF	10	3	INF	6
4	INF	INF	INF	6	INF



Trees - Node/Link Diagram

Marks

Nodes represents items

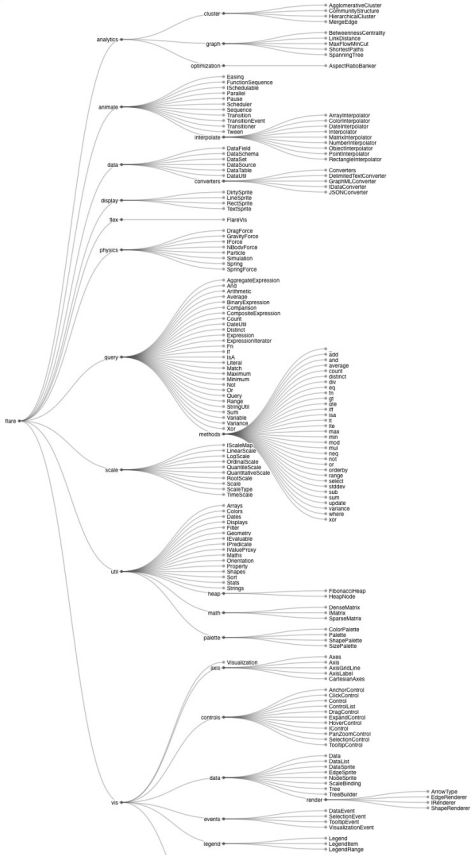
Link represents hierarchical relationships

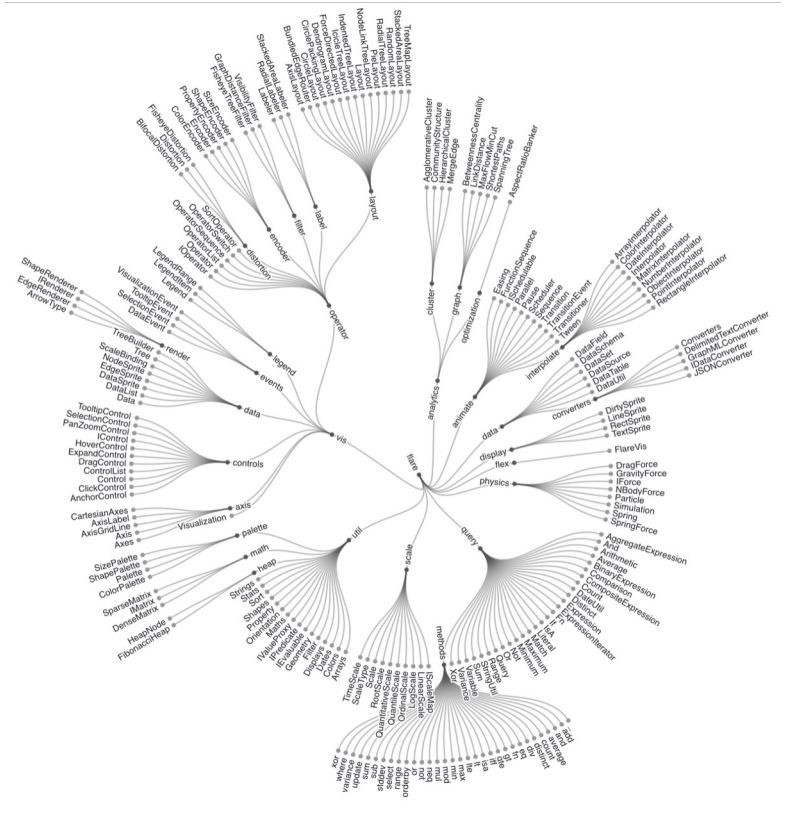
Many different layouts

Left: Reingold-Tilford layout

Right: Radial layout

Difficult to lay out due to its combinatorially increasing complexity





Trees - Indentation

To alleviate complexity, branches can be closed

Marks

In this example:

Leaf nodes represents one type of items – Files

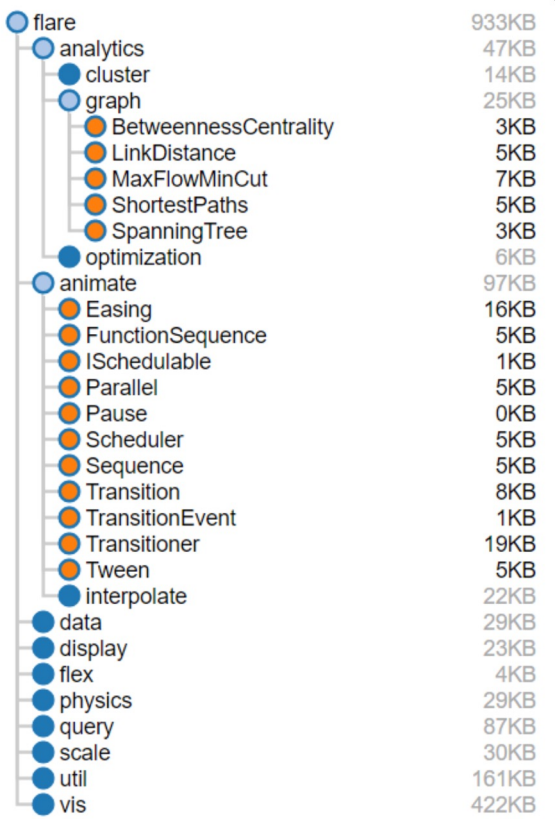
Branch nodes represents a different type of items – Folders

Sub-nodes in unopened branches are hidden

Channels

Branch nodes use color to express whether the folder is open or not

Leaf nodes are colored differently from the branch nodes for easy identification



Trees - Treemaps

Marks

Leaf nodes (items) are represented as Areas

Enclosure represents hierarchical relationships

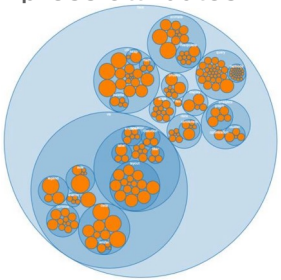
Recursively fill space based on a size metric for nodes

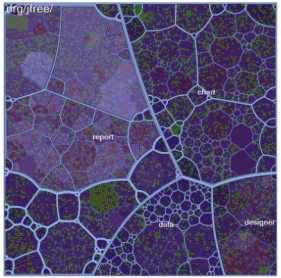
Channels

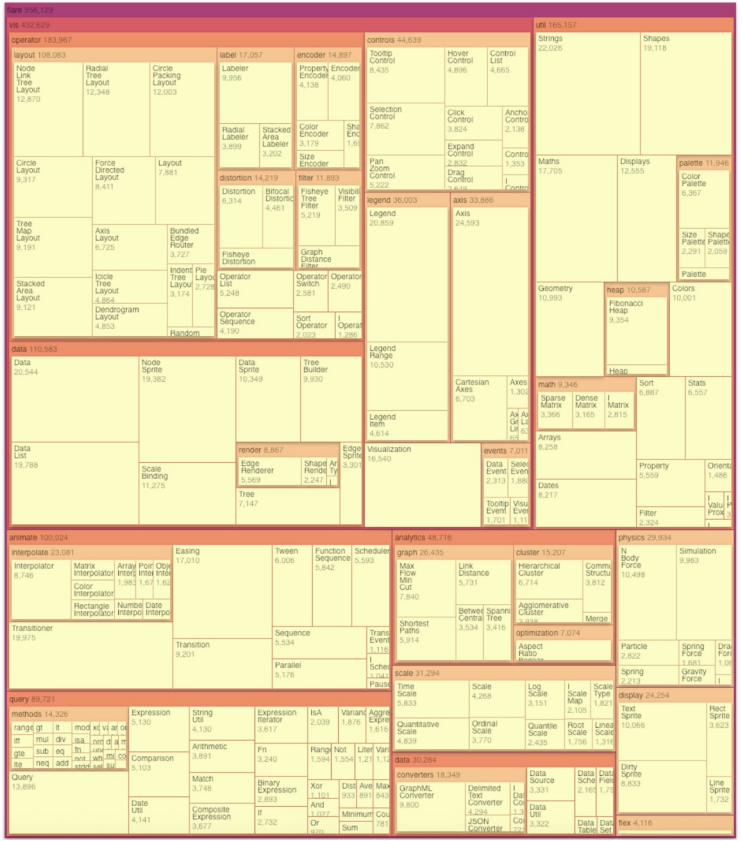
Node and enclosure areas also expresses an attribute (e.g. file/folder size)

Node color hue, saturation, etc can express attributes

Often uses rectangles, but other shapes are possible, e.g., circles and Voronoi tesselations

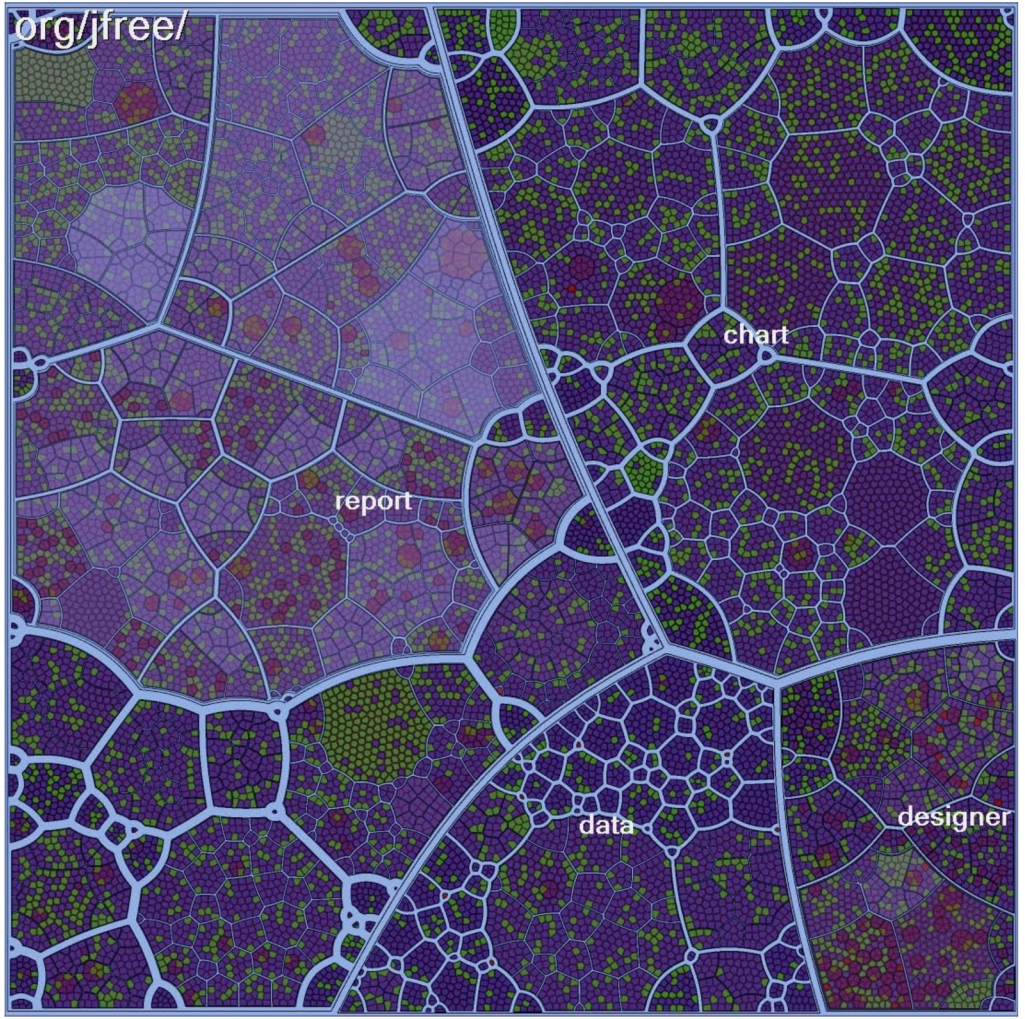




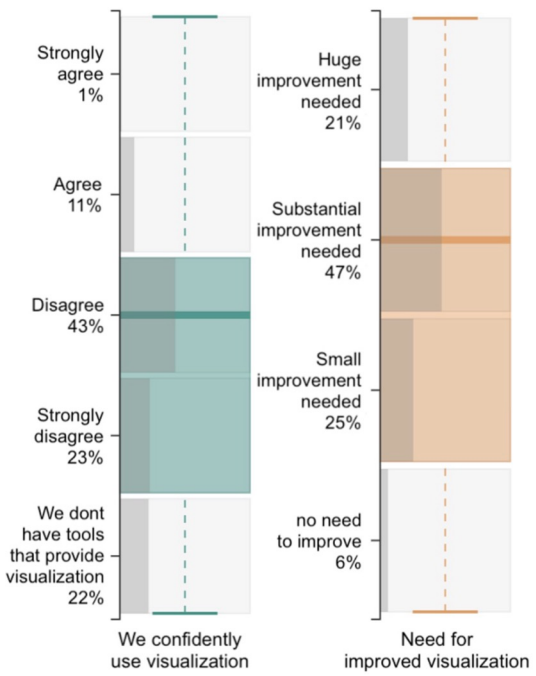


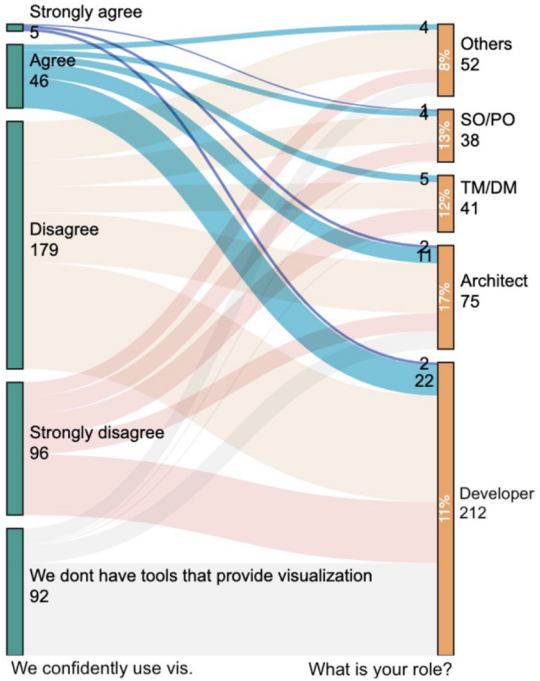
Part 2

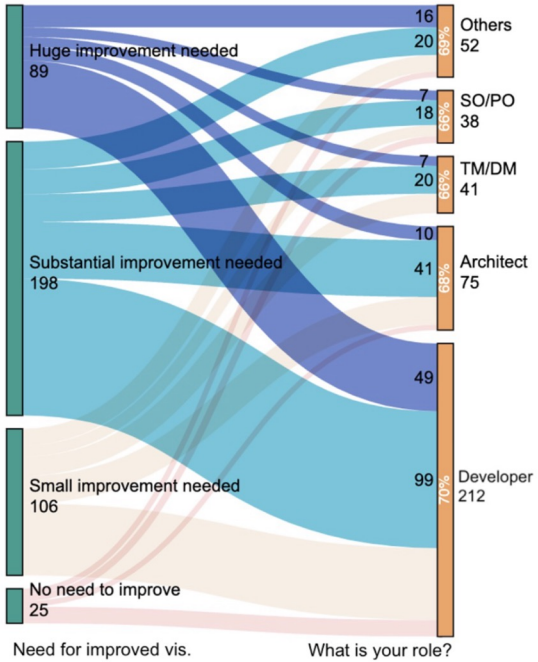
Software Visualization Techniques



Should you care about Software Visualization?



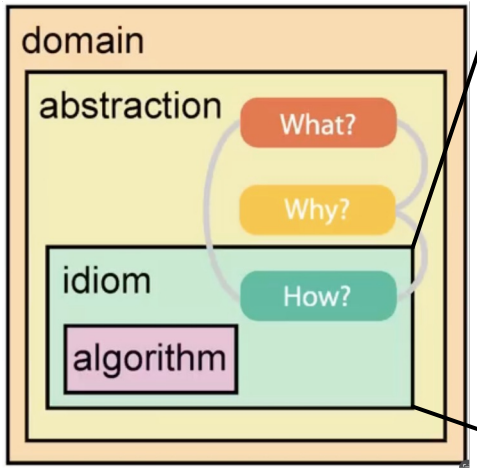


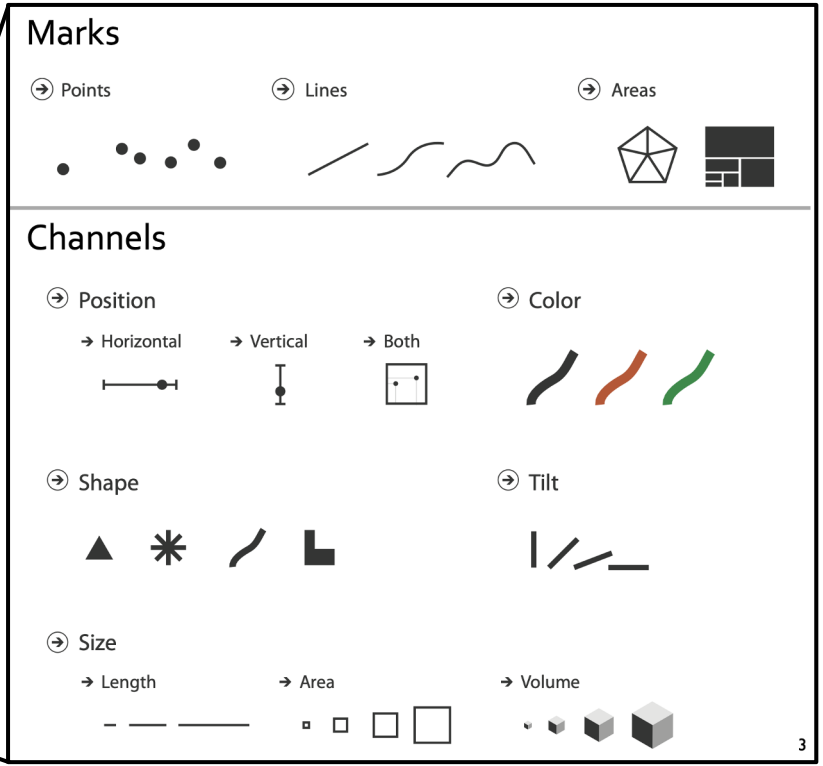


Irgens & Martini (2025): Visualization Usage in Technical Debt Management

Visualization Analysis Using Munzner's Model

- Domain Problem Characterization
- Data Abstraction (**What**)
- Task Abstraction (**Why**)
- Visual Encoding Idiom (**How**)
  - Marks
  - Channels





SeeSoft – Text Visualization

Abstract

Visualizes up to 50,000 lines of code simultaneously

Each line of code is mapped to a line

Color expresses an attribute of interest, e.g. time since change

Domain Problem Characterization

Traditional text-based representation make it difficult to summarize source code statistics at the line level

This is especially true for large systems with millions of lines of code

The lack of compact overviews limits developers' ability to comprehend change history, authorship, code activity, etc.



Eick et al. (1992): SeeSoft–A tool for visualizing line oriented software statistics

SeeSoft – Text Visualization

Data Abstraction

Items: Individual lines of source code

Attributes: Time since last edit, Total number of edits, Author of last edit, etc.

Task Abstraction

Summarize Features (of selected attribute)

Identify Outliers (f. inst. regions of high activity)

Discover Trends (certain file types have certain features)



SeeSoft – Text Visualization

Marks

Each line-of-code *(item)* is represented by a line

Each file is represented by an enclosing rectangular area, grouping lines of code (but does not have attributes in SeeSoft)

Channels

Each line-of-code's selected attribute value is expressed by color hue

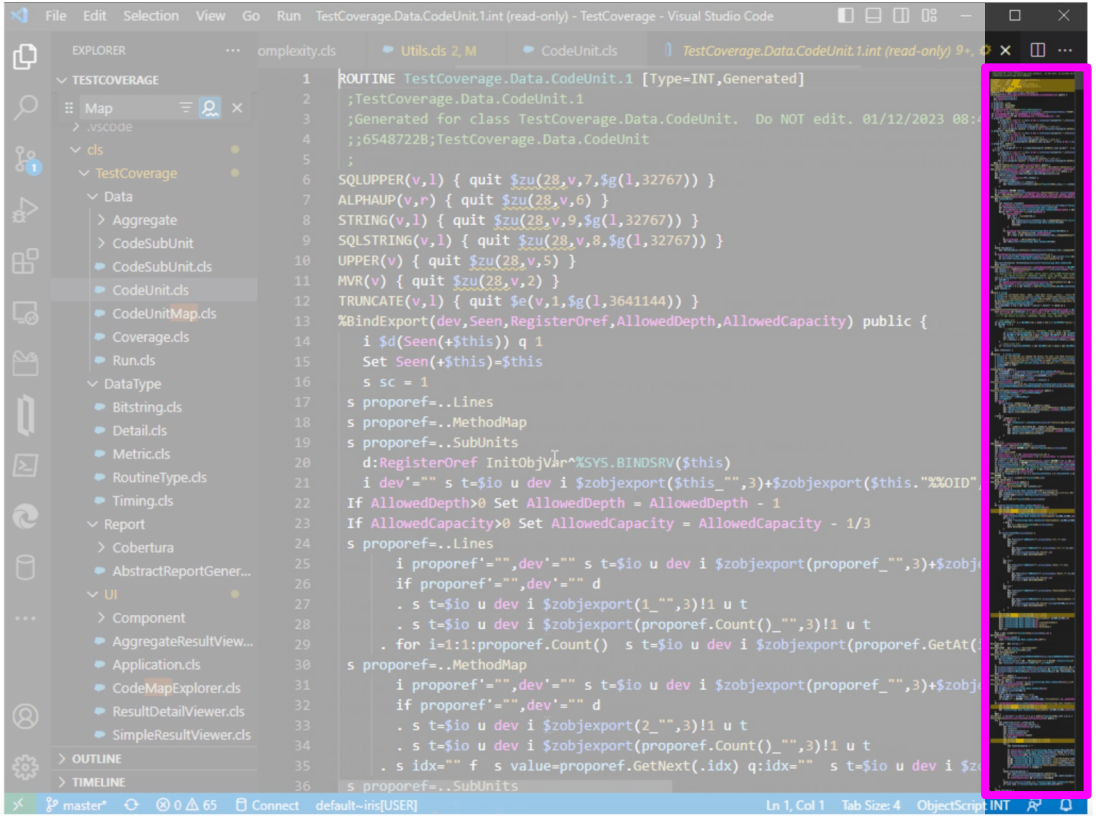


SeeSoft – Text Visualization

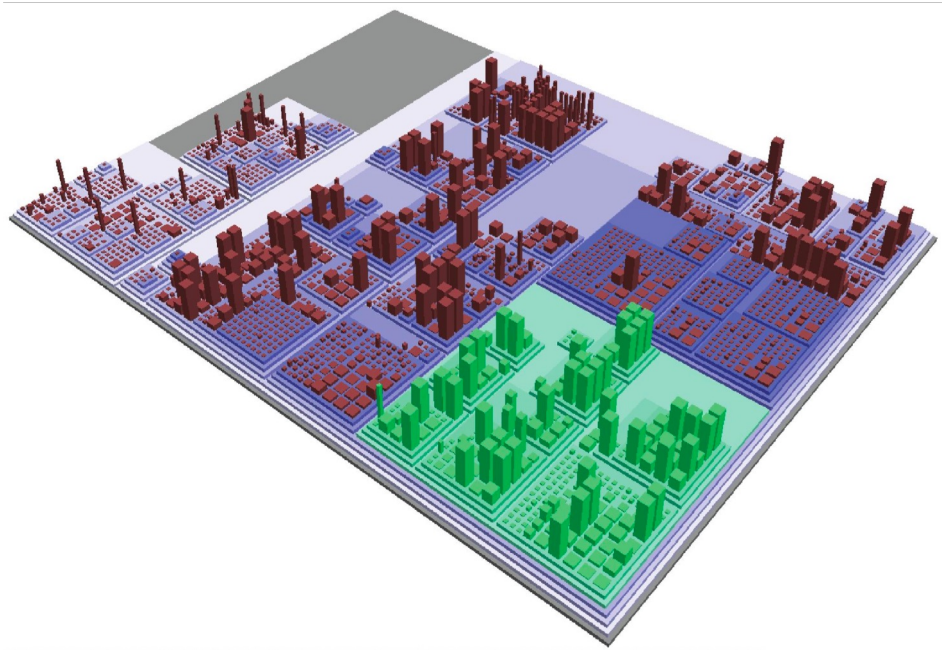
Most modern IDEs include a side panel that provides a visual overview of a file's contents

However, this panel is primarily used for navigation, not analysis

But, it does provide an overview, with color expressing syntax highlighting. So technically, it does provide summarization of features, identification of outliers, etc. with respect to the syntax



Software Cities



Abstract

Uses a 3D visualization approach based on a city metaphor

Classes are mapped to buildings, and packages are mapped to city districts

Visual properties (e.g., building height, base size, layout) are driven by source code metrics

The city metaphor provides a sense of locality to support program comprehension

Lanza et al. (2007): Visualizing Software Systems as Cities

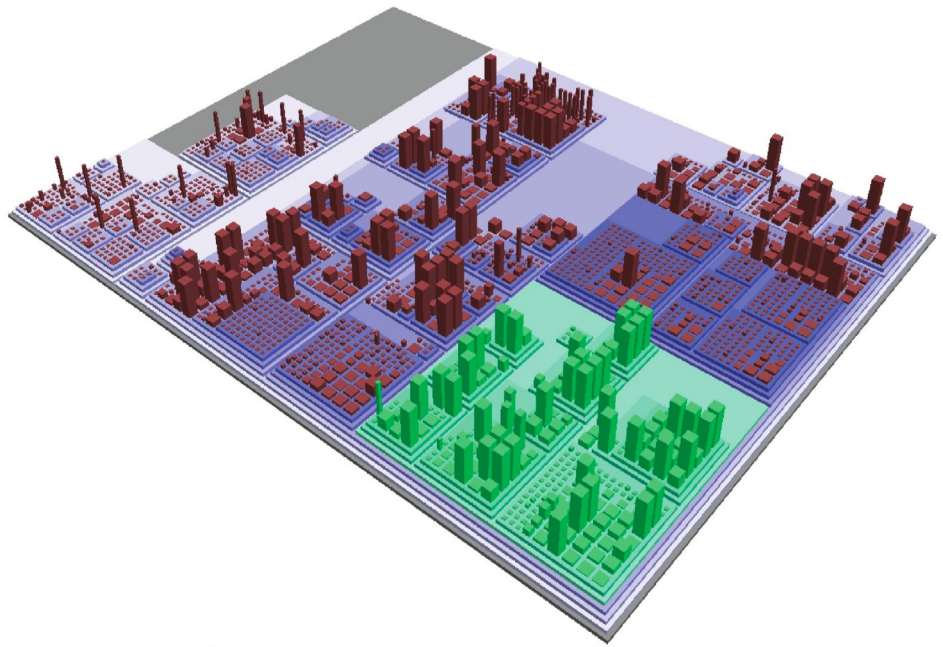
Software Cities

Domain Problem Characterization

Developers struggle to understand the structure of large software systems

Traditional 2D visualizations do not provide a sense of immersion or locality

3D visual metaphors (like cities) offer potential for cognitive enhancement, leveraging human spatial reasoning to improve comprehension



Software Cities

Data Abstraction

Items: Classes

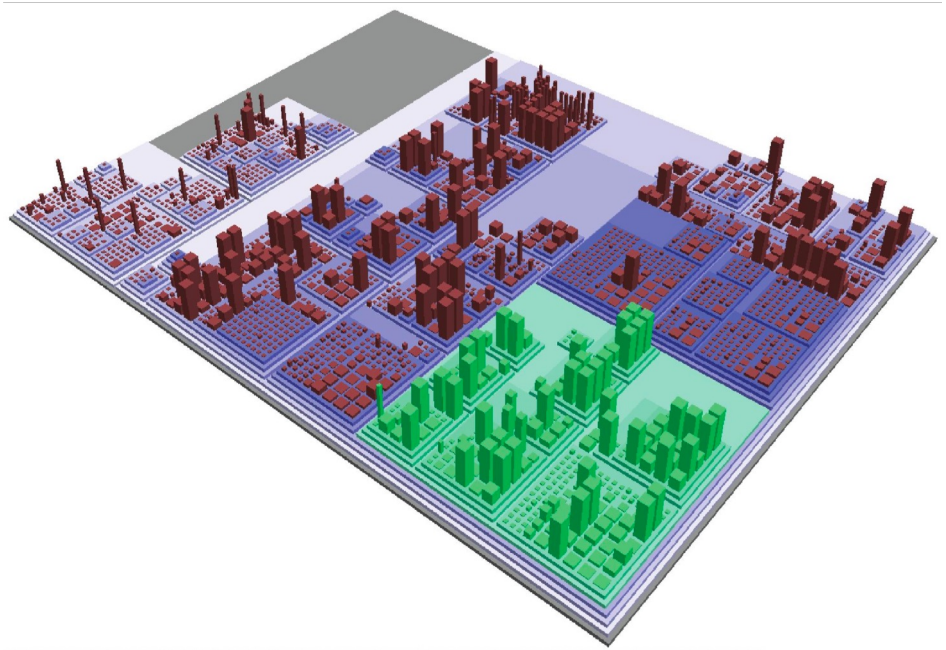
Attributes: Number of methods, LOC, Access level (public/private), Class type, Package membership

Task Abstraction

Summarize and Explore Features (of architecture)

Compare Features (between packages)

Discover Outliers (like large/complex classes)



Software Cities

Marks

Each class is represented by a box (3D building)

Each package is represented by an area/plateau/district

Basically a treemap, with an added third spatial dimension

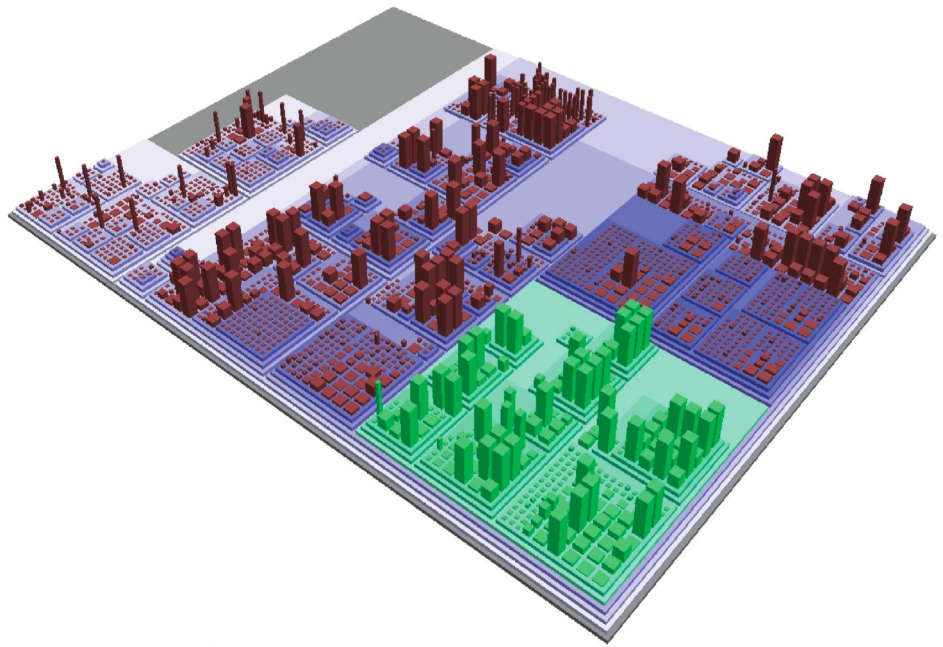
Channels

Number of methods is expressed by height (on a common scale)

LOC is expressed by box base area

Access level/class type is expressed by color

Package membership is expressed by grouping

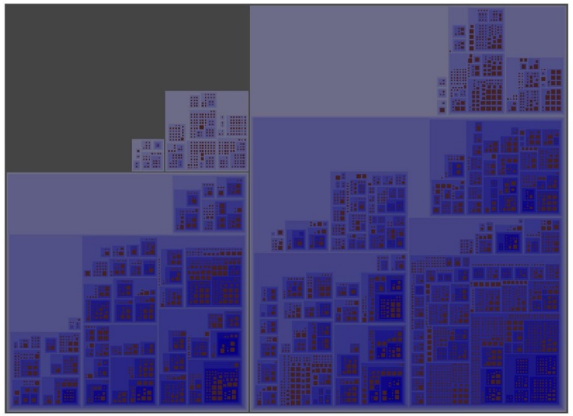


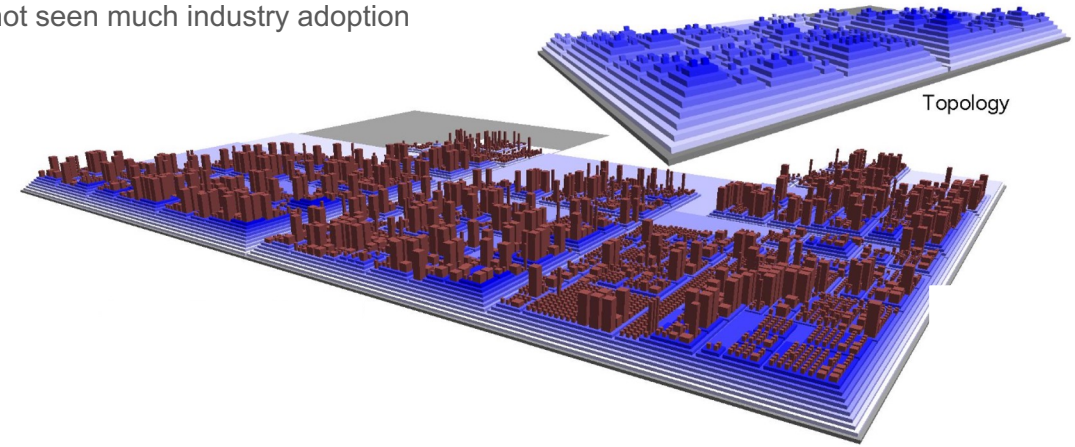
Software Cities

In general, 3D is discouraged in information visualization design due to:

- Occlusion of background elements by foreground elements
- Perspective distortion, which impairs comparison across distances
- Increased complexity in programming and interaction design
- Much higher computational cost for the user

However, the argument for cognitive enhancement is interesting. As a result, Software Cities continue to receive attention in research, even though they have not seen much industry adoption





CodeScene – Circular Treemaps

Abstract

Uses a circular treemap to visualize program structure – Files as innermost circles, and folders as enclosing circles

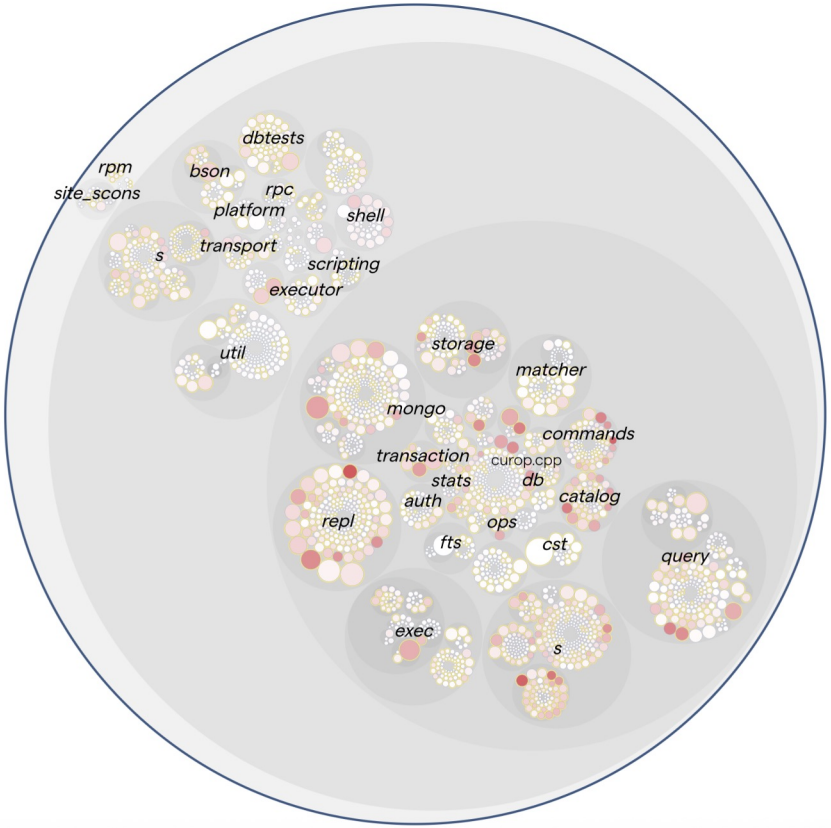
Main use case is to detect technical debt and guide refactoring prioritization

Domain Problem Characterization

Large codebases accumulate technical debt over time, but not all debt is equally important

Refactoring is expensive, so efforts must be prioritized

Developers need to understand the technical debt in the system to better decide where to put their efforts



Thornhill (2018): Software Design X-Rays

CodeScene – Circular Treemaps

Data Abstraction

Items: Files

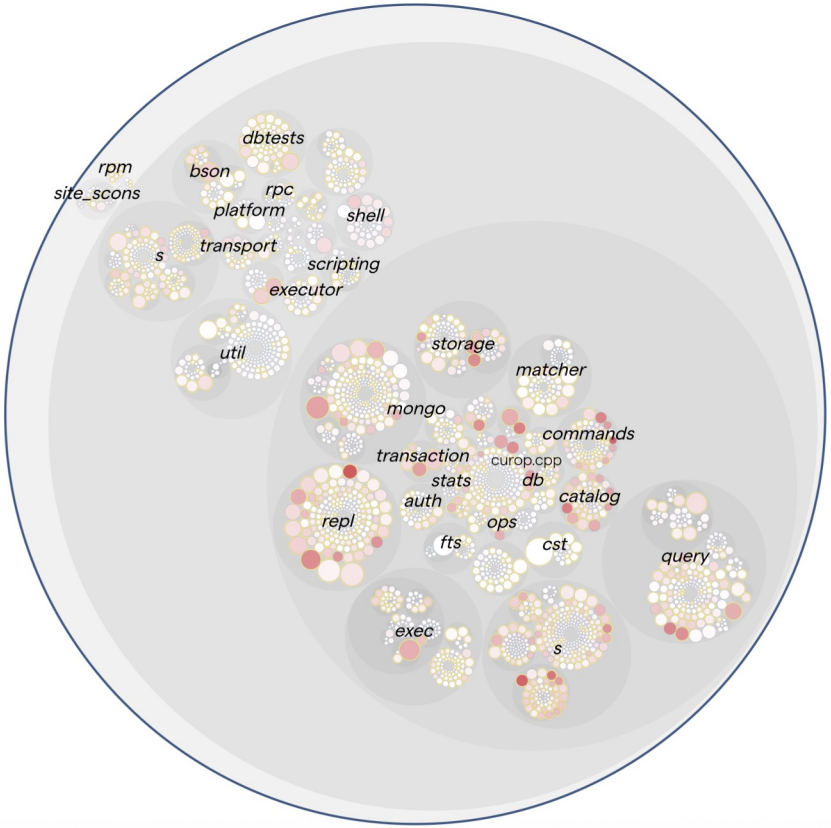
Attributes: File activity, Code health score, Code smells, Authors, Programming language, etc.

Task Abstraction

Identify Outliers (files that are changed frequently and has a low code health score)

Summarize Outliers (to help planning and proritization of refactoring)

Summarize Features (gain overview of techical debt in the system)



CodeScene – Circular Treemaps

Marks

Each file is represented by a circle

Files are contained in folders, represented by enclosing circles

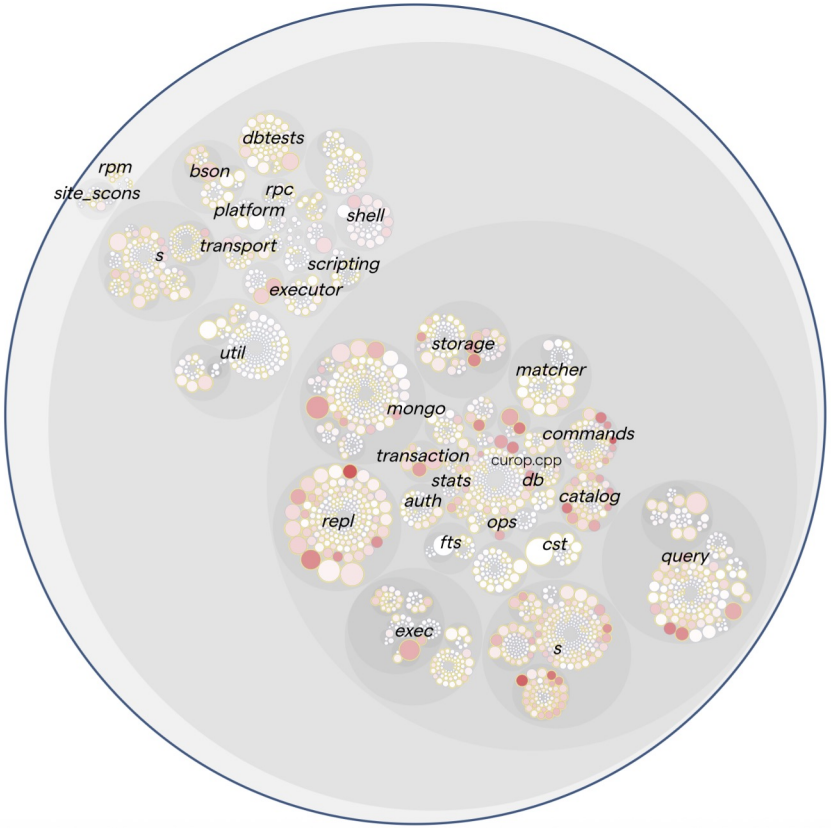
This is called a circular treemap

Channels

File size is expressed by circle area/size

A selected attribute is expressed by color

Folder hierarchy is expressed by grouping



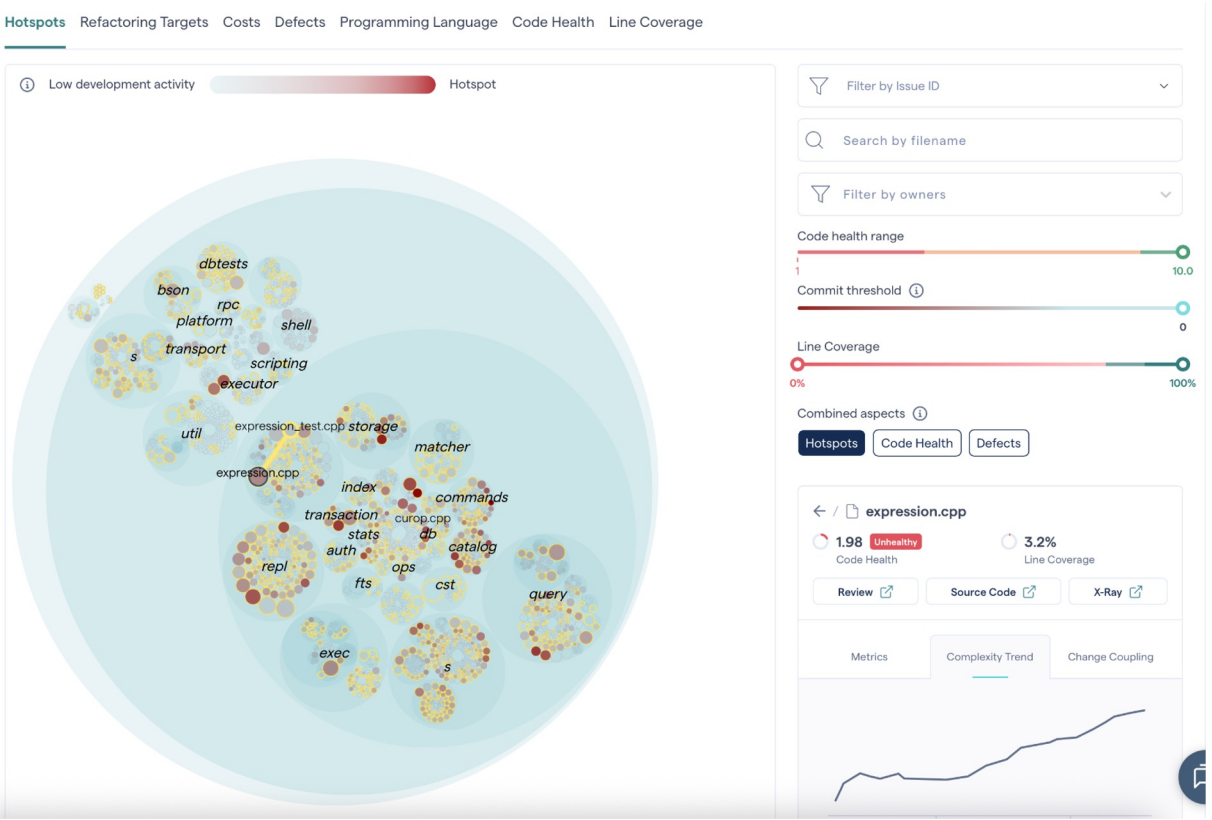
CodeScene – Circular Treemaps

CodeScene is a successful software visualization tool

CodeScene is used by global companies across many countries, including Fortune 500 firms

The company raised €7.5 million in funding to scale its “software engineering intelligence” platform

I recommend reading “Software Design X-Rays” by Adam Thornhill if you find this interesting



Voronoi Treemaps

Abstract

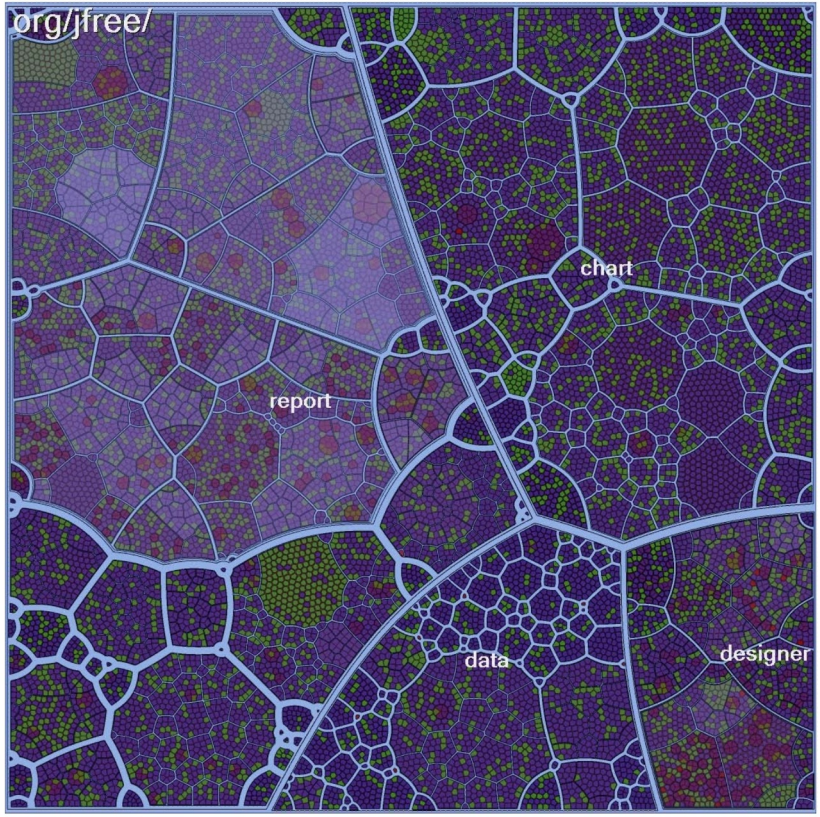
Replaces traditional rectangular treemap layouts with Voronoi polygons to improve visual clarity and boundary separation

Aims to enhance comprehension of large, complex software systems through better shape differentiation

Domain Problem Characterization

Traditional treemaps can lead to uniform and repetitive shapes, ambiguous borders between nested items, and a lack of clear spatial grouping, especially in deeply nested hierarchies

These limitations hinder quick shape recognition, visual clarity, and effective comparison of structural and metric-based properties in software systems



Balzer et al. (2005): Voronoi Treemaps for the Visualization of Software Metrics

Voronoi Treemaps

Data Abstraction

The paper is a technique paper, and does not specify data type

Items: Software entity (e.g., File, Class, Function, Package)

Attributes: Entity's metrics (e.g., LOC, complexity, methods)

Task Abstraction

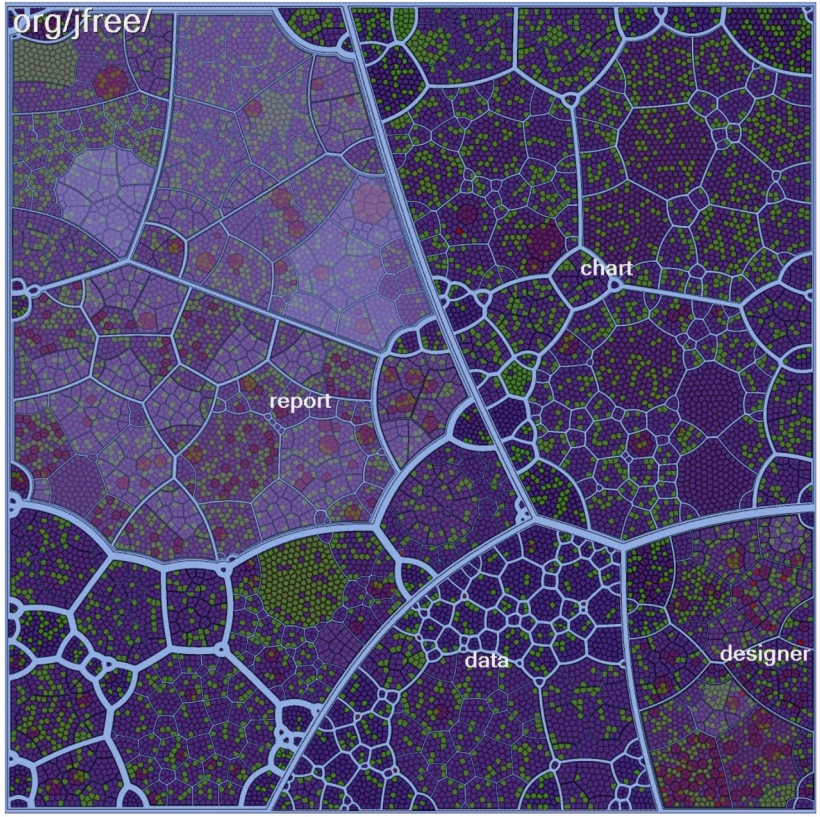
This technique is proposed to be better at:

Summarize and Identify Features (of architecture)

But of course, typical treemap tasks are also available, such as:

Identify Outliers (of selected entity metrics)

Compare Features (between hierarchical branches)



Voronoi Treemaps

Marks

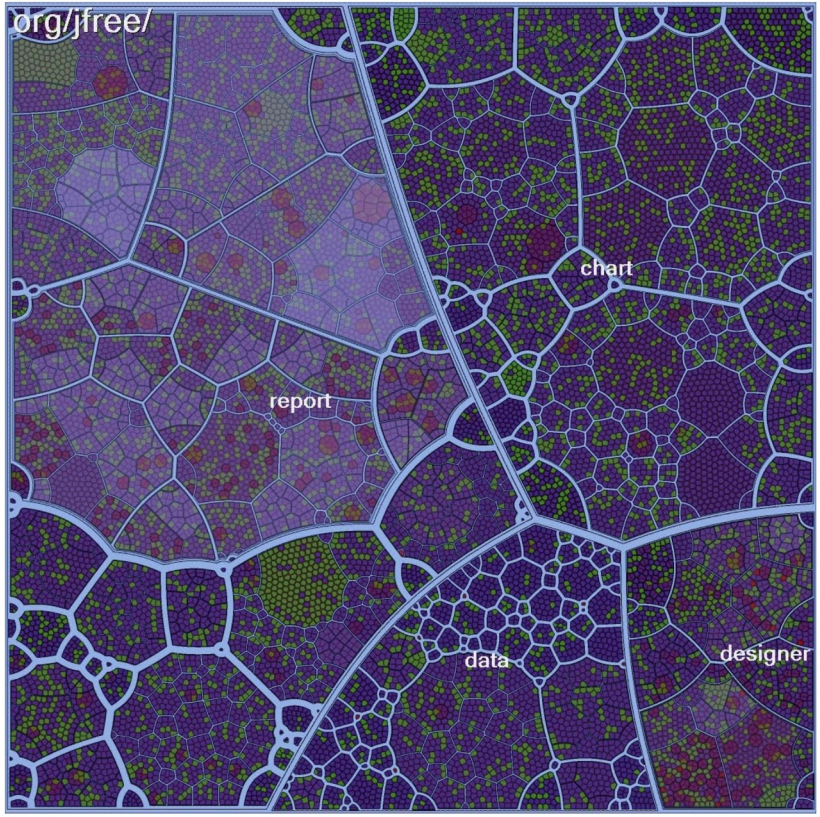
Each software entity is represented by a polygonal area

Nested hierarchies are represented by Nested polygon regions

Channels

Typical size metrics can be expressed by polygon size (e.g., LOC, number of methods, file size in bytes)

Other metrics can be expressed by polygon color hue and saturation



Treemaps Are Important In Software Visualization



https://treevis.net/

E-Quality – Dependency Graphs

Abstract

Uses a directed class dependency graph (network)

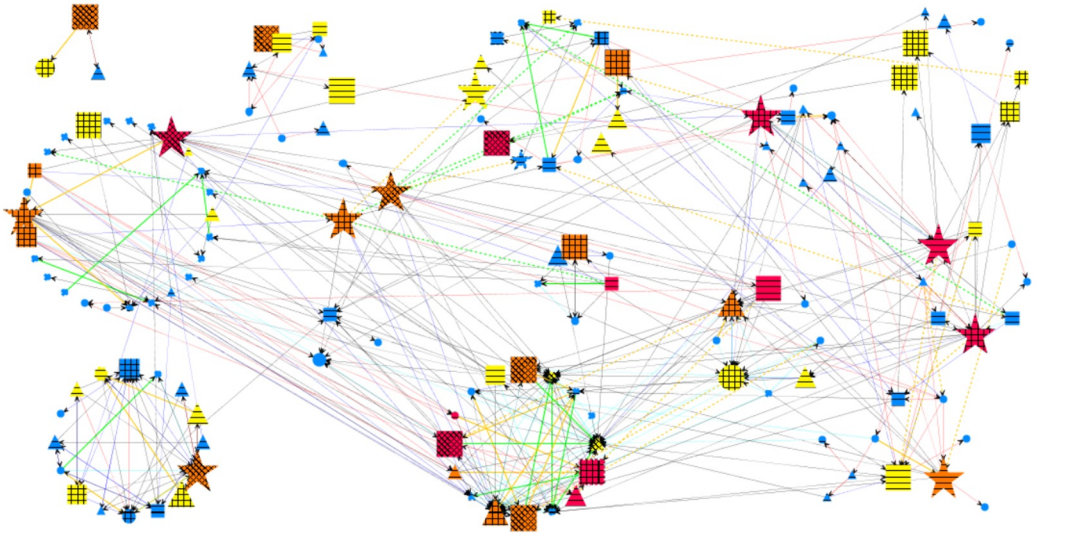
Maps quality metrics (e.g., coupling, cohesion, complexity, size) to node properties like shape, color, texture, and size

Maps relationships (e.g., type and strength) to link properties, like link direction and color

Domain Problem Characterization

Developers need help detecting design flaws, structural issues and low-quality classes

Quality attributes are hard to interpret in raw form, especially relationship-based metrics like coupling



Erdemir et al. (2005): E-Quality: A Graph Based Object Oriented Software Quality Visualization Tool

E-Quality – Dependency Graphs

Data Abstraction

Items: Classes

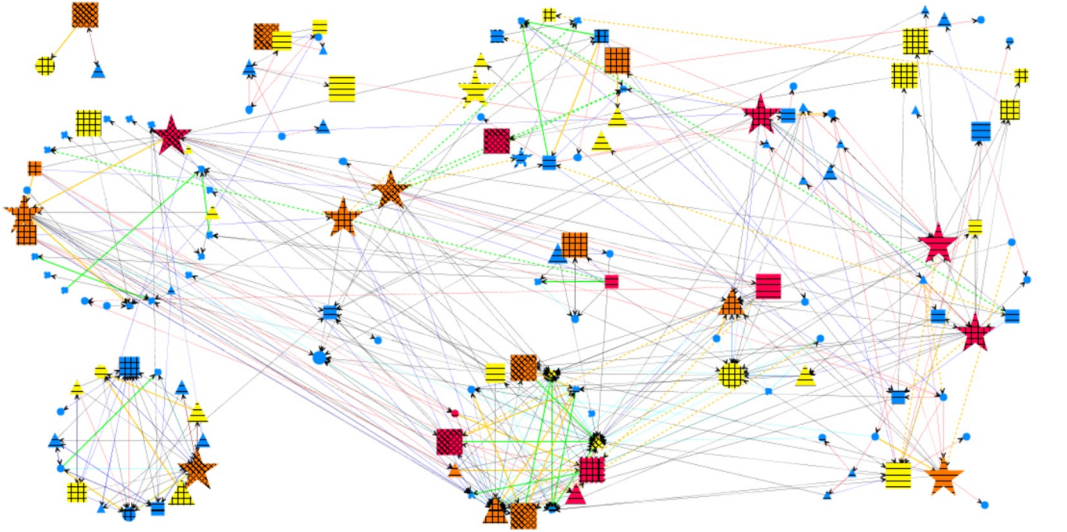
Attributes: LOC, cohesion, coupling, complexity, etc.

Task Abstraction

Summarize and Explore Topology (of object-oriented architecture)

Discover Paths (e.g. problematic or unknown dependencies)

Identify Outliers (e.g. complex classes, god classes)



E-Quality – Dependency Graphs

Marks

Classes are represented by nodes

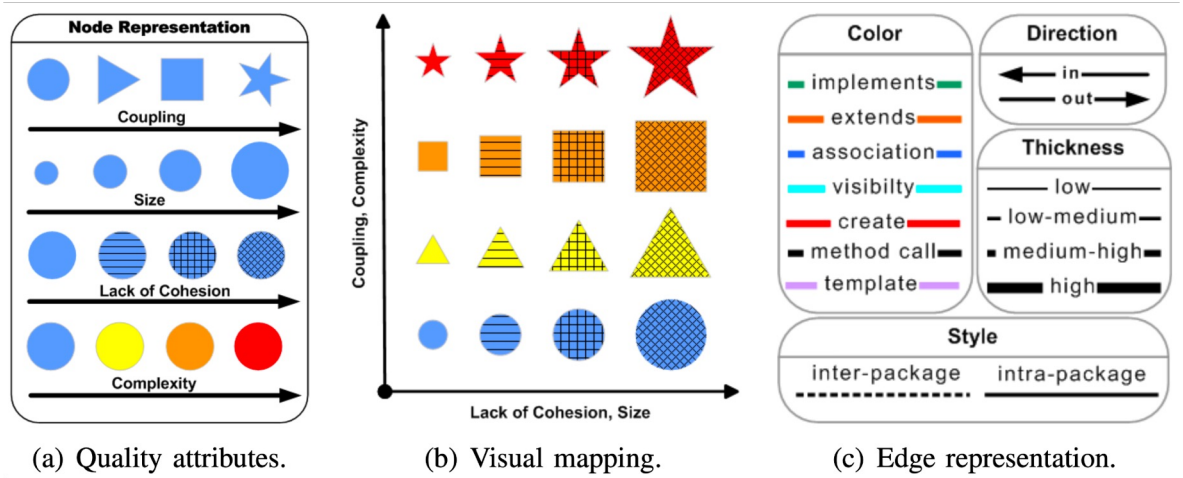
Relationships are represented by links (edges)

Channels

Different class quality attributes are expressed by the node's shape, size, texture and color.

Different relationships are expressed by the link's color, thickness and style.

An arrow expresses the directionality of the relationship



(a) Quality attributes. (b) Visual mapping. (c) Edge representation.

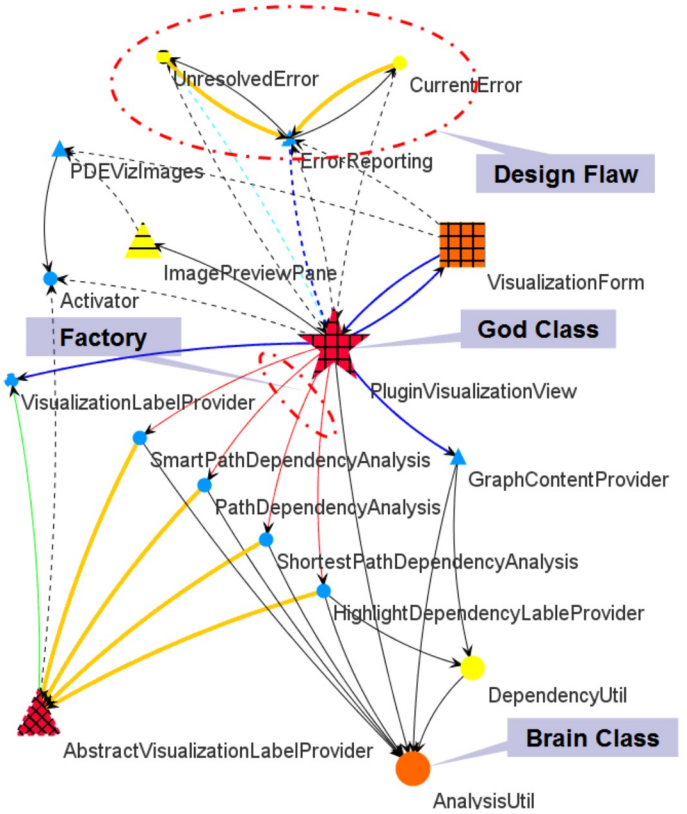
E-Quality – Dependency Graphs

This is, in my opinion, a really good example of effective channel use in a network visualization

But is the technique itself valid?

Object-Oriented Programming is less dominant today

The chosen metrics (e.g., cohesion, coupling, complexity) are are not considered indicators of quality by many developers



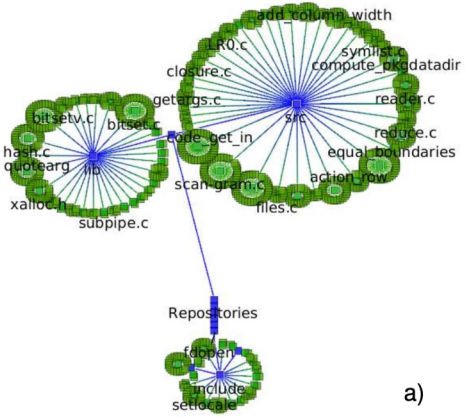
Dependency Graph Complexity

Dependency graphs can become incredibly complex

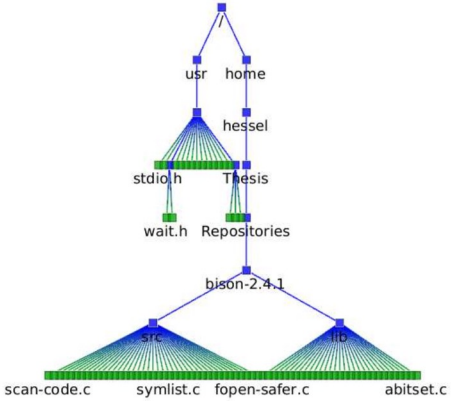
Top: File structure only (Trees)

a) Bubble tree
b) Rooted tree

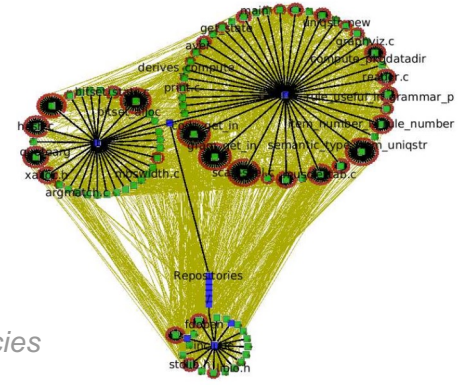
Bottom: Calls added (Networks)

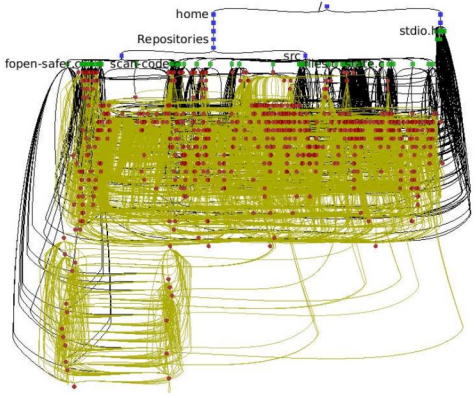


a)



b)





Telea et al. (2009) Extraction and Visualization of Call Dependencies for Large C/C++ Code Bases: A Comparative Study

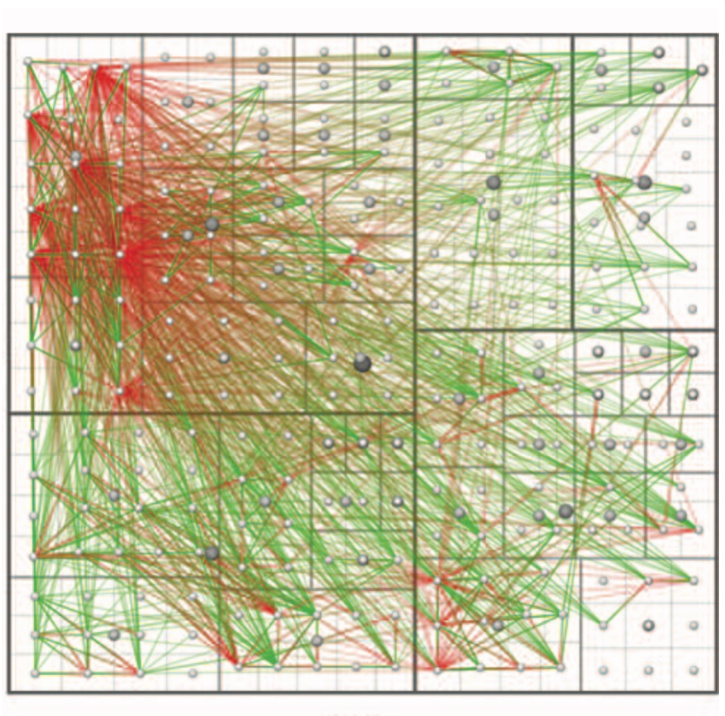
Dependency Graph Complexity

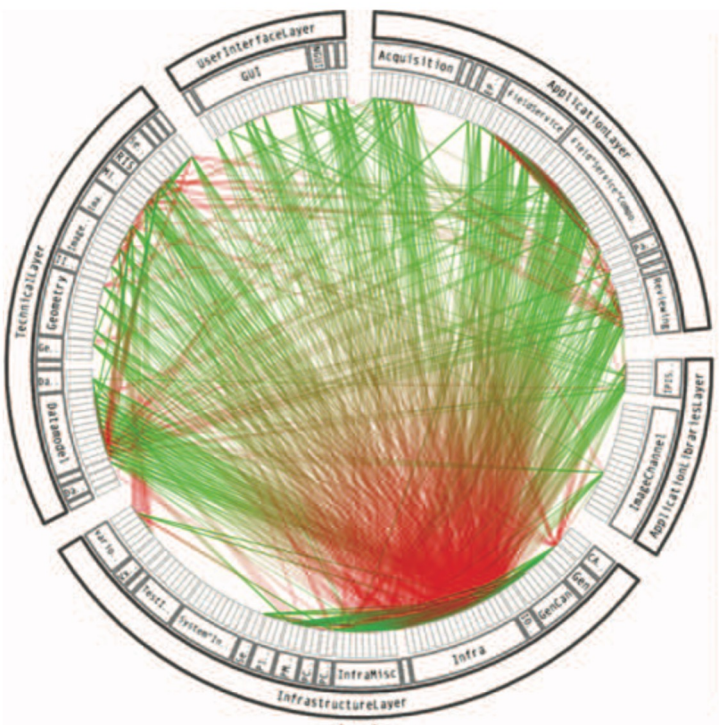
Treemaps with calls

Left: Rectangular treemap

Right: Radial treemap

Radial treemap is a bit better, but still hard to read





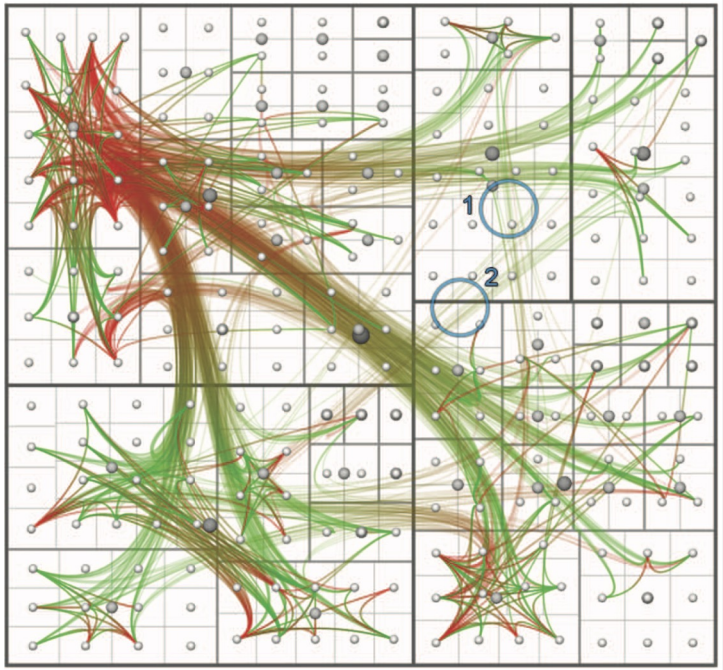
Holten (2006) Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data

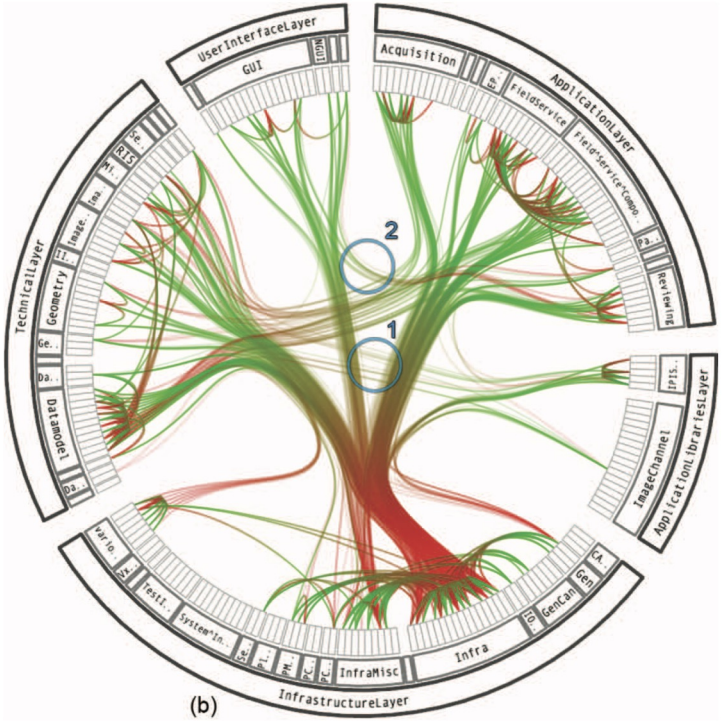
Dependency Graph Complexity

Edge bundling can help

Very important technique in visualization of big networks

Radial treemap with edge bundling is probably the most typical for software system dependencies





Holten (2006) Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data

CVSGrab – Software Evolution Visualization

Abstract

CVSGrab is a technique for exploring the history of large software projects

Visualizes version control history for all the files, using a line for each file

The horizontal axis is time, and a line starts when a file was created

Domain Problem Characterization

In order to better understand the current state of a system and predict how it might evolve, it is important to analyze its past

Large systems generate massive commit histories involving thousands of files and developers

Raw version control logs are difficult to analyze

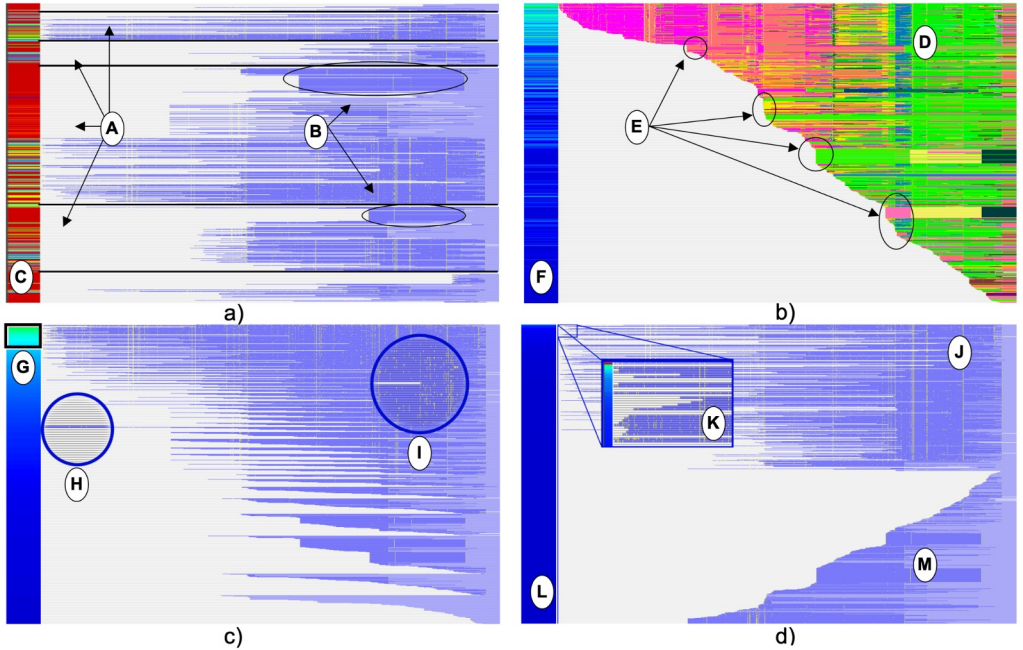


Figure 7: *Interactively built layouts of the VTK project using sort operations: (a) files sorted alphabetically, vertical metric shows cluster IDs, (b) files sorted by creation time, vertical metric shows activity, (c) files sorted by activity, vertical metric shows activity, (d) files sorted by similarity with respect to a reference file- vtkIntArray.cxx , vertical metric shows similarity.*

Voinea & Telea (2006): CVSgrab: Mining the History of Large Software Projects

CVSGrab – Software Evolution Visualization

Data Abstraction

Items: Files, commits

Attributes (File): List of commits

Attributes (Commit): Time of commit, author of commit, etc.

Task Abstraction

Explore and Discover Trends

Identify Outliers (e.g., files with a high number of commits)

Identify Features (e.g., short periods where a high amount of files was added)

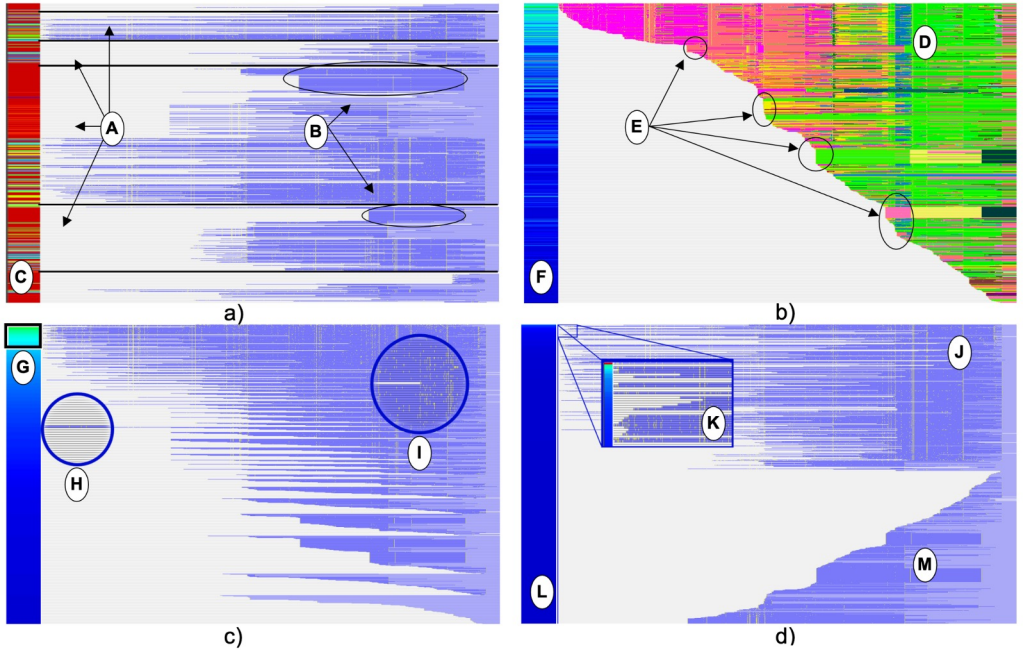


Figure 7: *Interactively built layouts of the VTK project using sort operations: (a) files sorted alphabetically, vertical metric shows cluster IDs, (b) files sorted by creation time, vertical metric shows activity, (c) files sorted by activity, vertical metric shows activity, (d) files sorted by similarity with respect to a reference file- vtkIntArray.cxx , vertical metric shows similarity.*

CVSGrab – Software Evolution Visualization

Marks

Files are represented by lines

Channels

Commit attributes are expressed by color changes along the lines

A bar on the left shows a single attribute for each file (e.g., activity – total number of commits over a certain timerange)

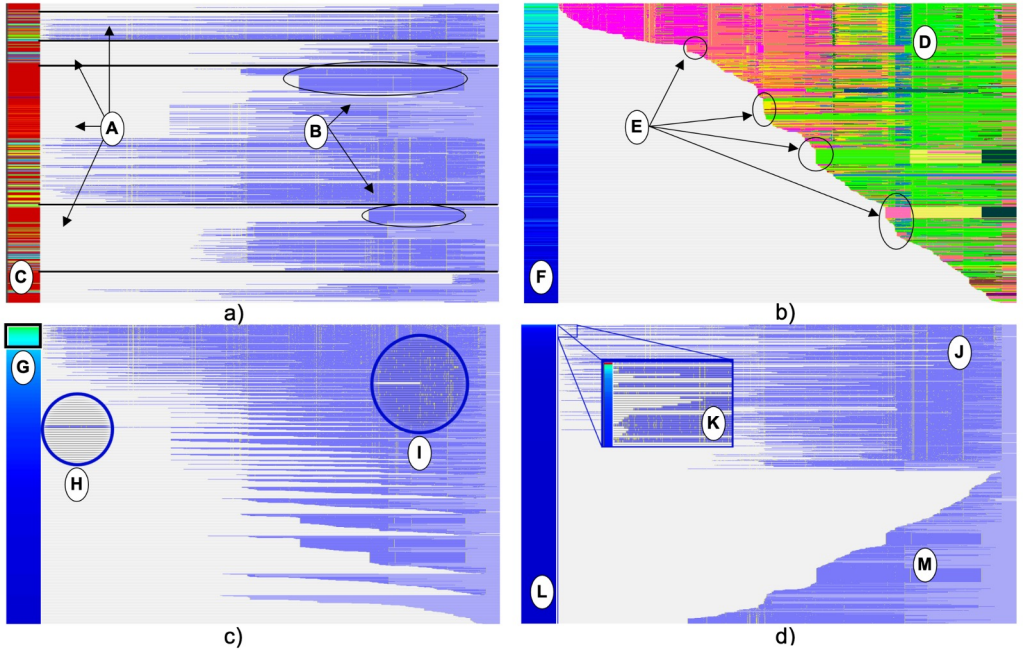


Figure 7: *Interactively built layouts of the VTK project using sort operations: (a) files sorted alphabetically, vertical metric shows cluster IDs, (b) files sorted by creation time, vertical metric shows activity, (c) files sorted by activity, vertical metric shows activity, (d) files sorted by similarity with respect to a reference file- vtkIntArray.cxx , vertical metric shows similarity.*

DebtQuest – Survey Visualization

Abstract

Visualizes survey data to detect technical debt management issues

Focuses on the organizational level rather than code level

Aims to reveal underlying causes of technical debt

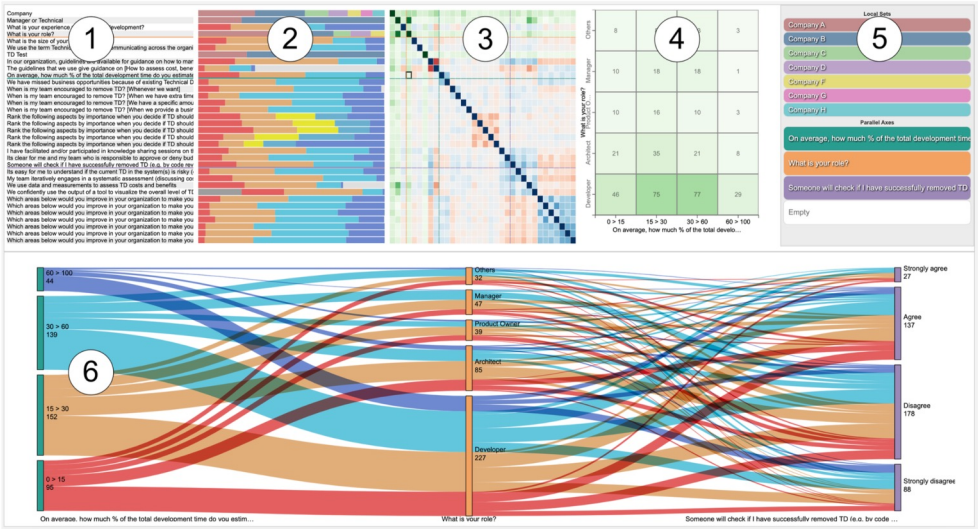
Builds on TD Pulse, a company-wide survey

Domain Problem Characterization

Managers lack visibility into how their teams manage technical debt

The TD Pulse survey was created to collect data on technical debt management, but it captures a lot of data

Traditional survey visualization (e.g., simple bar charts and box plots) are not well-suited for advanced analysis



(a) 1: Questionnaire Statements 2: Stacked Bar Charts 3: Association Matrix 4: Bivariate Heatmap 5: Menu 6: Parallel Sets

Irgens (2025): DebtQuest: Discover Technical Debt Management Issues with Survey Visualization

DebtQuest – Survey Visualization

Data Abstraction

Items: Survey participants

Attributes: Questionnaire answers (Likert scales and demographics – ordinal and categorical types)

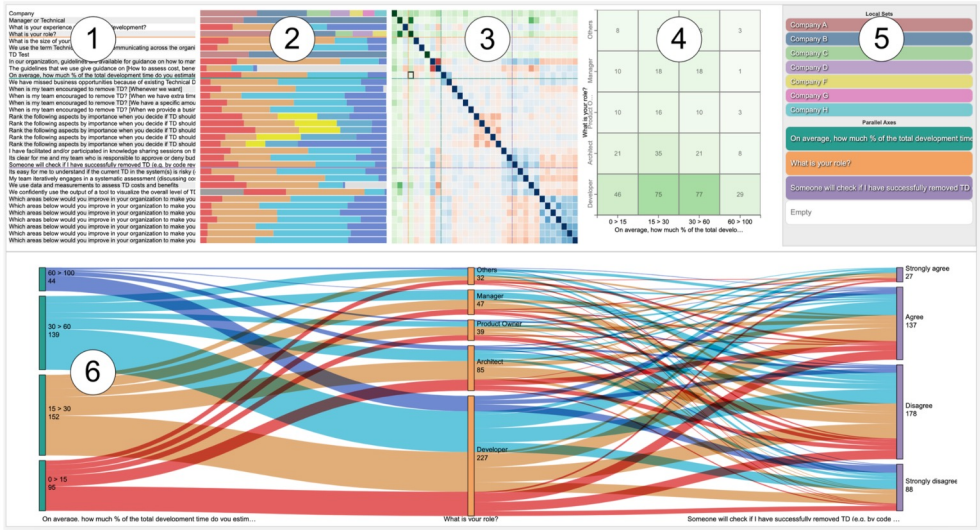
Task Abstraction

Summarize and Discover Distributions (of answers)

Identify Correlations (demographics associated with certain answers to certain questions)

Compare Distributions (especially misalignment between or within teams)

DebtQuest combines multiple techniques in a dashboard, so lets focus on the parallel sets visualization



(a) 1: Questionnaire Statements 2: Stacked Bar Charts 3: Association Matrix 4: Bivariate Heatmap 5: Menu 6: Parallel Sets

DebtQuest – Survey Visualization

Marks

A selection of questions are presented along the horizontal axis

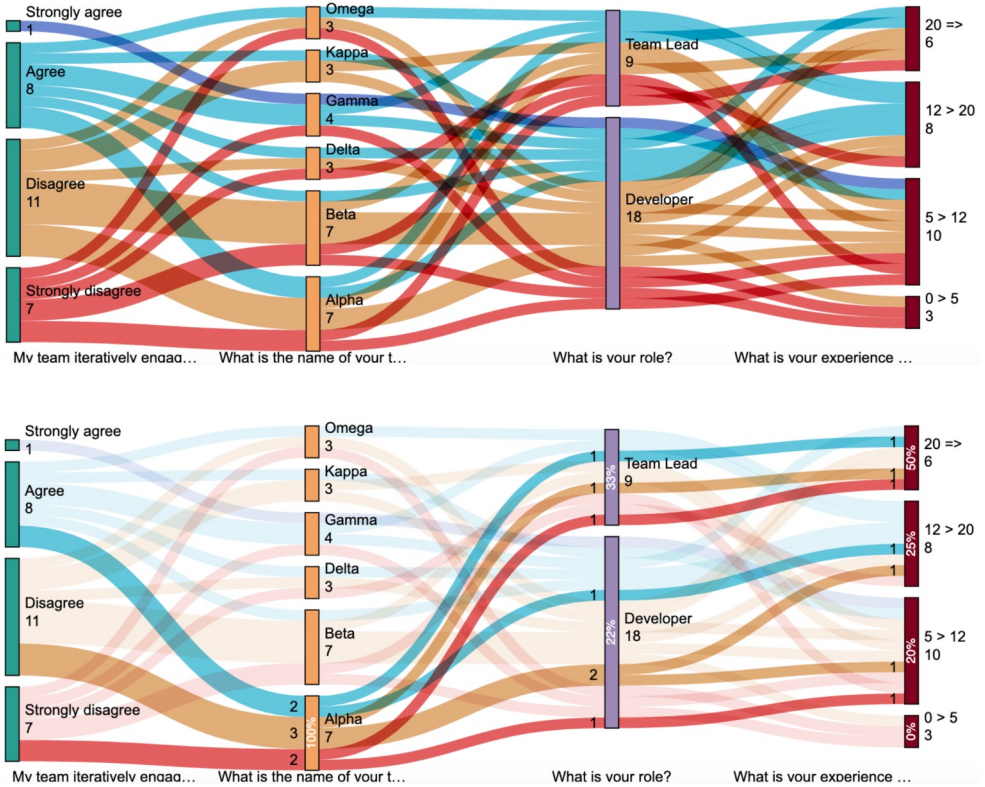
The answers are presented as rectangular nodes along the vertical axes

Combinations of answers are presented as links between two nodes (and paths through multiple nodes)

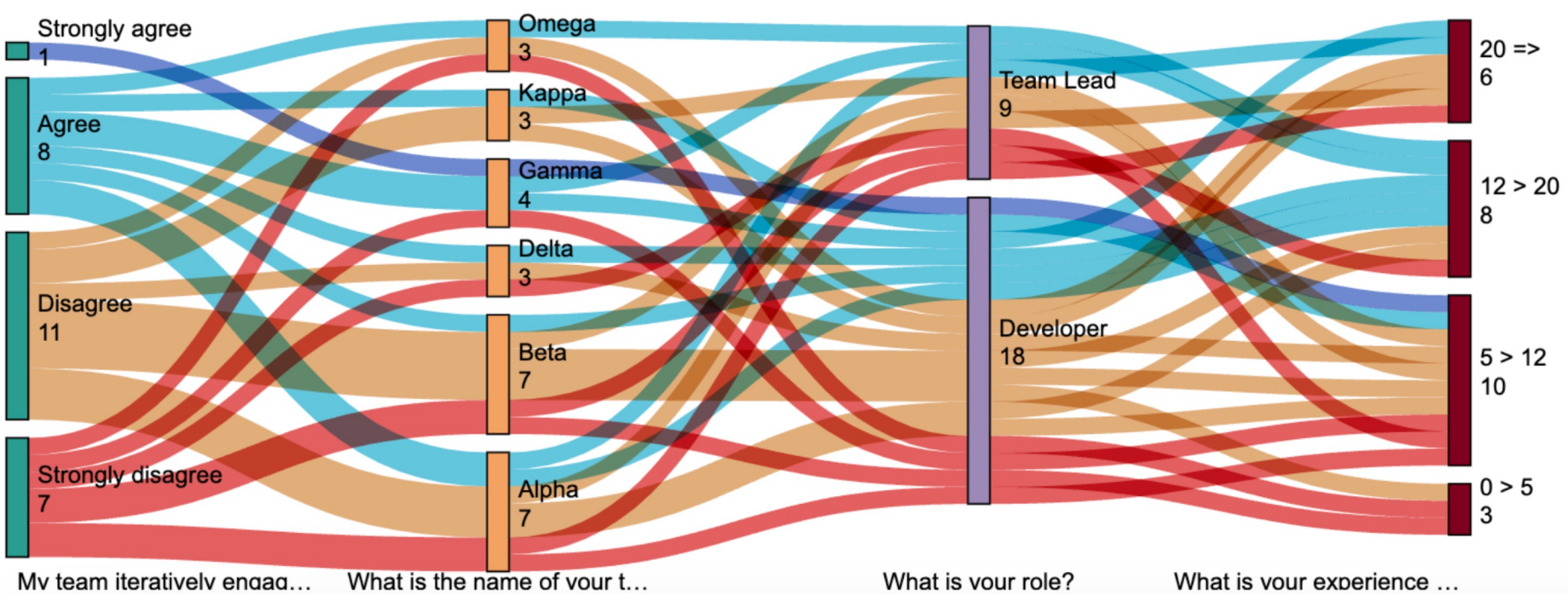
Channels

Answer to leftmost question is expressed by color

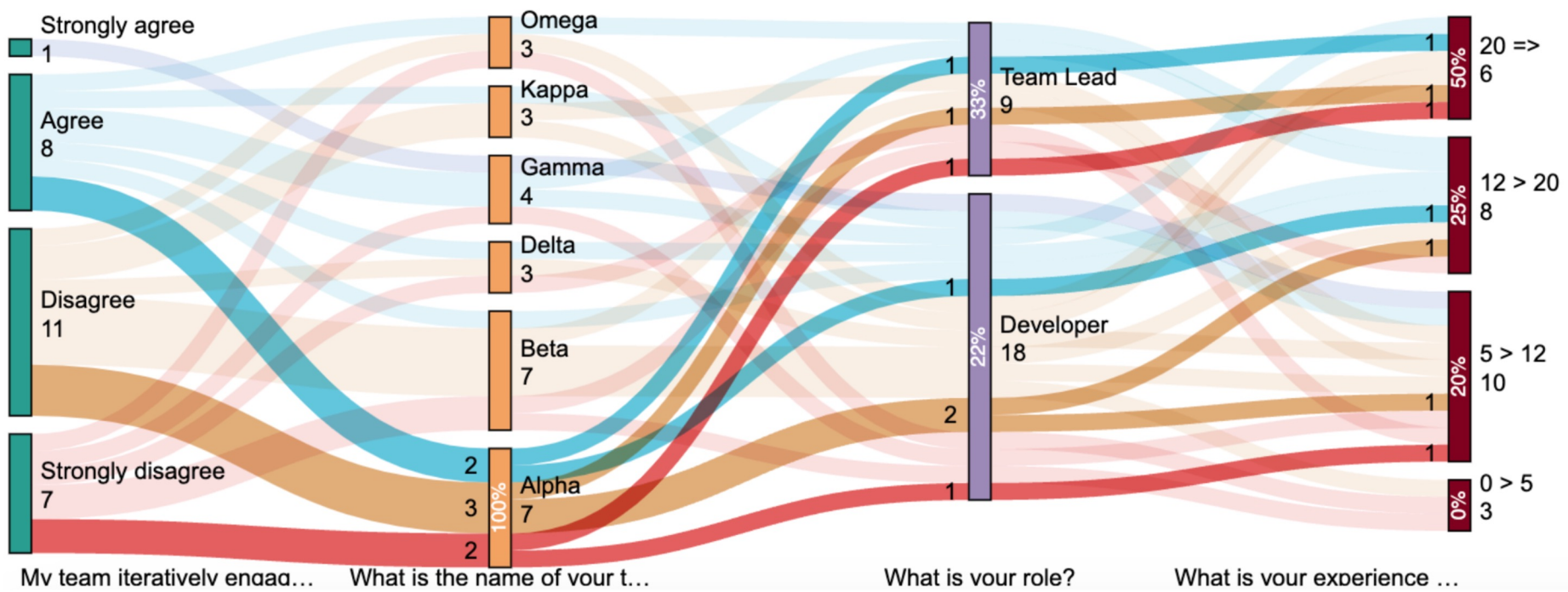
Focus is expressed by link opacity and extra information



DebtQuest – Survey Visualization



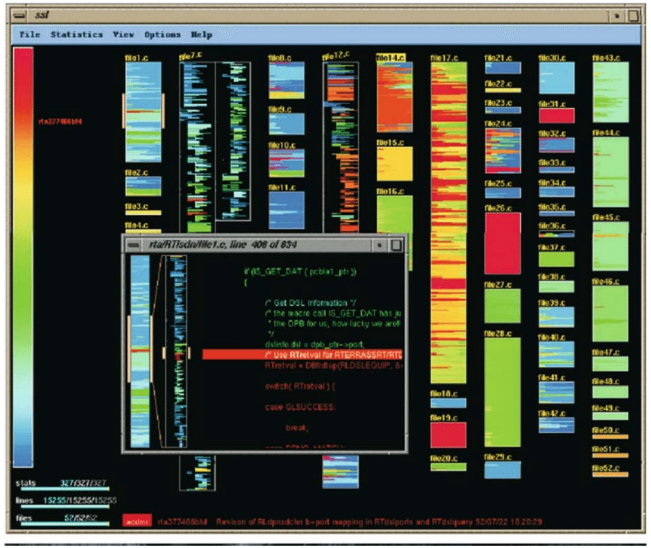
DebtQuest – Survey Visualization



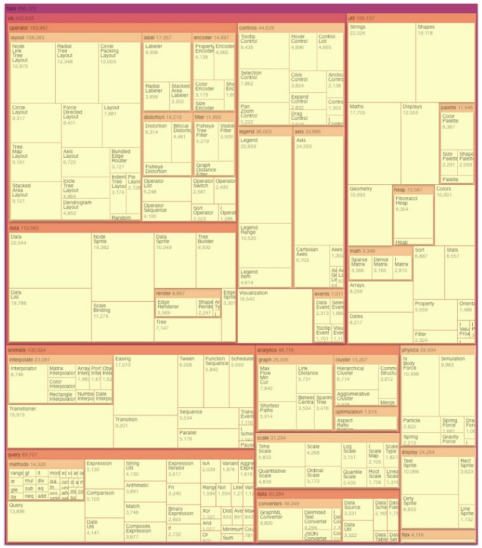
Main Techniques

Outside standard visualization techniques (bar charts, line charts, etc.) we find a few important techniques:

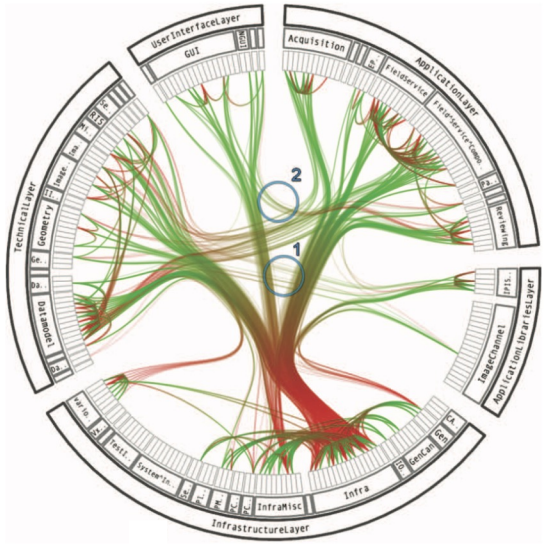
Text Vis (for code)



Treemaps (for hierarchies)



Networks (for dependencies)



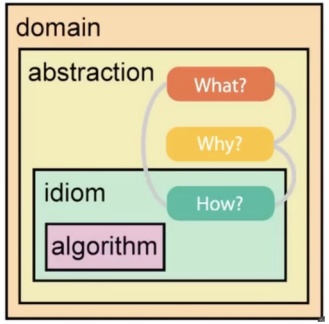
Important Parts To Remember

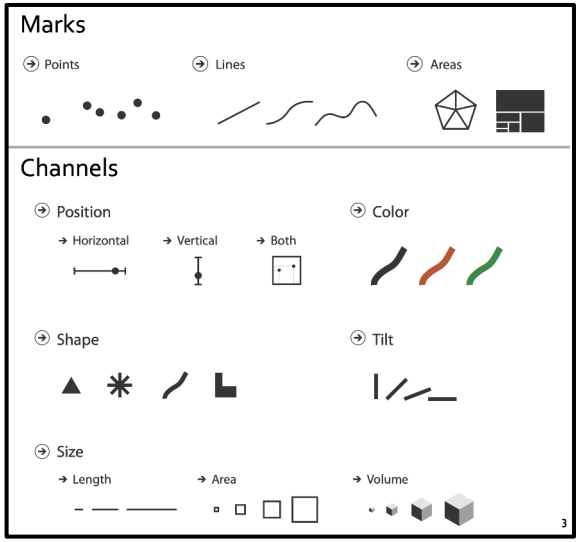
The most important concepts from this lecture are:

- **Problem-Driven Visualization Design**
- **Marks and Channels**

If you get any questions from this lecture on your exam, it will be on one of these two concepts

To understand these concepts, the examples are hopefully helpful





Thank You!

If you are interested in doing a master thesis on software visualization, contact me or Antonio:

mariuir@uio.no
antonima@ifi.uio.no

