Project Compara

Final Project for Data to Perception

Marcus Gordon
Michael Carnevale
Minsheng Zheng

Background for Problem

The current data visualization project is directly related to the graduate research assistantship work of the three authors of this project. The graduate assistantship work is focused on OCAD University's special role in the multi-institutional multi-stakeholder joint project iCity. The iCity project brings together academic, government, and industrial partners in order to develop and integrate advanced IT infrastructure for the city of Toronto for the purpose of managing city resources and improving quality of life for citizens. Part of OCAD University's role in this project is to offer informational visualization services in support of other project teams focused on IT system design, city economics analysis, transportation research, policy analysis, etc.

The OCAD Visual Analytics Laboratory (VAL), of which we are a part, have been tasked with providing a taxonomy of the broad software and end user-type landscape that could potentially play a role in developing the new city IT infrastructure. Part of the current research being done by the VAL team is to populate this taxonomy by researching software from a wide-range of applications and technologies, and to categorize them in a usable index.

One of the primary uses of this taxonomy is to help various project teams and stakeholders locate software that may be useful to their design and development process, as well as to understand the end-users who will be the recipients of this technology. In example, this taxonomy may be used to develop a city management dashboard used by city planners and analysts, or may be used to design a city-facing mobile service application. Therefore, the full and complete taxonomy will also include attributes such as types of users, tasks performed by each software, software technologies, file formats and compatibilities, etc. The current taxonomy currently exists as a spreadsheet that is still a work in progress, and we thought that we could take this as an opportunity to develop a custom data visualization that could help users sift through the growing taxonomy of relational qualitative information.

Description of Problem

The data visualization problem we are undertaking is thus how do to convert a large and growing spreadsheet of categorized, hierarchical, and relational qualitative data into a visual index that can be used to quickly understand the broad-scale taxonomy landscape and hierarchical patterns therein, as well represent the associations the taxonomy has with its sub-attributes in a useful way. Put another way, how can we visualize the entire software taxonomy and visualize its relationships with sub-attributes such as user-types, technologies, file-formats, software tasks, etc. Since the ultimate aim of the taxonomy is to help end-users sort through the software landscape to help them accomplish their end-user projects, we will focus primarily on presenting the taxonomy's relationship with user-types. Figure 1 illustrates the overall structure of the taxonomy spreadsheet as it currently is.

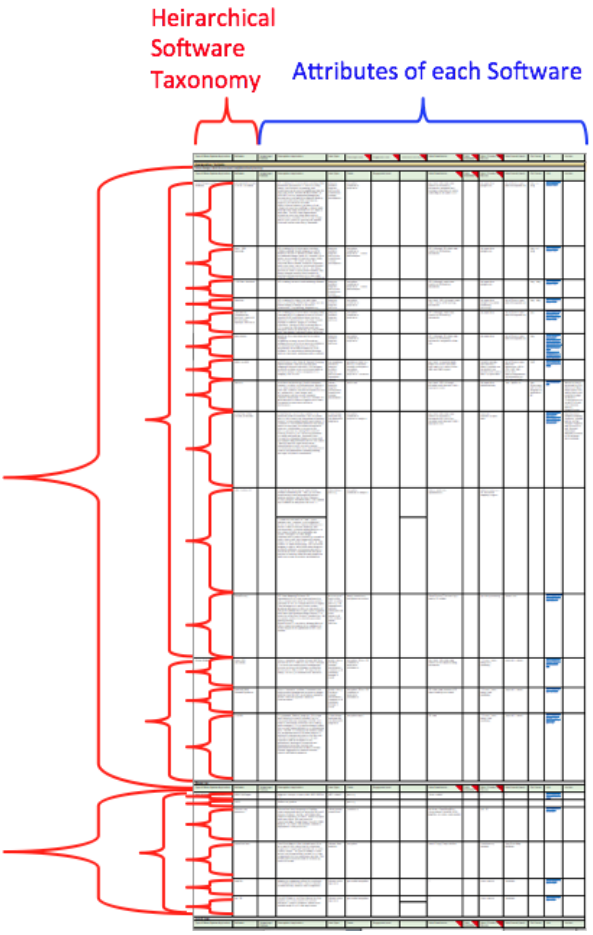

Figure 1: Cropped subsection of spreadsheet showing how the software hierarchy is organized, as well as the associated attributes for each software. Software are categorized to different levels in the hierarchy based on their use and type. Smallest brackets represent individual software.

Purpose of Problem

The purpose of this project is to:

- Visualize the entire qualitative software taxonomy and its hierarchical structure
- Visualize the relationship of each software to its related attributes. A consistent glossary of attribute terms is used, and thus the same attribute terms can appear for multiple software.
- Allow users to identify software associated with specific user-types. Since user-type is one attribute amongst many, generalize the identification algorithm for other attributes.
- Create a visualization technique that can be generalized and scaled as the software taxonomy continues to grow.

Objectives and Questions

The immediate objectives and research questions are:

- The objective is to implement a visualization that can simultaneously show an hierarchical qualitative structure, and its associations to external attributes. Does such a visualization technique exist?
- How can the spreadsheet format be simplified to show the overall structure of its relational contents more clearly?
- How can an indexical structure be visualized in an aesthetic and engaging way?
- Can this visualization be implemented most effectively as a static visual artifact, or can its functional purpose be improved through interactive elements?

Scope and Limitations

The scope and limitations of this data visualization problem includes:

- The dataset we are working with is still a work-in-progress. The spreadsheet currently has empty cells and the overall list is not complete. This implies that the number of items may significantly increase, potentially impacting the viability and usefulness of the final visualization that we create.
- The software taxonomy can never said to be fully complete, as new software are always being released, and identifying every last one in the market is difficult. This is more of a data collection problem than a visualization problem.

- Ideally, the visualization that we develop should be able to generalize to other instantiations of this kind of problem, namely visualizing an entire hierarchical index and its associated attributes.

Literature Review

For this brief literature review, we focused on the general themes of hierarchical structures, node-link representation, and examples of visual taxonomy from software research.

A taxonomy is essentially a system of classification that typically organizes categories by their sub-ordinate or super-ordinate relationships within a hierarchy (Cain, 2011). Taxonomic classification systems exist within many domains of research including software engineering, database work, geneological recording, bibliometrics, etc., but the idea is most typically associated with the classification of organisms in the biological sciences (Figure 2 left). Taxonomies and hierarchies are often visualized using tree diagrams, a visualization technique that dates back centuries and whose early works are often attempts to depict the ancient classification systems developed by philosophers such as Aristotle (Figure 2 right).

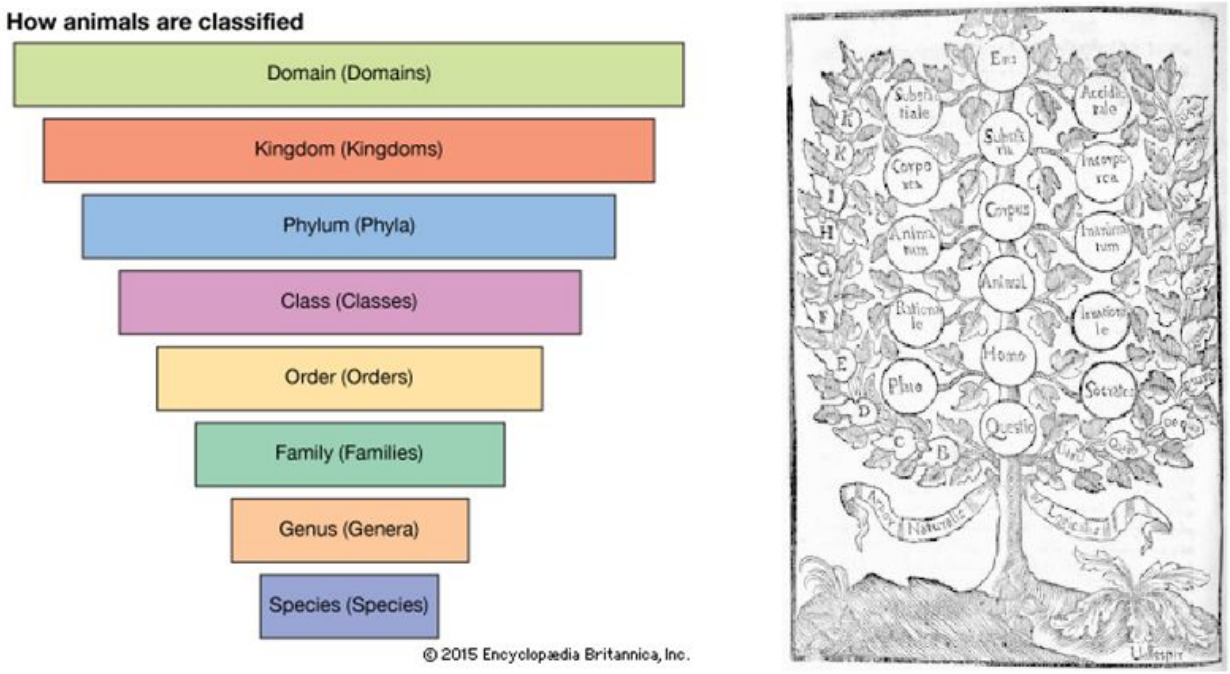

Figure 2: Classification of organisms from the most specific sub-ordinate species level, through to the highest and most broad super-ordinate level of domain (left). Early example of tree diagram visualization

entitled the Porphyrian Tree by Ramon Llull from Logica Nova 1512, depicting the levels of being as originally described by Aristotle (right).

In *The Book of Trees: Visualizing Branches of Knowledge* by Lima (2014), various kinds of tree diagrams and their historical origins are depicted. Tree diagrams are not limited to the typical visual structure that actually looks like a figurative tree, and the concept of visualizing hierarchical structures has taken on more and more abstracted forms over time, each with their unique advantages. Figure 3 depicts the various tree diagrams described in Lima's book. Some points on each tree diagram adaptation is provided here:

Figurative Tree Diagram: Original tree diagrams that looked like literal trees. Diagrammatic terms like root node, branch link, and leaf were adapted from this original metaphor.

Vertical Tree Diagram: Common tree diagram abstracted from figurative trees. Nodes and links can incorporate a variety of visual features, symbols, or icons to convey information.

Horizontal Tree Diagram: Common tree diagram with correspondence to left-to-right reading.

Multidirectional Trees: Tree diagram with no hierarchical ordering along a particular spatial axis. Effective for very large hierarchical systems as visual space is optimally used, but ranking structure becomes unclear due to flexible layout.

Radial Tree: Tree structure ordered circularly, with the root in the center and sub-ordinate hierarchies moving towards the peripheral. Spatially efficient compared to tree diagram (see Figure 4 for visual comparison).

Hyperbolic Tree: Radial tree typically rendered in digital space where some links and nodes are more emphasized than others. Optimizes screen space that radial trees might waste.

Treemap: Space filling visualization depicting hierarchies using nested rectangles. Size of rectangles can correspond to quantitative attributes. Spatially efficient and legible.

Voronoi Treemap: Similar to Treemaps but use polygons rather than rectangles to present hierarchies. Size of polygons can correspond with quantitative attributes. More

spatially optimized than Treemaps and hierarchies are easier to distinguish due to non-recurring shapes.

Circular Treemap: Similar to other treemaps, hierarchies depicted as shapes within shapes, and size can correspond to data. Unlike other treemaps this type wastes visual space.

Sunburst: Treemap structure ordered circularly. Hierarchies depicted with root in center and sub-ordinate hierarchies moving towards the periphery. Size of shapes can correspond to data. Spatially efficient but distinctions between layers can be hard to perceive.

Icicle Tree: Combination between conventional Tree Diagram and Treemap. Organized top-to-bottom or left-to-right, hierarchy is shown as adjacent rectangles to show rank. Size of shapes can correspond to data. Not as spatially efficient as conventional Treemap.

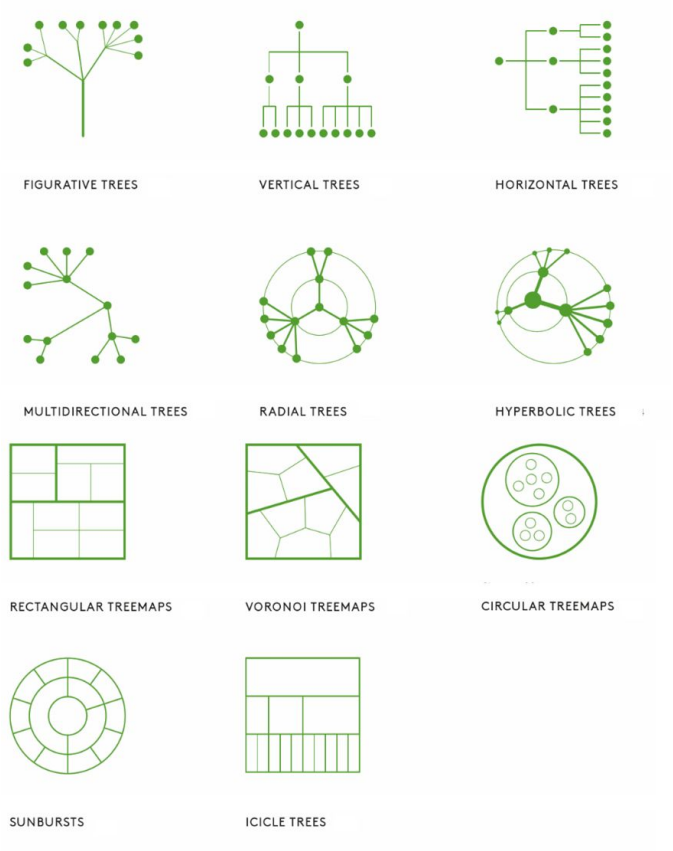

Figure 3: Depiction of various hierarchical visualization types inspired by the initial tree diagram as shown in Lima (2014). Each visualization type has unique advantages and disadvantages.

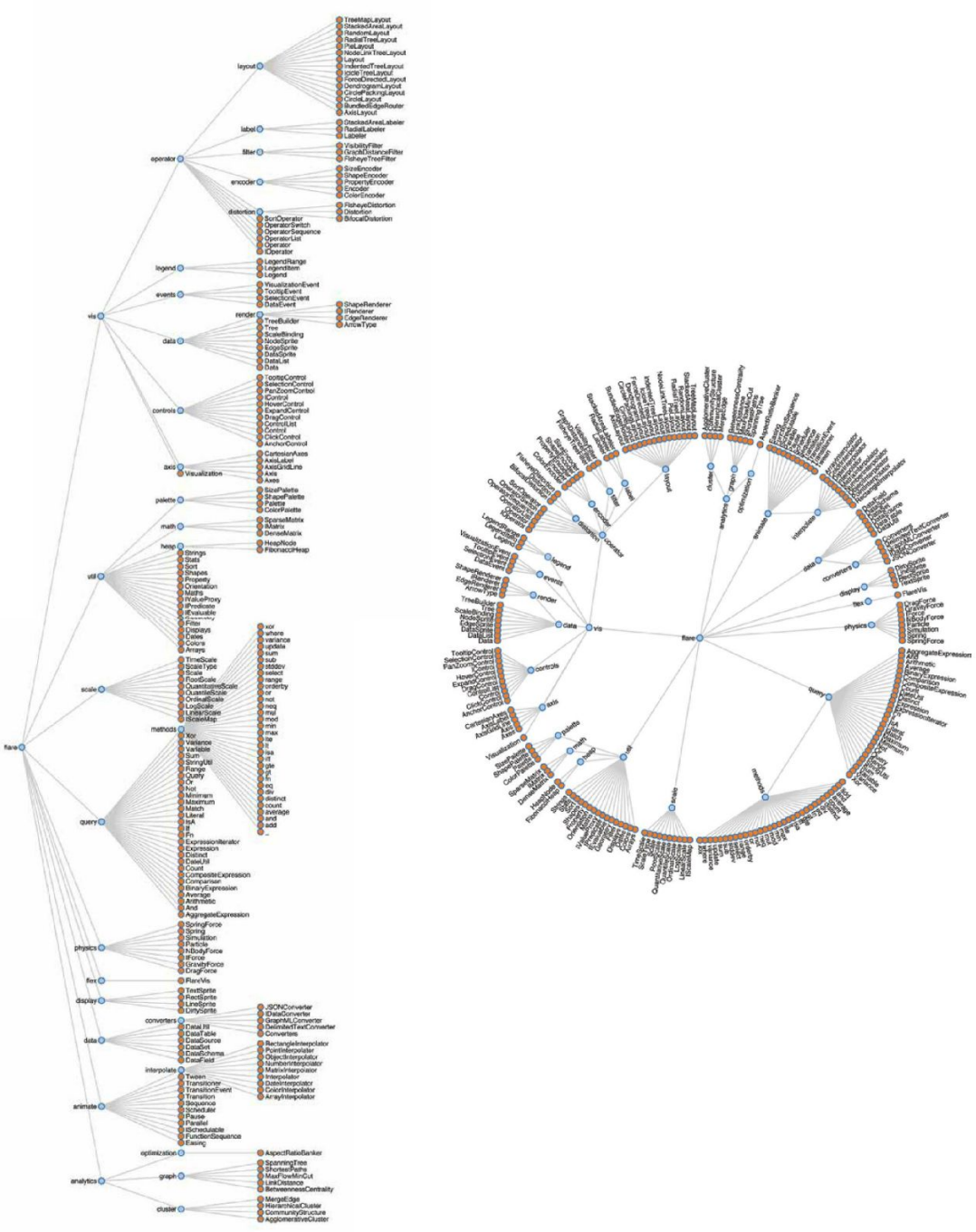

Figure 4: Horizontal tree diagram vs. radial tree diagram. Both diagrams depict the same hierarchical data, namely the code structure of an open-source software package called Flare (Heer, Bostock, Ogievetsky, 2010). While both effectively show the hierarchical structures, the circular diagram is spatially more efficient. The horizontal diagram however follows the typical left-to-right reading pattern.

The above examples provide options for visualizing hierarchical structures, but our visualization problem also involves revealing attributes associated with individual software, which in the current project context are found as the most subordinate leaf

nodes. Visualizations showing software and associated attributes can be found in software engineering research. One such example comes from Grimstead et al (2005) where they create a list of categorized software systems and score them based on their scalability. As can be seen in the table they present, these software systems are each associated with the presence or absence of some attributes. To summarize the data, the researchers made numeric attribute values and averaged them, and then visualized the scores along the axes of a polar plot to create shapes for each software group (Figure 5). Note that leaf-level taxonomical information is lost in the averaged the polar plot.

Visualization System	*Number of Users*	*Access Control*	*Communication Architecture*	*Transmitted Data*	*User Synchronization*
Problem Solving Environments					
CAVS [30]	128	None	Single server	Raw data	Loose
COVISA [53]	*10*	*Unspecified*	Single server	Raw data	Loose
COVISE [33]	10	Per session	Single server	Scene graph	*Loose*
CUMULVS [32]	*20*	Per object	Single server	Raw data	Loose
ICENI [19]	*10*	Per object	Single server	Frame buffer	*Loose*
MANICORAL [15]	*10*	None	Single server	Raw data	Loose
Sieve [29]	*100*	*Unspecified*	Peer-to-peer	Raw data	Loose
Multi-User VR Environments					
Avango [51]	*100*	None	Multiple servers	Scene graph	Loose
Community Place [35]	700	None	Multiple servers	Scene graph	Loose
DEVA3 [40]	*100*	None	Multiple servers	Scene graph	Asynchronous
DIVE [17]	*1,000*	None	Peer-to-peer	Scene graph	Loose
ERCIS [6]	11	None	Single server	Raw data	Loose
MUVEES [10]	60	None	Multiple servers	Scene graph	Loose
Collaborative VR Environments					
COVEN [37]	*100*	Unknown	*Peer-to-peer*	Scene graph	Loose
CSpray [39]	*10*	*Per object*	*Peer-to-peer*	*Raw data*	Asynchronous
CVD [34]	*10*	*Unspecified*	*Peer-to-peer*	*Scene graph*	*Loose*
DIS [1]	*1,000*	N/A	Peer-to-peer	Raw data	Loose
DIVISION Mockup [41]	*10*	*Unspecified*	*Peer-to-peer*	Scene graph	*Loose*
EQUIP [20]	*10*	*Unspecified*	Single server	Scene graph	*Unspecified*
HLA [2]	*1,000*	N/A	Peer-to-peer	Raw data	Loose
Nomad [55]	*100*	*Unspecified*	Multiple servers	Scene graph	Loose
Octopus [23]	*100*	*Unspecified*	*Single server*	*Unspecified*	Loose
PaRADE [43]	1000	Per object	*Peer-to-peer*	*Raw data*	*Loose*
RAVE [22]	100	Per object	Multiple servers	Scene graph	Loose
SCAPE [25]	*10*	*Unspecified*	Single server	N/A	N/A
StudierStube [45]	*10*	Per object	Peer-to-peer	Scene graph	Loose
Virtual Worlds 5 [28]	1,000	*Unspecified*	Multiple servers	Scene graph	*Loose*
Multi-Player Online Games					
Age of Empires [8]	8	Per Object	Peer-to-peer	Raw data	Loose
Butterfly.net [9]	20,000	Per Object	Multiple servers	*Unspecified*	Loose
Half-Life [7]	100	Per Object	Single server	Raw data	Loose
Net-Z [14]	*100*	Per Object	Multiple servers	Raw data	Loose
TRIBES [18]	128	Per Object	Single server	Raw data	Loose
Multi-User Enabling of Single-User Applications					
Access Grid [11]	*10*	*Unspecified*	Multiple server	Frame buffer	*Unspecified*
AG Remote Ren [31]	*100*	*Unspecified*	Multiple server	Frame buffer	Loose
NetMeeting [12]	*10*	Per session	*Single Server*	*Scene graph*	Loose
Sametime [27]	1,000	*Per session*	Multiple Servers	*Scene graph*	Loose
SGAB [57]	*10*	Per object	Peer-to-peer	Scene graph	Loose
SharedX [24]	*10*	Per session	Single server	Scene graph	Loose
VizServer 3.0 [46]	*10*	Per session	Single server	Frame buffer	Lockstep
VizServer 3.1 [47]	*30*	Per session	Single server	Frame buffer	Lockstep
VNC [42]	8	None	Single server	Frame buffer	Lockstep
XMX [5]	*10*	Per session	Single server	Scene graph	Loose

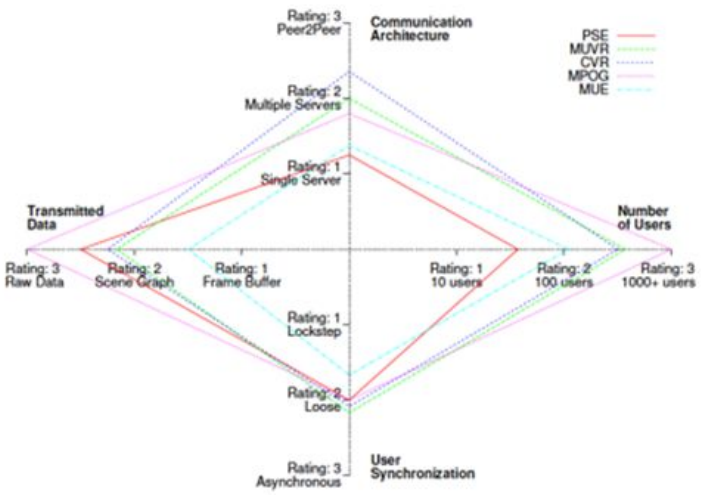

Figure 5: Table of grouped software and attributes (left). Groups of software visualized based on their averaged attribute values in a polar plot.

Another example of taxonomy in the area of software development comes from Kucher and Kerren (2015) where they visualized a taxonomic classification system for text-based visualizations. Their taxonomy is an attempt to understand the inventory of text-based visualization that is a result of social media creating an explosion of data to mine. In their paper these authors offer a taxonomy of text-based visualization software as well as a browser-based visual survey of text-based visualizations that they have identified (http://textvis.lnu.se/). Figure 2 shows the taxonomy they developed where nodes are defining visualization characteristics that could be used to sort text-based visualization types.

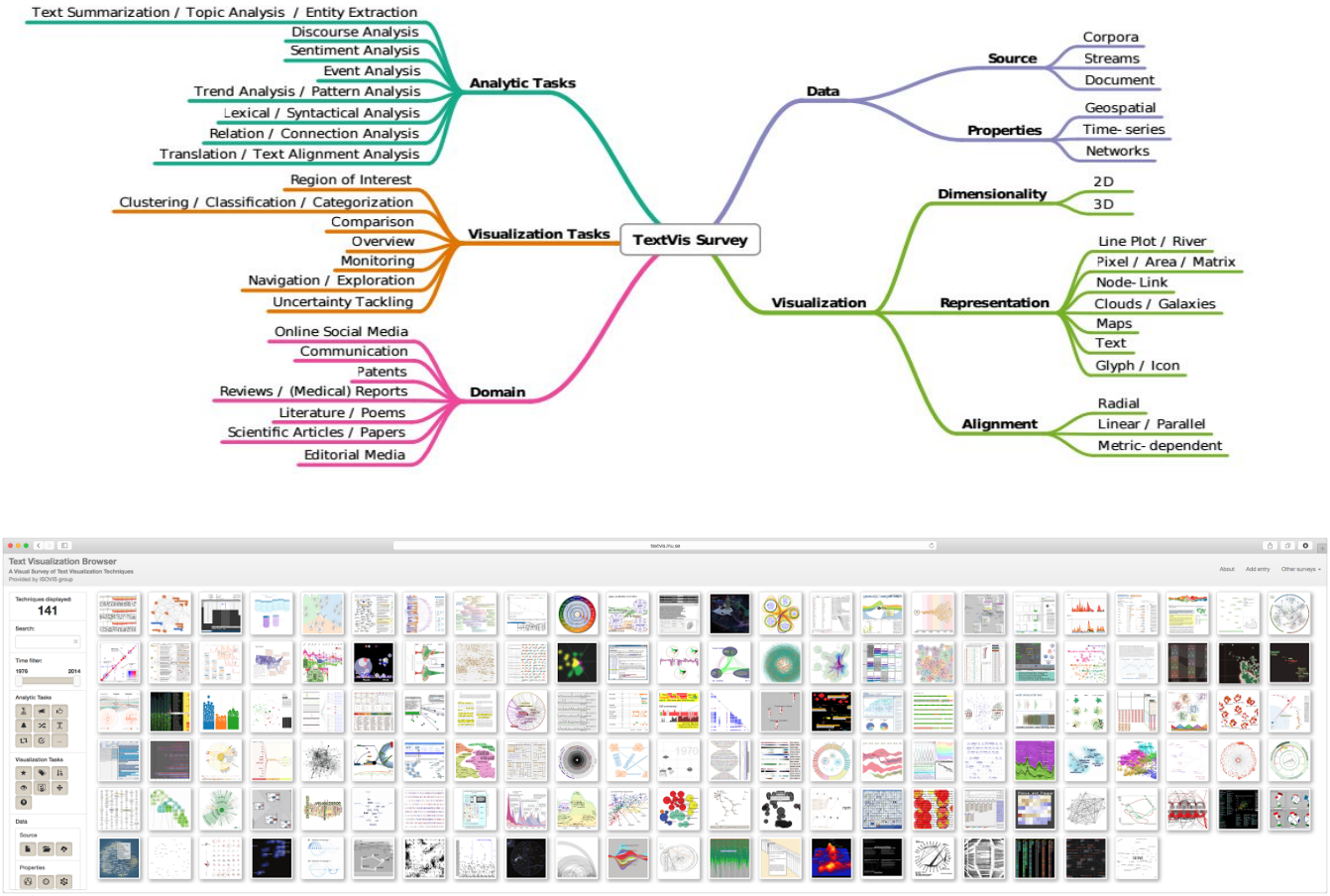

Figure 6: Kucher and Kerren's (2015) visualization of text-based visualization types. Leaf nodes represent defining characteristics, which are ordered in a hierarchy from leaf to root (top). Online visual survey of digital text-based visualization types developed by the authors (bottom).

For our taxonomy and attribute visualization to be comprehensive, we must be able to show connections at the leaf-level revealing individual software to be compared. Types of node-link visualizations showing leaf-level connections include arc-diagrams and bi-partite diagrams. Arc-diagrams draw link connections between nodes that exist along a single axis (Wattenberg, 2002) while bi-partite diagrams show linked associations between two sets of nodes representing distinct classes (Wang et al, 2015). An example of an arc-diagram depicting relations between software task types is shown in figure 6 from a visualization created by Autodesk in 2010 depicting which application tasks are most used in their 3D modelling software 3DsMax. In contrast, figure 7 shows a bipartite graph from a neuroscience article where they showed which of two sets of brain regions are connected (Bohland et al, 2009) .

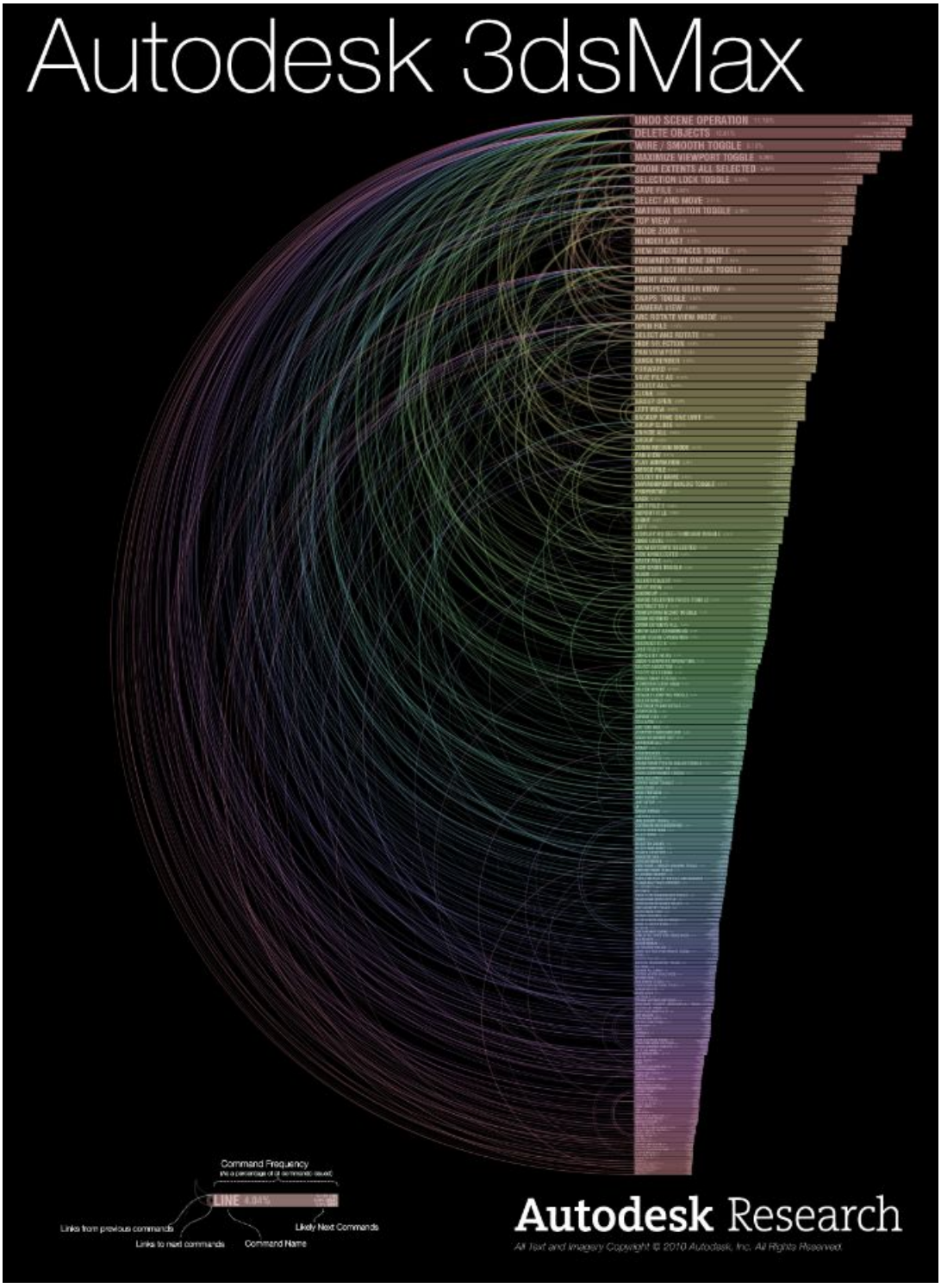

Figure 6: Arc diagram created by Autodesk research plotting the use of specific applications tasks in their software 3DsMax. The arcs show which commands are used together in sequential order.

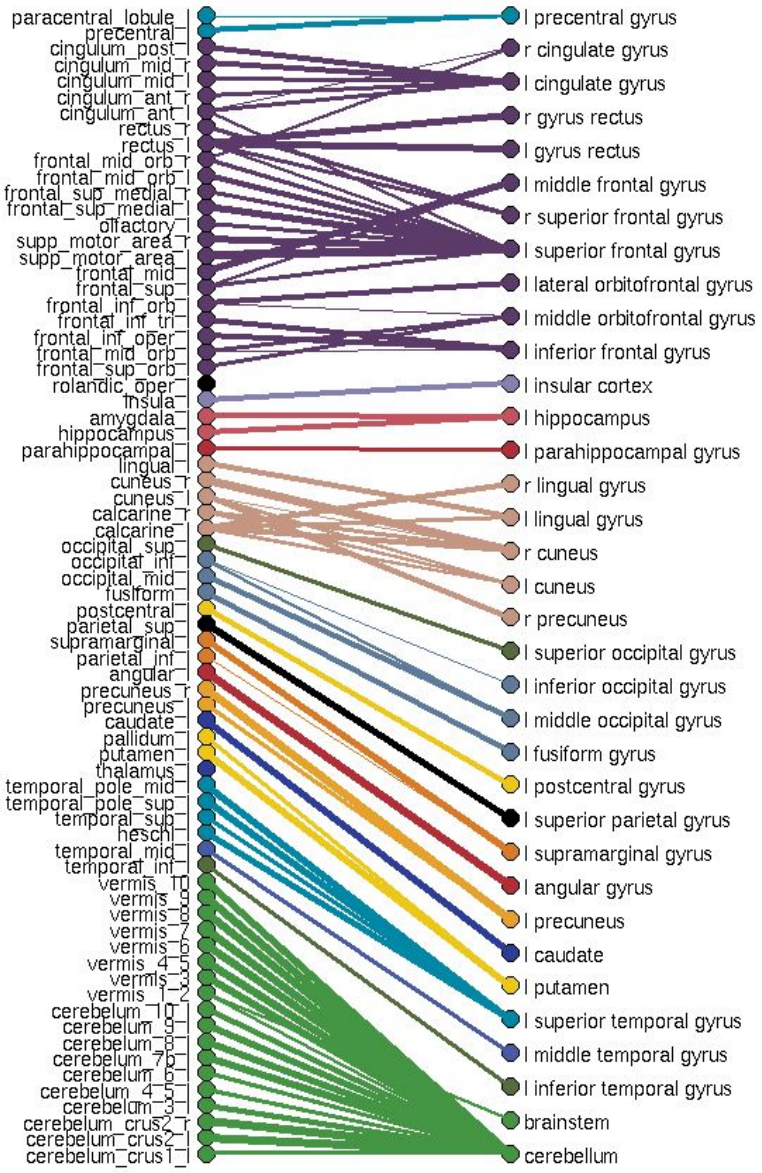

Figure 8: A bipartite graph showing the relationships between two sets of classes. This graph shows which of two sets of brain regions are connected to each other.

While not comprehensive, the above brief literature review offers a view of the landscape for hierarchical and software taxonomic visualization. Interestingly, we did not find an identical example of software taxonomic visualization that solves the exact same problem as our current project mandate. While the taxonomic systems visualized in the literature showed categorization schemes for software, we have an entire database of software that must be organized into a hierarchy and sorted based on external attribute values. In the next section, we outline the design process and results of our prototyping efforts.`

Methods - Design Process

To solve the visualization problem we primarily went through a process of design sketching and rapid-prototyping. For our visualization to be useful, we aimed to visualize as much of the spreadsheet information as possible without making the visualization confusing or difficult to understand. We started our design process focused mostly on the hierarchical structure of the software taxonomy, and later integrated the attribute visualization as well. Below is presented a series of images that demonstrate our design sketching and thinking process.

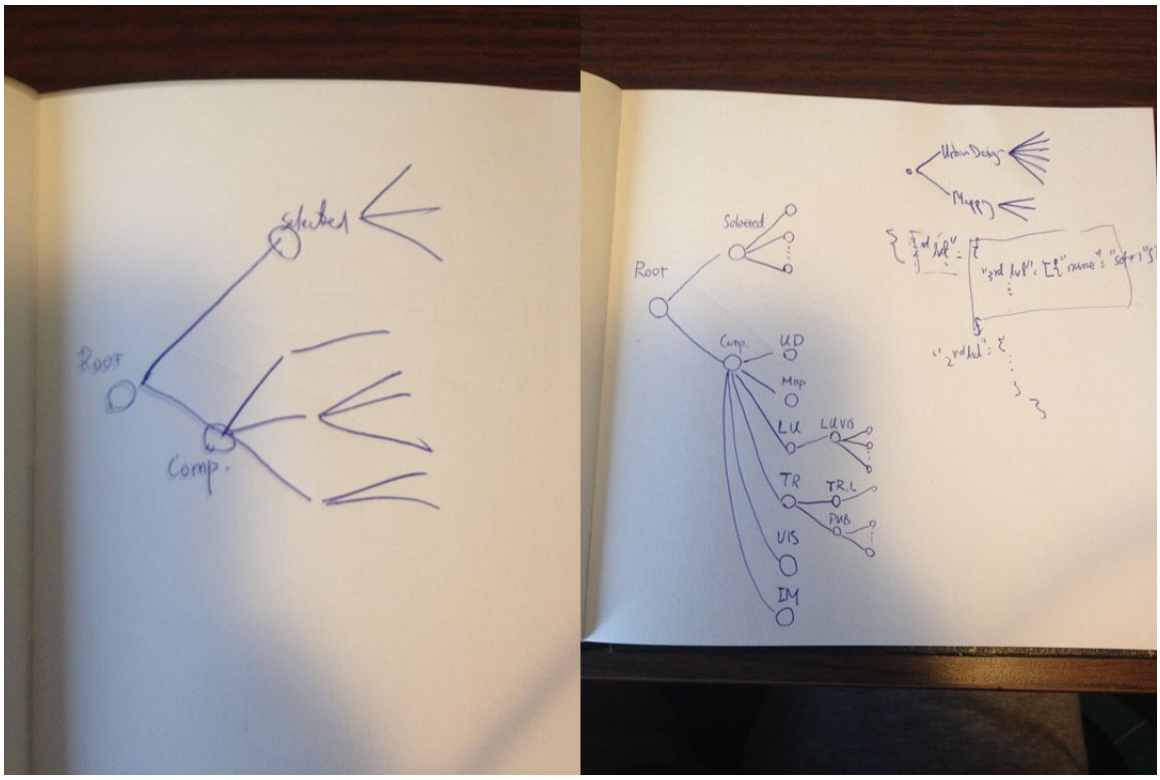

Figure 9: Early sketching ideas for the software taxonomy. The first instinct was to begin thinking about standard tree diagrams.

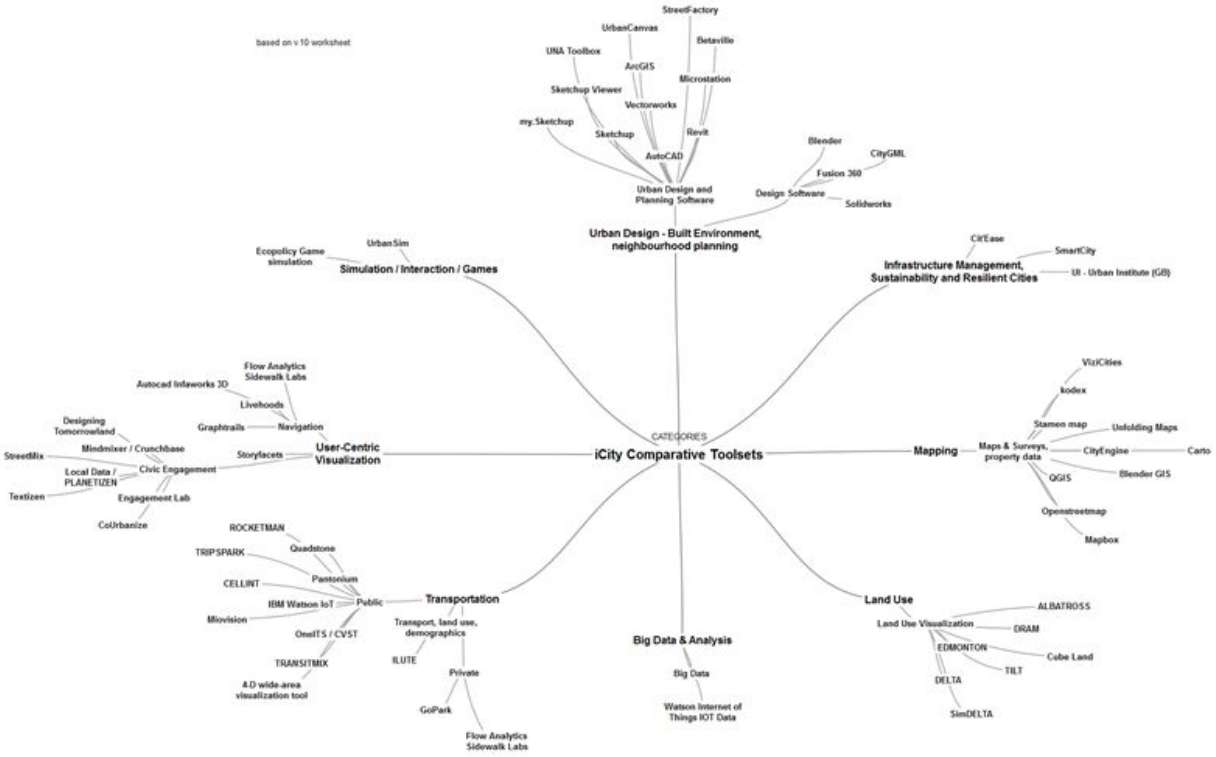

Figure 10: Our first real tree diagram showing the software taxonomy categorized in a hierarchical structure. The root node represents the entire dataset, which branches out into software categories. The final sub-ordinate level, the leaf level, consists of each software system. This diagram is an example of an multidirectional tree diagram and was manually constructed using Simplemind, a mind-mapping application. This first step was promising as it represents our first digital representation. This visualization however was hand-made and not generated by inputting data and running an algorithm, and therefore was not a suitable solution for this project.

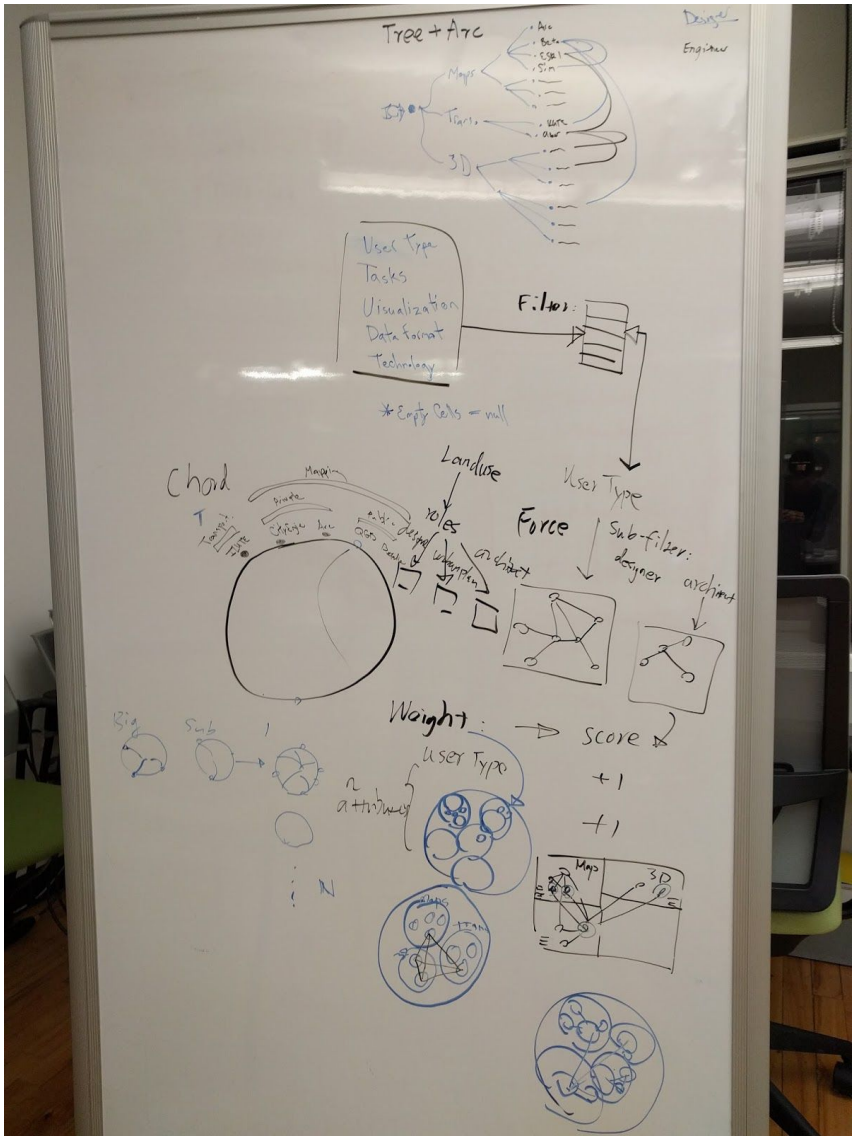

Figure 11: Design sketches trying to think about how to visualize more than just the hierarchy of software and categories, but to also introduce external attributes into the visualization. We experimented with a series of different ideas. The top-right shows our thoughts about trying to connect the leaves of the tree diagram to show relationships between software through arcs. In the middle left you can see our attempts at creating a sunburst-like visualization, with a chord diagram in the center to show relationships. In the middle-right you can see our thinking about force-diagrams, a type of multidirectional tree diagram using web-based spatial auto-organization based on weighted relationships. Finally, at the bottom you can see our attempts at treemaps with inner-connections between leaves, and the same using circular treemaps. In all cases, we basically tried to represent a hierarchy, while then introducing new lines or visual variables. None of these represent a comprehensive solution, but this was an important brainstorming session when considering our various options.

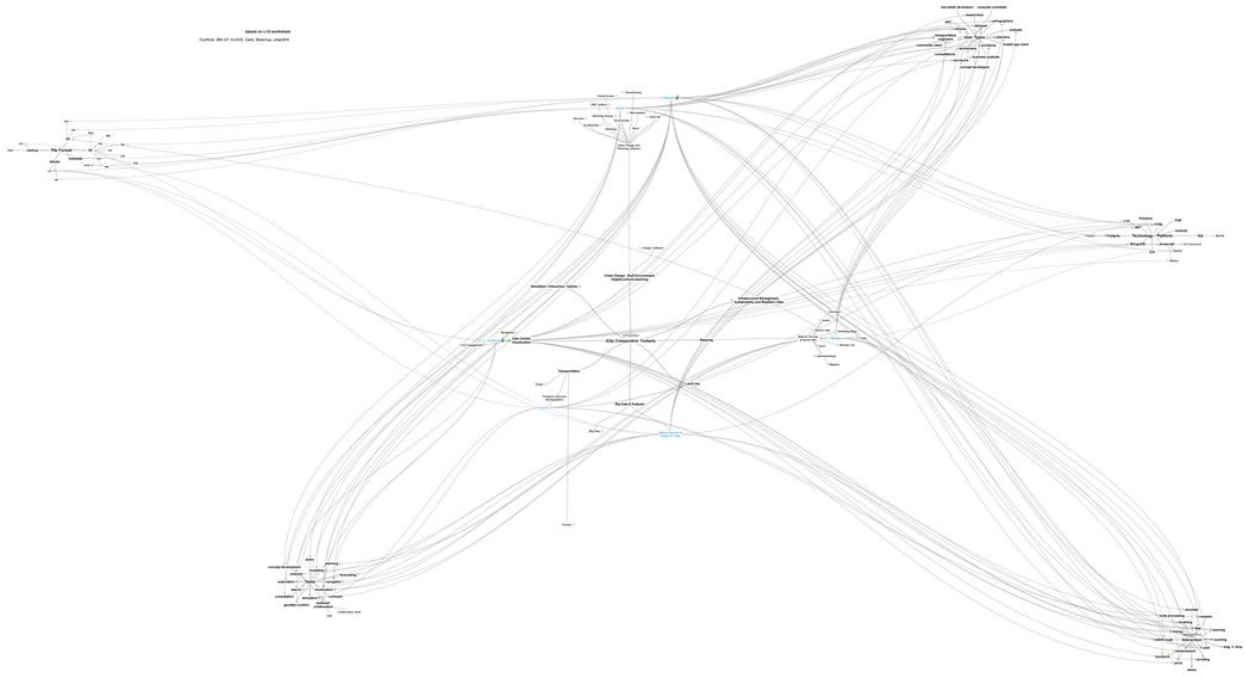

Figure 12: Revisiting our initial tree diagram from figure 10, but now adding external nodes to represent the attribute information. Figure 10 can be seen in the center of the diagram, while the clusters are the periphery represent the external attributes. Connections from the taxonomy leaves to the attribute nodes visualizes which software is associated with which of the attributes. This visualization is a precursor to our final project solution, but reveals some important lessons and limitations. What this clearly shows is that while this complex node-link setup does in fact show all the information simultaneously, it remains rather confusing and difficult to understand. Tracing the connections from attributes to software taxonomy is difficult to follow, and the image is too busy to easily follow and see the intricate relationships. This visualization was also made manually using Simplemind.

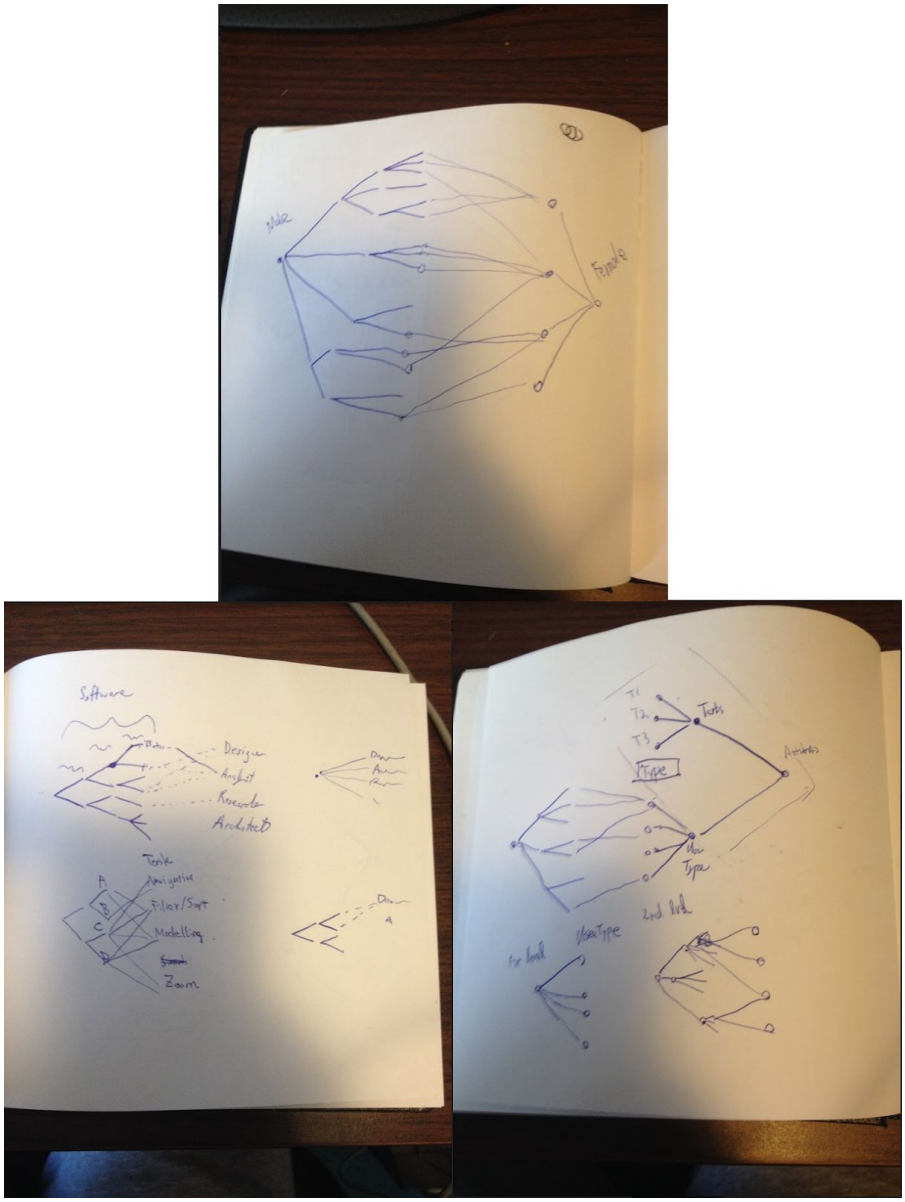

Figure 13: Design sketches showing our conceptualization for connecting an organized horizontal tree diagram representing the taxonomy, with an external set of nodes representing the external attributes. We experimented with the idea of connecting multiple tree diagrams in an ordered fashion, but it became clear that this was interesting but unnecessary, and that the concept of a bipartite graph (mentioned in the literature review) would be sufficient for matching software with attributes. At this level of design thinking we became confident that our visualization concept of combining tree diagram with a bipartite graph could actually work to represent both the taxonomy and the associated attributes. From here we felt comfortable to begin prototyping using more committed and advanced methods, as will be shown below.

Results - Design and Implementation of the Tree Diagram / Bipartite Graph

The final result of our design process and implementation is an interactive web-based tree-diagram/bipartite graph as described in figure 13. The final browser-based visualization can be viewed online at https://val-vis.github.io/compara/main.html. The program scripts and contents can be found on Github at https://github.com/val-vis/val-vis.github.io/tree/master/compara. This section will describe the functionality and characteristics of the final prototype, as well as discuss the design process for the final iterations. Figure 14 shows the final iteration of the tree-diagram/bipartite graph that shows the entire software taxonomy as well as associated attributes for each leaf-level software platform.

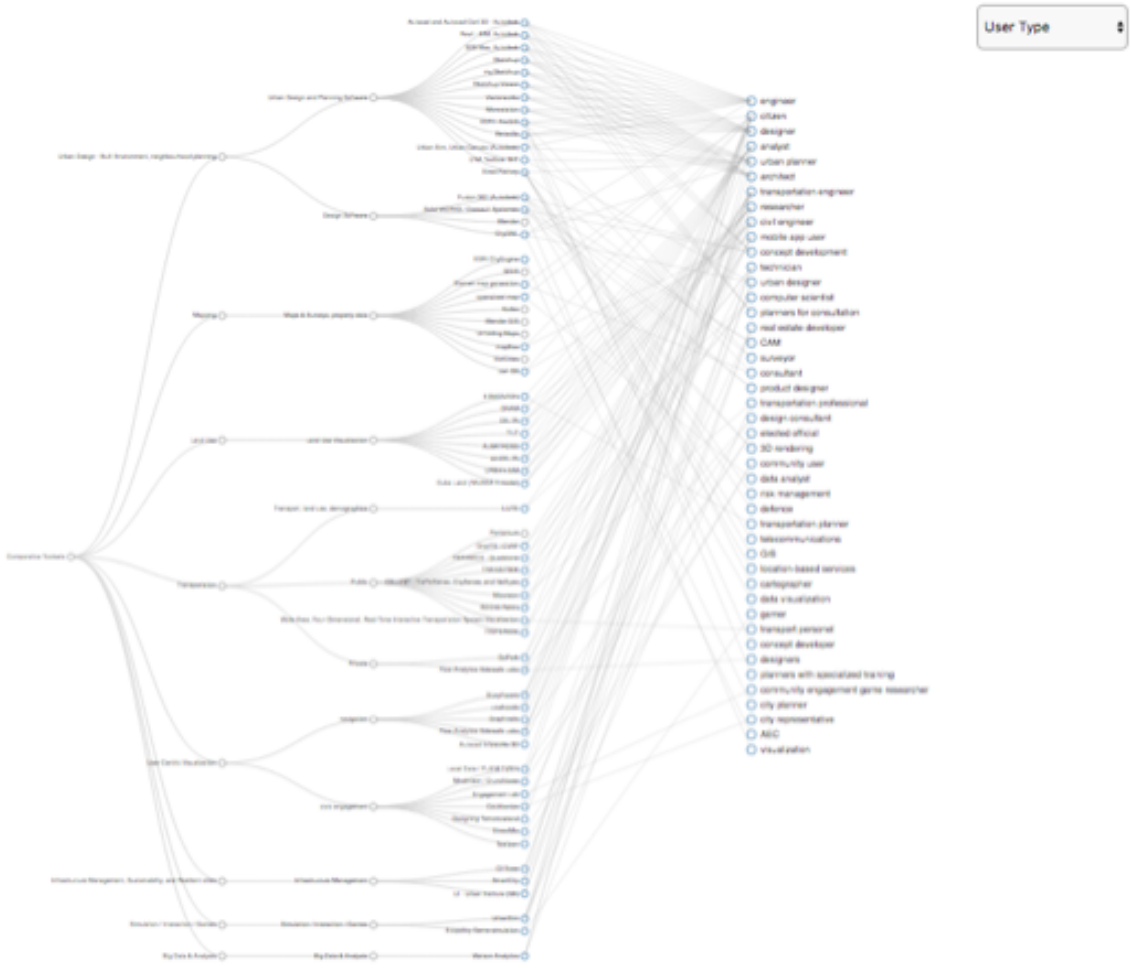

Figure 14: Final tree-diagram/bipartite graph visualization of the software taxonomy and attributes dataset. The horizontal tree-diagram (left) shows the hierarchical categorization structure of the

individual software platforms that are found as the sub-ordinate leaf level of the tree. The bipartite graph is composed of the software leaf-level from the tree-diagram, combined with the set of nodes and links on the right, which represent connections to qualitative attribute values for the category of “user-type”. Effectively, this visualization shows the entire taxonomy of software and the types of users (e.g., designer, planner, citizen, etc.) most likely to benefit from using each software. At the top-right, one can see a drop-down menu that allows the user to switch attributes from “user-type” to “task type” and vice-versa.

Interactive elements were added to the visualization in order to enhance its usefulness in terms of highlighting software/attribute associations for analysis, switching between sets of attributes linked to the tree-diagram, and increasing readability. The interactive elements added include the following:

(1) Link Highlight

Figure 15: When mouse hovers over a node or link between software and attribute, the node-link connection is highlighted in red. If a node has multiple links, all attached links are highlighted.

(2) Node Highlight

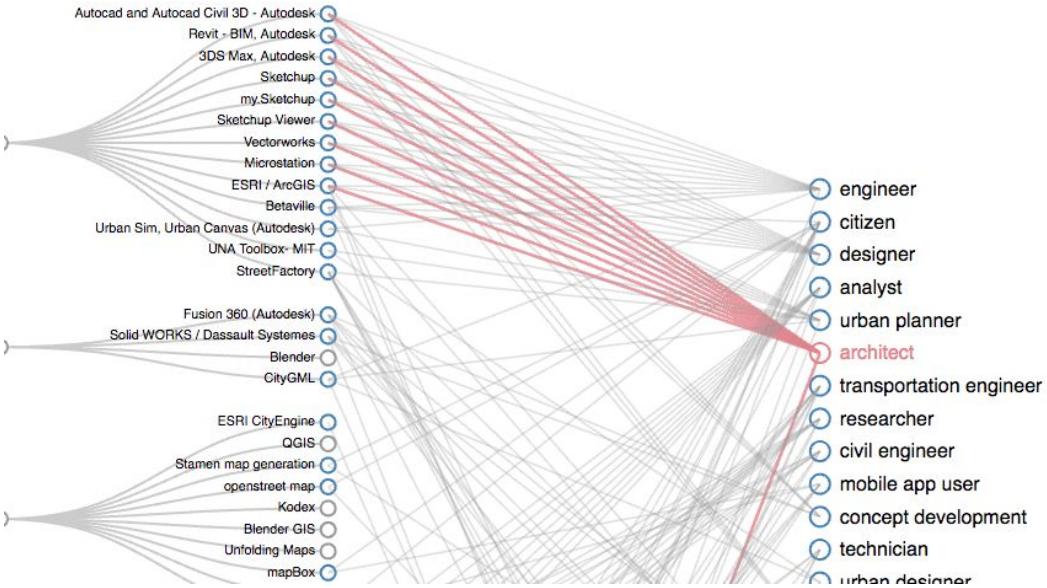

Figure 16: When mouse hovers over a node (either software or attribute), the connections to that node are highlighted in red. This acts to make more clear which attribute features are related to which software platforms, and vice-versa. The bipartite graph with its grey links is difficult to interpret on its own and this was added to help with exploring relationships.

(3) Node Selection

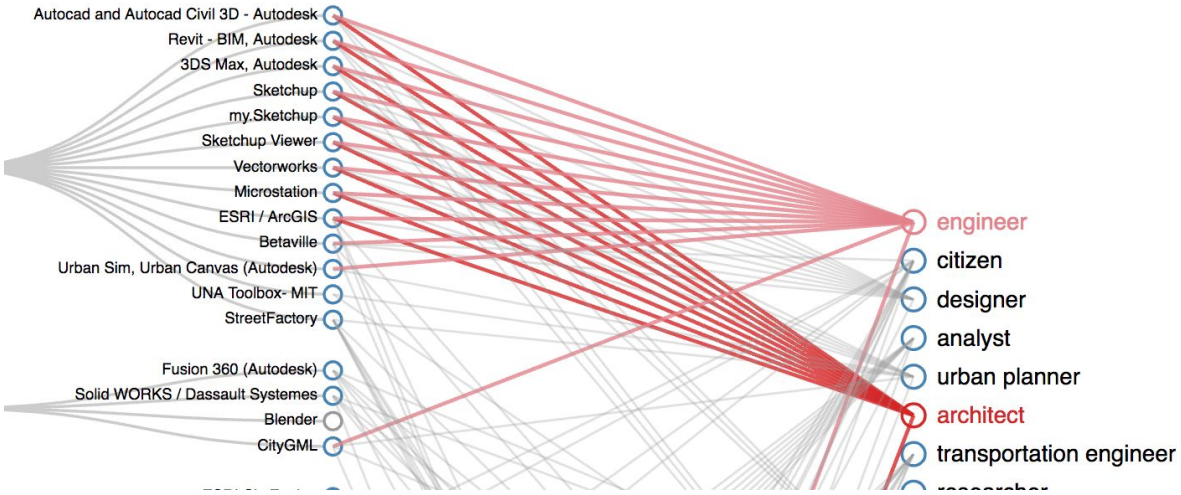

Figure 17: When users mouse-click on a node, the connections to that node are highlighted and locked until further clicks. This allows users to compare one set of selected connections to a second set that is highlighted by mouse-hover.

(4) Attribute Switch

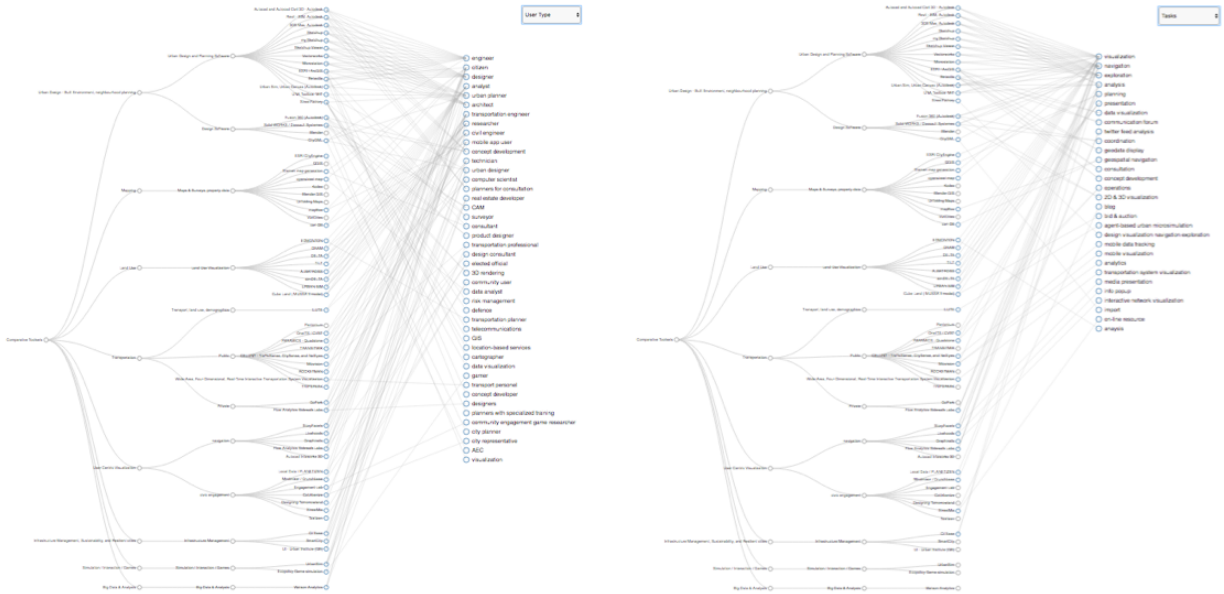

Figure 18: Using the drop-down menu at the top-right of the visualization users can switch between attribute and their value sets on the right side of the bipartite graph. The two attribute sets include “user types” (left) and “tasks” (right).

(5) Zoom

To increase the legibility of our graph, users can zoom close up to the visualization to read text or focus on specific visual regions. The zoom centers on the current mouse location.

Implementation

The implementation of our tree-diagram/bipartite graph is described below. As stated above, the code for this visualization can be found at https://github.com/val-vis/val-vis.github.io/tree/master/compara.

To implement the design for the tree-diagram/bipartite graph, we decided to use the web-browser visualization technology D3. We chose D3 for the following benefits:

- A great amount of visualization techniques are implemented using D3.
- D3 is a robust javascript library with rich development resources and extensive community support
- D3 is open source.
- Interactivity is supported.

- It is easy to integrate D3 into any web application framework. There will thus be less required effort in producing a web-based dashboard application that could be useful for presenting to OCAD researchers or stakeholders.

We planned to write a script that would produce the visualization in a web-browser. To make it more accessible, we hosted it at github.io. The script parses and models the data read from a JSON file and renders a hybrid visualization. The data processing is shown in Figure 19 System diagram.

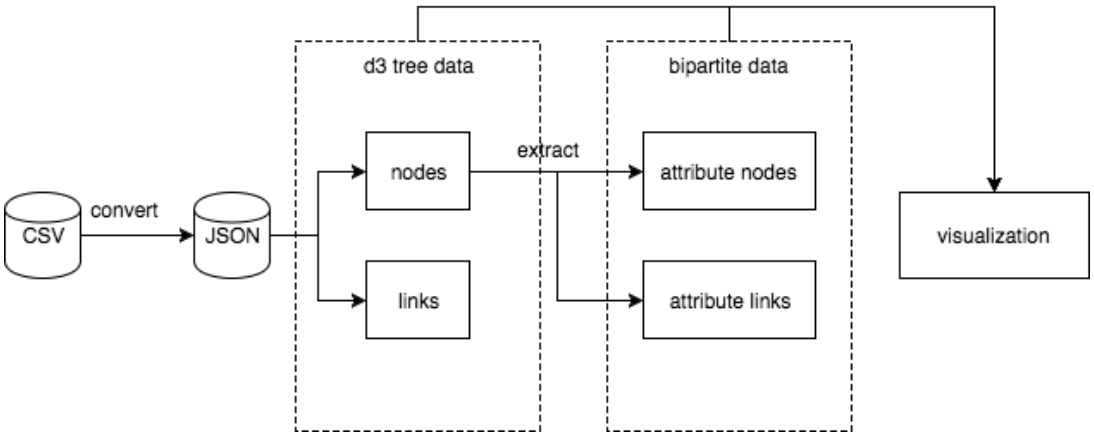

Figure 19: System diagram. First, a CSV file is converted to a JSON file. Next, the data gets parsed and modelled from a JSON file into d3-compatible data structure. Finally, the visualization is rendered.

Since D3 works best with standard file format such as CSV and JSON, we manually extracted and formatted the data from the raw spreadsheet file to a CSV file. Then we converted the CSV file to a JSON file using an external web service with a custom template.

We explain the algorithm to build the tree and bi-partite diagram in terms of node and link construction. For the nodes and links of the tree diagram, d3 has built-in support for generating a tree layout object from a provided JSON data file. The d3 code for creating collapsable tree diagrams can be found at https://bl.ocks.org/mbostock/4339083. For the styling of the leaf nodes, nodes with connections or links (as part of the bi-partite diagram) are in blue and those without connections are in grey.

For the bi-partite diagram, since the raw data itself does not contain explicit relationships, we had to do some text mining to extract and construct relationships between software leaves and its attributes. There are currently two attributes used (i.e., “user type” and “user task”) from the larger dataset spreadsheet which contains others. For example, given an attribute called “User Type”, we construct a list of nodes

of distinct user types from the raw data. We draw a link between a software node and a user type node if the software supports that user type. We also sort the list of attribute nodes by their number of connections from most to least in order to reduce overlap between links, thereby making the visualization more legible. It should be noted that the bipartite aspect of our visualization was coded in d3 custom by the authors of this project.

In terms of interactivity, we implemented pan/zoom and node/link highlighting and selection. Since the visualization is scaled to fit the browser window, nodes and texts are scaled down and may not be legible. With panning and zooming, users are able to freely adjust the area of focus in the visualization. The compound visualization consists of multiple levels and numerous nodes and edges, which may take higher cognitive loads to trace the origin of one or more links. We implemented node and link highlighting to ease that problem. Currently, users can highlight a single link between a software node and an attribute node by hovering the mouse cursor over the link. Alternatively, they can hover on either side of the nodes to highlight a node label and its links if there is any. They are able to persist the highlighting by mouse clicking, but only one node or link can be highlighted at a time.

Conclusions and Future Works

To conclude we will review the initial criteria of this project including the project purpose and the objectives and questions to evaluate the success of our visualization. Finally, we will discuss potential future improvements as well as possible applications for our visualization project.

Based on our initial criteria, we believe our visualization is a viable solution to the stated purpose of this project. We managed to visualize the entire qualitative software taxonomy, making clear its hierarchical structure in a way that is organized and understandable using a horizontal tree-diagram. We implemented a bipartite graph to successfully visualize how attributes are connected to software platforms, even when some software or platforms have multiple relationships. We achieved one of the central uses for this visualization project, which was to help users identify which software are associated with specific user-types. We also created a way to discover relationships between different sets of attributes and software platforms using the drop-down menu. We believe this visualization technique can be generalized to other similar problems. This visualization can definitely be scaled to datasets of different sizes to show different hierarchical structures and associations, but comparing qualitative attributes becomes difficult when the structure becomes too large.

Based on our initial objectives and questions, we believe our visualization has appropriate answers for each. In order to visualize a hierarchical structure and associated external attributes, we implemented a tree-diagram/bipartite graph. From our own research, we cannot find an example of this kind of visualization being used in any similar context. It is possible that we have created a new visualization type, and if this is grounds to publish a paper using VAL research as underlying content, we hope to determine this soon. We successfully managed to convert a structured spreadsheet into a visualization that makes the hierarchical structure much clearer than a traditional tabular format. We believe that our visualization has translated an indexical structure into something that is both aesthetic and engaging, incorporating smooth d3 vector visuals with exploratory interactive features. Finally, we believe this visualization is most effective when it supports user interactions, which allow users to solve the perceptual problems that arise when dealing with an intricate and dense visual structure as this one.

Future Improvements for Prototype

While our visualization results succeeded in fulfilling the initial criteria for this project, our visualization could be improved and developed further. Below is a list of suggested immediate improvements that can be made:

- Nodes in the hierarchy should be collapsible, so they users can observe connections made between the hierarchy and attributes at different levels of the hierarchy. This would allow the comparison of aggregated data rather than only individual software platform data points.
- To optimize the visual space and reduce clutter, the link nodes could be made into curved lines rather than straight lines.
- The node links could be optimized to further reduce edge crossings, thus reducing visual clutter. This process would be algorithmic, and automatically optimize for every new set of attributes or changes to the visualization.
- Highlighting parent nodes in the tree diagram should highlight its ancestor nodes and links as well as its children nodes and links in the tree diagram and the bi-partite graph. This would offer a more comprehensive exploration by the user.

Integrating the Visualization into a Comprehensive Dashboard Application

Here we propose a design for taking our visualization a step further and integrating it into a web-based dashboard application that offers information over and beyond what is already available through the tree-diagram bi-partite graph. We have named this proposed design as Project Compara. Figure 20 shows a visual mockup of the dashboard application layout.

To visualize the data as a tree, a web-based dashboard design is created to house the visual and have other support data views to complement it. Based on our research of visualization methods, the force-directed layout as well as treemap reflect the ability to place focus on relationships and hierarchy, respectively.

The dashboard design is created with four functional sections for analyzing the data of the toolsets. The main section is designed to have the main visualization type as a tree, with the root titled as the dashboard title itself: Compara. It expands out into the first level of headings, which expands out into another group of subheadings of the software tools. Changing this tree structure would be based on the JSON data file that is loaded into the dashboard.

The next section consists of a force-directed layout that provides the user the ability to seek direct relationships between software packages. The third section acts as a middle ground, providing general information about the selected tool, including title, product description, the highlighting of the user types and tasks associated. The last section is a treemap visualization that places focus on hierarchy, from main category of software down to the software itself. Additional items considered for the treemap include a list of keywords that contribute to the rationale and description of the categories. This places emphasis on hierarchy allowing the user to have a clear understanding of how software selected in the tree relates to the subject matter of focus, intended by the manufacturer of the tool.

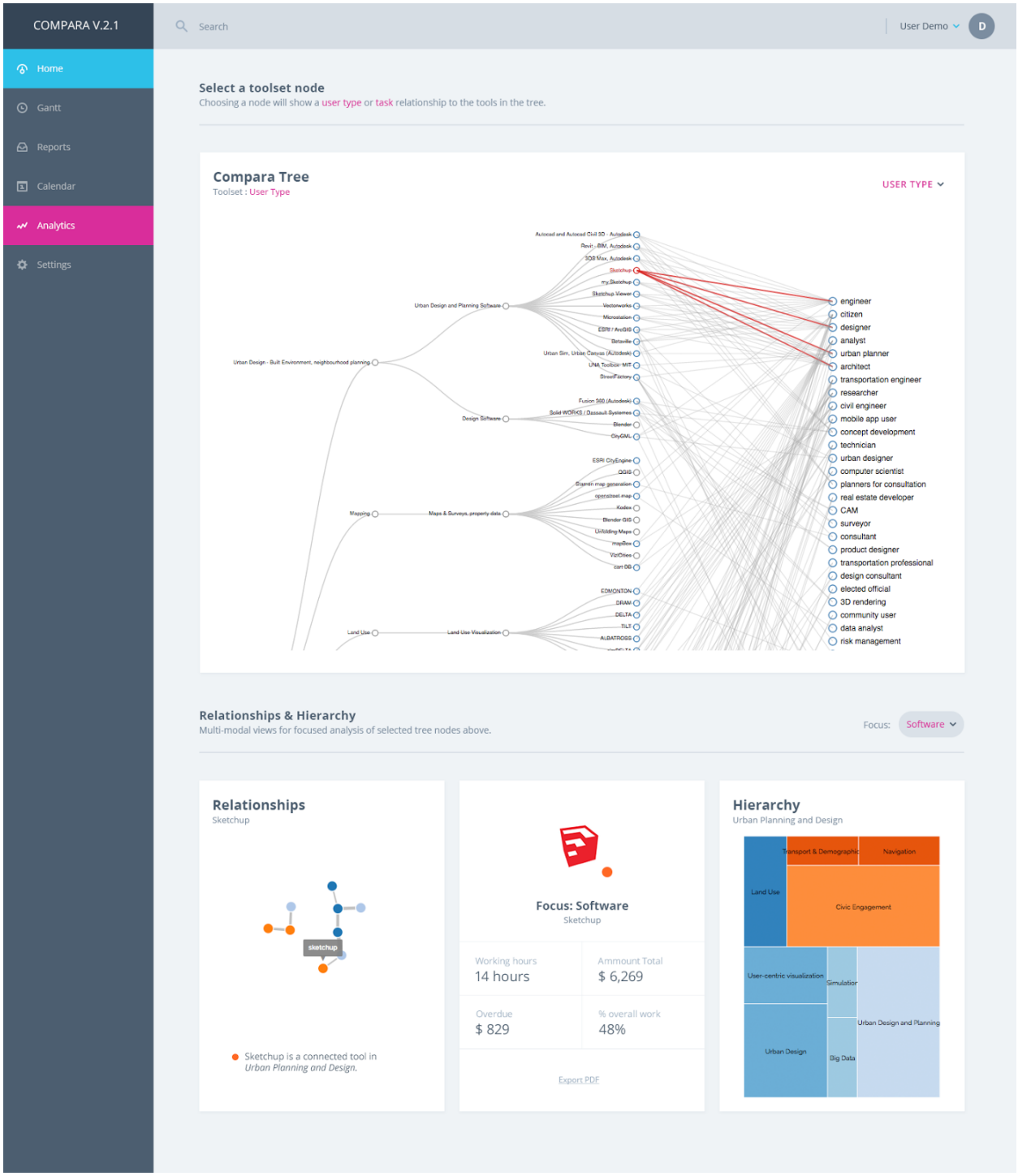

Figure 20: Project Compara is dashboard design created to house the visualization, as well as integrate other support data views to complement it. Based on our research of visualization methods, the force-directed layout (bottom-left) as well as treemap reflect the ability to place focus on relationships and hierarchy, respectively. Extra d3 visualization methods can be daisy chained from the original data set, and further qualitative information can be embedded to offer extra information regarding software platforms or other features (bottom-middle).

REFERENCES

Kucher, K., & Kerren, A. (2015, April). Text visualization techniques: Taxonomy, visual survey, and community insights. In Visualization Symposium (PacificVis), 2015 IEEE Pacific (pp. 117-121). IEEE.

Bohland, J. W., Bokil, H., Allen, C. B., & Mitra, P. P. (2009). The brain atlas concordance problem: quantitative comparison of anatomical parcellations. *PloS one*, *4*(9), e7200.

Parr, C. S., Lee, B., Campbell, D., & Bederson, B. B. (2004). Visualizations for taxonomic and phylogenetic trees. *Bioinformatics*, *20*(17), 2997-3004.

Frost, B. J. (2010). A taxonomy of different forms of visual motion detection and their underlying neural mechanisms. *Brain, behavior and evolution*,*75*(3), 218-235.

Gulden, J., Reijers, H. A., Grabis, J., & Sandkuhl, K. (2015). Toward Advanced Visualization Techniques for Conceptual Modeling. In *CAiSE Forum* (pp. 33-40).

Cain, A.J. (2011). Taxonomy. In Encyclopaedia Britannica Online. Retrieved from https://www.britannica.com/science/taxonomy

Lima, M. (2014). *The book of trees: visualizing branches of knowledge*. S. E. Stemen (Ed.). Princeton Architectural Press.

Grimstead, I. J., Walker, D. W., & Avis, N. J. (2005, October). Collaborative visualization: A review and taxonomy. In *Distributed Simulation and Real-Time Applications, 2005. DS-RT 2005 Proceedings. Ninth IEEE International Symposium on* (pp. 61-69). IEEE.

Wattenberg, M. (2002). Arc diagrams: Visualizing structure in strings. In *Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on* (pp. 110-116). IEEE.

Wang, L., Wu, H., Wang, W., & Chen, K. C. (2015). Socially enabled wireless networks: resource allocation via bipartite graph matching. *IEEE Communications Magazine*, *53*(10), 128-135.

Appendix

Early comments from expert user

- zoom/pan is good
- the visualization should have node/link highlighting
- if embedded URLs are added to the software nodes, you can turn it into an online reference library
- fix duplicate data
- the layout seems to be a hybrid of tree diagram and bipartite graph (based on graph theory)
- reduce crossings of links to show a clearer patterns of distribution and relationship by reordering leaves in the tree and nodes on the right