Quality Models Inside Out: Interactive Visualization of Software Metrics by Means of Joint Probabilities

Maria Ulan, Sebastian Hönel, Rafael M. Martins, Morgan Ericsson, Welf Löwe, Anna Wingkvist, Andreas Kerren
Department of Computer Science and Media Technology
Linnaeus University
Växjö, Sweden
{maria.ulan | sebastian.honel | rafael.martins | morgan.ericsson | welf.lowe | anna.wingkvist | andreas.kerren}@lnu.se

***Abstract*—Assessing software quality, in general, is hard; each metric has a different interpretation, scale, range of values, or measurement method. Combining these metrics automatically is especially difficult, because they measure different aspects of software quality, and creating a single global final quality score limits the evaluation of the specific quality aspects and trade-offs that exist when looking at different metrics. We present a way to visualize multiple aspects of software quality. In general, software quality can be decomposed hierarchically into characteristics, which can be assessed by various direct and indirect metrics. These characteristics are then combined and aggregated to assess the quality of the software system as a whole. We introduce an approach for quality assessment based on joint distributions of metrics values. Visualizations of these distributions allow users to explore and compare the quality metrics of software systems and their artifacts, and to detect patterns, correlations, and anomalies. Furthermore, it is possible to identify common properties and flaws, as our visualization approach provides rich interactions for visual queries to the quality models' multivariate data. We evaluate our approach in two use cases based on: 30 real-world technical documentation projects with 20,000 XML documents, and an open source project written in Java with 1000 classes. Our results show that the proposed approach allows an analyst to detect possible causes of bad or good quality.**

***Index Terms*—hierarchical data exploration, multivariate data visualization, joint probabilities, t-SNE, data abstraction**

I. Introduction

Quality maintenance of software is essential for a sustained development life cycle. While automated testing and continuous integration ascertain quality to some degree, not all such methods are of automated nature. Decisions, such as which components of the artifact shall be refactored preferably next, are hardly obtained automatically and do require a wholly analysis. It is therefore of great help to an analyst to examine the entirety of a software artifact, regardless of its size.

Software systems can be decomposed into logical components or subsystems at different levels to form a hierarchy, and multiple software characteristics can be measured at any given level. Most software quality models are also organized in a hierarchical structure, consisting of quality characteristics and the metrics associated with them. Although such quality models provide a basic understanding of what data to collect and which metrics to use, it is not clear how the metrics should be combined to a provide a single score. Metrics have different units, scale types, and distributions of values, which makes quality assessment challenging. Well-known aggregation methods, such as *mean* and *median*, do not capture all aspects of a distribution which may be relevant to users.

Additionally, the set of all metrics and characteristics that can be extracted from software systems consist on multivariate data that carry information about source code and highlight different aspects of software quality [1]. Hence, to get an overview of the behavior of an entire system or a specific artifact one needs to combine and compare different measurements, effectively building a multivariate representation of software artifacts that conforms to the quality model at hand. Visualization techniques potentially allow the human brain to study multiple aspects of complex multivariate problems in parallel [2]. However, by themselves, multidimensional visualization techniques can be hard to interpret, may omit or misrepresent valuable information from the original high-dimensional space (due to its mapping to a lower dimensionality), and are often not very scalable. An approach which combines these visualization techniques with the actual measurements to support an overview+detail analytical process could provide an efficient way to manage, present, and get insights about software quality.

Extracting meaningful information from multivariate data — presented as plain text or tables, for example — to effectively achieve an *overview* is, however, a difficult task. To make an analysis effective and efficient, smooth interaction techniques are required for combining, selecting, and filtering the data. The degree of uncertainty and subjectivity should be reduced [3] as it is generally difficult to extract meaningful information out of multi-dimensional data. Since human pattern recognition may only be applied in low dimensions, it is necessary to map the data into a lower dimension when dealing with multivariate data [4] to allow the detection of patterns, anomalies or trends which are not immediately evident. The visualization should support an active process of discovery and allow the user to extract meaningful insights and relations.

We introduce an approach for the visual exploration of quality of software systems based on a combination of visualization and metrics. Marginal and joint distributions of metric values are computed and presented to allow users to detect patterns, correlations, and anomalies. The proposed approach makes it possible to compare different software artifacts and provides the basis for further analysis and decision making.

A tool that implements the proposed approach is also introduced, using multidimensional visualization techniques to analyze quality goals and get insights from the data to uncover the roots of good or bad quality. The implementation combines the state-of-the-art dimensionality reduction method *t-SNE* [5] and graphical representations of statistical overviews of quality metrics and characteristics over the whole hierarchical structure of the quality model. It also enables users to examine, compose, and change quality models, and to investigate the effects of these changes in the corresponding multivariate data. The tool accompanies the paper as an accepted artifact [6].

In summary, the work described in this paper introduces the following main contributions:

- A novel framework for aggregating metrics into hierarchical quality models by using their probability distributions.
- An approach to use multiple linked views and dimensionality reduction to visually explore and compare multiple aspects of software quality between different software artifacts.
- The design and implementation of a new visualization tool to support the detection of possible roots of bad or good software quality.

The remainder of the paper is structured as follows. We summarize related work and actual challenges in Section II. Then, we provide needed background in Section III and formulate design requirements in Section IV. To address the challenges in Section V, we introduce our approach and present a tool which fulfills the requirements. We provide theoretical foundations for the approach and a detailed description of the tool. In Section VI, we evaluate our tool and apply it to real-world examples. Finally, we discuss and conclude the results and point out directions for future work in Section VII.

II. Related Work

Quality models are usually defined based on concrete measurements of software metrics [1]. While many quality models have been proposed, sometimes with tooling support [7]–[9], it is not common to find visualization tools that support the interactive definition and manipulation of these models. According to Merino et al. [10], very few software visualization papers target *quality assurance engineers*.

Most metric visualization tools support the combination of metrics on the fly. Explora [11] is a visualization tool that allows the exploration of relationships between multiple software metrics using a visualization technique called *PolyGrid*. Each cell of the grid shows, for a collection of artifacts, the values of four metrics mapped to the position, height, width, and color of rectangles. Relationships between the metrics can then be found while interacting with the multiple linked views. Like our approach, it is geared towards the exploration of relationships between metrics. However, there is no logical aggregation of the metrics into a quality model, and they only support four metrics at a time.

MetricView [12] maps the values of metrics over the nodes of a UML diagram of the system, using a combination of icons and colors for each selected metric. Again, there is no quality model involved, so the metrics are all independent. The relationships between metrics must be inferred from their values in each node of the diagram, and only a limited amount of metrics can be mapped at any time before the visual mappings lose their effectiveness. It is also important to notice that, different than projects such as MetricView, our work does not focus on the structural organization of the software artifacts, i.e., their connections regarding dependencies or the hierarchy of the file system play no role in our visual analysis.

The Small Project Observatory [13] offers the possibility of analyzing many different projects at the same time, showing metrics in tables and stream graphs with the evolution of metrics' values over time. Our tool can also be used to analyze many different software projects at the same time as long as they use the same quality model.

In general, the observations above can also be applied to other metric visualizations [14], [15]. Most of the related work is geared towards the exploration of a fixed set of metrics mapped to specific visual channels. Our tool does not impose any limits on the number of metrics or artifacts.

Furthermore, our tool allows the user to define and manipulate the quality model as they explore the visualized metrics, and it can be used with models from different domains, with no restriction as to the type of artifacts from which the metrics were extracted. Our tool is interactive, works with multiple linked views, and shows the impact of changing the quality model into all the visualized artifacts. The work of Chotjaratwanich et al. [16] is similar in this regard; they also propose to visualize the quality of software projects based on a hierarchical visualization of the quality model, using visual mappings such as color and size on the models' elements. In contrast, our proposal uses a different visual abstraction: while the measurements are based on the model, the visualization is based on the artifacts under analysis. r Finally, it is important to highlight one of the major novelties of this work: our quality models are not based on aggregation as linear combinations of metrics, as is the usual case. We suggest using an automated aggregation approach where quality scores are based on joint probabilities. Our quality model expresses quality as the probability of observing an artifact with equal or better quality; good and bad quality is expressed as higher or lower probabilities to find worse (sub-) systems.

III. Background

We briefly introduce the theoretical background that the tool is built upon, including concepts of software quality, graphs, probabilities, and dimensionality reduction for the visualization of high-dimensional data.

A. ISO standard

The ISO/IEC 25010:2011 standard [17] describes how software attributes and qualities are related. This standard decomposes *software quality* into eight *quality characteristics*: functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, and portability.

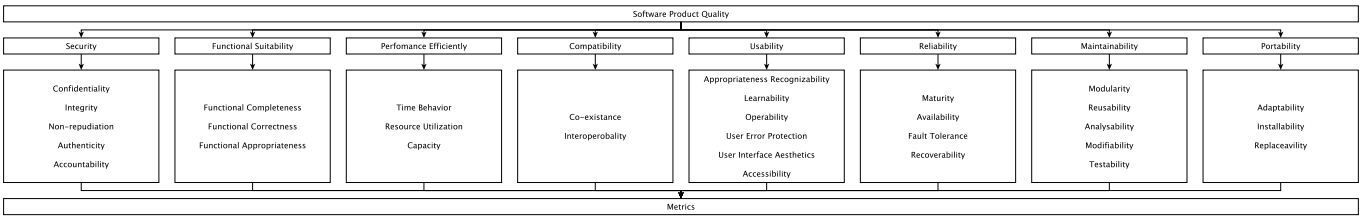

Fig. 1. The ISO/IEC 25010 software product quality model.

Each quality characteristic is further decomposed into *quality sub-characteristics* (as depicted by Figure 1).

The standard assumes that sub-characteristics can be measured by quality metrics. These metrics support the software development process by identifying, for example, cloned source code, bad coding style, performance issues, and security problems.

Ideally, *Software Quality Models* based on the ISO standard are organized in a tree-like structure. However, in practice, the same quality metrics are used to measure multiple sub-characteristics, so the quality models in use are generally structured as *Directed Acyclic Graphs* (DAG).

The main challenge in visualizing a quality model as a DAG is to find a balance between avoiding edge crossings and preserving the hierarchical structure of the nodes. The following diagram (see Figure 2) shows how one and the same quality model based on the same number of metrics and characteristics could be presented in different ways.

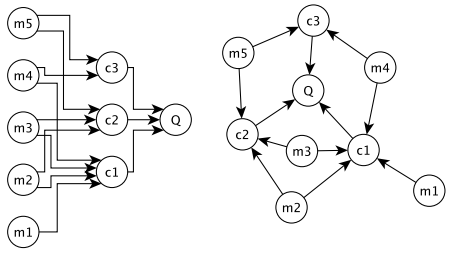

Fig. 2. Different priorities for visualizing a DAG: preserving hierarchy (on the left) and avoiding edge crossings (on the right).

B. Probabilities

We are briefly introducing some basic concepts of probability theory. In practice, one could make measurements experimentally by applying different tools or devices and consider their values as *outcomes*. However, possible uncertainty and randomness play a role in predicting future events. Hence, we are dealing with so-called *random experiments* with unknown results for which outcomes cannot be predicted with certainty. In order to formalize and describe all scenarios, consider a so-called *sample space*, a set of all possible outcomes of a random experiment. To describe all possible combinations of outcomes, we need to consider a collection of subsets of the sample space, which elements are called *random events*. In order to quantify properties, we use the concept of *random variables*. Possible variable values are numerical outcomes of a random event. The measure to quantify the likelihood that specific event will occur called *probability*. Probabilities could be expressed in terms of the *relative frequency* of an event as the absolute frequency normalized by the total number of events.

C. t-SNE

When a software artifact is measured according to a quality model (as described in Section III-A), the result is a multidimensional vector of measures (commonly real numbers), where each component is the value of a metric. A set of software artifacts measured according to the same quality model (and, thus, applying the same metrics on all artifacts) can be seen as a multidimensional data set, similar to a table or a spreadsheet. The visualization of multidimensional data sets is an active research field of information visualization, with different visual abstractions used for different tasks and goals [18].

One way of visualizing multidimensional datasets uses *Dimensionality Reduction* (DR), a set of techniques that reduce and transform the original attributes of the data set to generate a lower-dimensional representation (usually in two- or three-dimensions) that maintains, as much as possible, the original structure of the dataset. The latter is then commonly visualized as a scatterplot. Classic DR techniques include Principal Components Analysis (PCA) and Multidimensional Scaling (MDS), but more recent non-linear DR techniques have shown promise in the visualization of complex real-world datasets. For a comprehensive review, see [19].

One such promising non-linear DR technique is t-SNE [5]. In t-SNE, each item i of a dataset is initially modeled as a neighborhood-based probability distribution P_{ij}, where high values mean that j is a close neighbor of i (in the original multidimensional space). A similar (but not identical) probability distribution Q_{ij} is computed from a *candidate* low-dimensional representation of the data set, and a cost is computed as the Kullback-Leibler Divergence between P and Q. The candidate low-dimensional representation is then iteratively improved with the goal of minimizing the cost as much as possible.

As mentioned above, the final result of t-SNE (as with most other DR techniques) is usually visualized as a scatterplot. Considering that the cost (or the representation error) was minimized, the analyst can then use this low-dimensional representation as a *proxy* to the original high-dimensional

dataset, examining groups of similar points, patterns, and trends in order to get insights about the original data.

IV. Design requirements

Quality assessment of the specific software artifact needs to combine various metrics into a total quality score. On the other hand, most of metrics are defined at the level of individual software artifacts, hence, for assessing the whole quality there is a need to summarize the metrics quality scores at the software project level. We formulate requirements for an appropriate visualization of a quality assessment of an artifact using a quality model. The output of the assessment is provided as multidimensional multivariate hierarchical data.

The potential users of the tool are developers, expert analysts, project managers, and researchers with a goal to explore and find hidden correlations between different aspects of software quality. Below, we formulate seven requirements for our visualization. The visualization should

1) enable users to extract quantitative information, see patterns, anomalies, relations, and structure,
2) involve elementary perceptional tasks to make further analysis and judgment more accurate,
3) provide an overview of the entire behavior of both software system and specific artifact,
4) support data exploration by providing sorting and filtering,
5) support selection of artifact or set of them and get details when needed,
6) support comparison of different artifacts, and
7) support smooth and effortless interaction.

The requirements are based on taxonomies of information [20] and software [21] visualizations, visual analytics [22], cognitive task analysis [23], basic perceptional principals [24], and the well-known *Visual Information-Seeking Mantra:* overview first, zoom and filter, then details on demand [25].

V. Quality Models Inside Out

The choice of a visualization technique depends strongly on the information, task, and context that are of interest for the users, and on the data to be visualized. Under the assumption that software metrics are well-defined mappings of software entities to numerical values, we consider metrics values as data to be visualized. In general, metrics have different units, scale types and distributions of values. These metrics and their values are aggregated to several criteria and finally to a single quality factor. Graph-like representations are suitable for visualizing such kind of data. Quality models can be directly mapped to graphs. The initial informal models are, in general, defined by experts. Preserving their hierarchical levels during graph drawing is therefore more important than avoiding edge crossings. Nodes of such a graphs represent information about metrics, edges represent relations between metrics, criteria, and factors. The main tasks are to compare different software artifacts in order to judge and compare their quality on all hierarchy levels and to detect anomalies which could point at the roots of bad or good quality. Considering these tasks we present an approach for the visualization joint distributions.

A. Approach

Quality does not mean the highest standard of the software, but it should satisfy the users, even if it is not of the highest absolute quality. Probabilities can express such a degree of conformance. The goal of our approach is to provide an understanding of the joint distributions through an aggregation of individual distributions, which will provide an overview and details at the same time. Such an approach provides insights and indicates parts of data for further analysis. A fundamental principle of the approach is the mapping of metrics to graphical attributes. Users can easily explore the values together with the hierarchical structure.

Metrics distributions provide details of different aspects of software quality, and joint probability distributions provide an overview of the total software quality. The similarity between software artifacts can be expressed as conditional probabilities if we consider each metric a random variable. We assume that values for each metric are known and that, without loss of generality, all metrics are defined such that larger values indicate worse quality. We define a metric's score as a probability of finding another software artifact with a metric value less or equal to the given value. The total score can now be calculated as the joint probability.

The probability density function, PDF, and its corresponding cumulative density function, CDF, provide a statistical overview of the data for metrics, criteria, and factors. We use the PDF to find the probability that variable is within an interval and the CDF to find the probability for a variable to be less than or equal to a particular value. The CDF of a random variable X is the function $F_X(x) = P(X \leq x)$, and the connection between a PDF and a CDF can be expressed as $P(a < X \leq b) = F_X(b) - F_X(a)$. Histograms are commonly used to visualize a PDF. For integer metrics, these are based on the area of the rectangle that indicates the frequency of occurrences of metrics values. Because histograms are usually based on equally spaced bins, the height of the rectangle could be interpreted as a frequency. However, visualizations based on histograms could be hard to interpret, especially if the user wants to study frequencies based on unequal intervals. In this case, the user needs to compare different areas of rectangles, which according to Cleveland and McGill [24] is more difficult than comparing lengths of intervals. Hence, it is better to consider the CDF since it only requires the comparison of lengths.

We discuss a tool which supports our approach and fulfills the requirements as formulated in Section IV and solves most of the problems addressed in Section II. In the following Subsections we give an overview of the design, architecture, the interactions, and implementation aspects. In Section VI we exemplify the tool for two use cases. We calculate scores empirically as relative frequencies. Software artifacts are grouped using a *similarity function* and visualized by using a t-SNE-based approach.

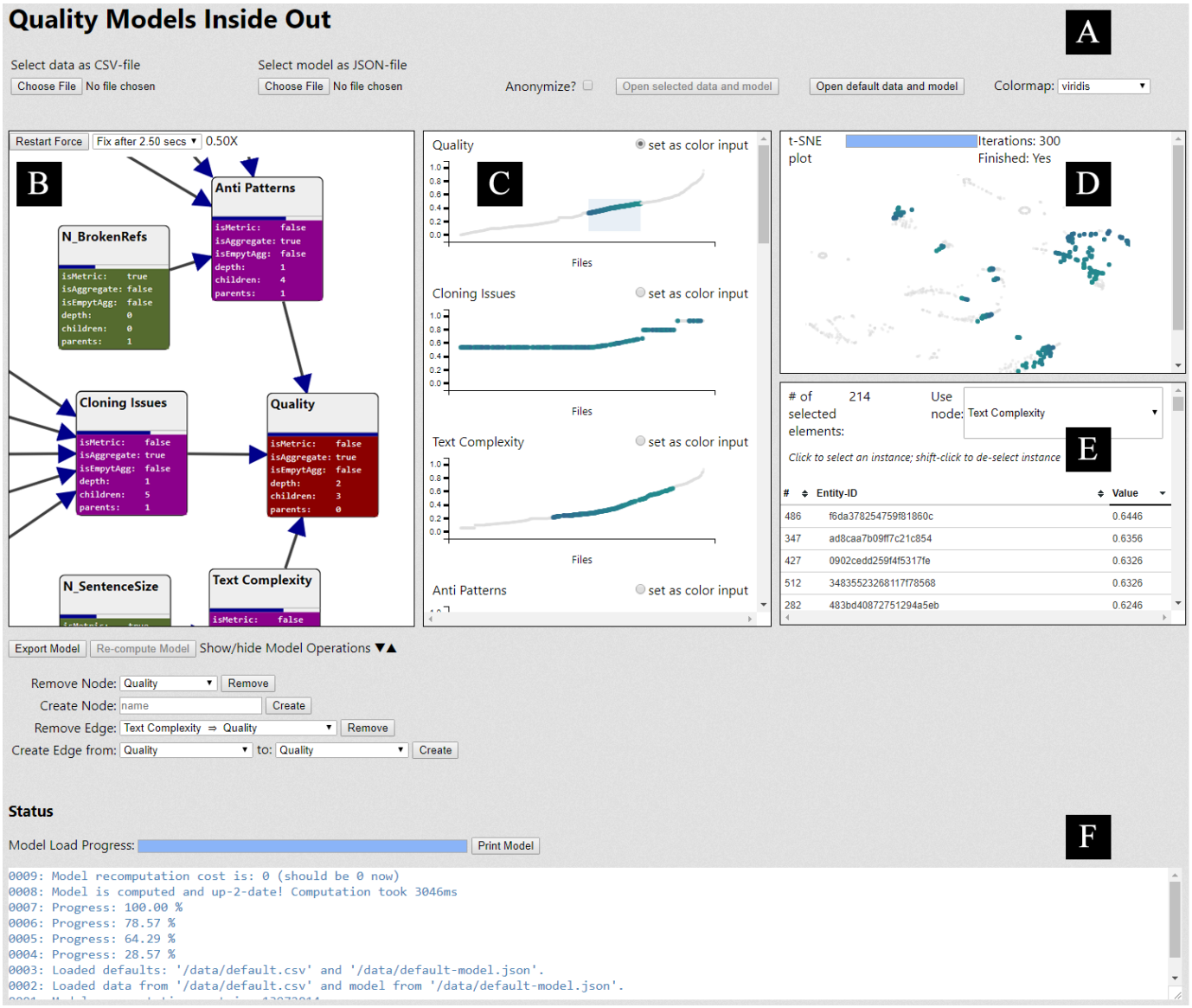

Fig. 3. Overview of the tool with a quality model for assessing technical documentation quality

B. Visualization Views

We give an overview of the features in our tool in the following subsections. Each subsection refers to a section in Figure 3. *Quality Models Inside Out* is composed of multiple interconnected views that provide a filterable view to the underlying input data and the model that describes it.

(A) Head Section: The head includes functionality to open data sets and quality models. It is beneficial to keep the data and its quality model separate, as they may change independently (the latter is not subject to frequent changes). If a new generation of the data is examined, its model could be reused. Furthermore, models can be altered and exchanged to validate them against the same data. An extra button was added to open the included default dataset and model. Those are part of the tool as they provide immediate insight into the clear and obvious structure it uses. Users can present their own data and models comfortable then. An option to anonymize the loaded dataset allows obfuscation of the entities by hashing their names.

All views of the tool share a common color scale for which the input (domain) values can be dynamically changed by the user (see Section V-D). A selection of different colormaps (obtained from ColorBrewer [26]) is found in the head section.

(B) Quality Model View: Quality models represent the relations of direct and indirect metrics within a system. Those entities participate in “based/depends on”-relations. Therefore, we have chosen to use acyclic, directed graphs to illustrate such models. Such graphs allow for quick insights into entity relations and make dependencies and influences visible. Each entity in the underlying model is represented by a node in the graph, and each such node provides technical insights into

the model, such as nesting depth or type, or the number of children and parents. Dependencies are shown using directed edges. The tool allows for disconnected nodes and multiple parents. While the former is useful during composition and examination of models, the latter shows the impact of one child on more than one parent. To not having to alter the model, a node can be temporarily suspended, without having to remove its edges. Furthermore, multiple root nodes (nodes with only incoming edges) are allowed. Although a general quality model has only a single root node (the main quality), this node is most often composed of sub-qualities, which may be adequate candidates for examination, too. A meter for each node indicates its depth, compared to the maximum depth of any node in the model. Currently, the ordinal scale of colors used supports eight hierarchy levels. Technically however, no limit is imposed on the depth or complexity of the hierarchy. It has been shown that human perception is practically bound to perceive no more than eight categorical colors best [27].

When a model is loaded or modified, it must be recomputed. The Quality Model view bears another button for that purpose that is only enabled if a re-computation is necessary. It must be explicitly commenced by the user, as it may be computationally expensive. Information about that cost is logged to the status box' log, whenever a re-computation is necessary. Details about the calculation of the cost are provided in Section V-C.

(C) Distributions View: For each node of the loaded model, we show a distribution of all entities in the data set, according to the aggregation of their values or their underlying data. Each plot is based on an empirical CDF. Since the plots show the cumulative probability distributions, the values on the y-axis are always in the range $[0, 1]$.

The order of the plots is a topological order of the corresponding node in the quality model hierarchy, i.e., the most highly aggregated node is on top, while nodes corresponding to direct metrics are found at the end. Each distribution can be selected as the input (domain) of the color scale to which all views are linked; more details are provided in Section V-D.

(D) t-SNE Plot View: A t-SNE scatterplot is used to support the analyst in identifying similar entities and detecting clusters and patterns in the underlying model's data set. More details about the t-SNE algorithm are provided in Section III-C. We use custom parameter settings that fit the dataset derived from published recommendations [5], [28].

(E) Selection View: The selection view has the general purpose of identifying the entities from the dataset. While the other views only plot their values, they do not allow to identify the entities. This task would be excruciatingly hard with thousands or more entities.

The view only provides the ID of the entity and its calculated empirical CDF value (the probability to find another entity within the selection with the same or lower value). Typically, one is interested in that value according to the general quality, or some root node of the model. However, it may be helpful to identify similar entities with regard to another aggregation or metric. Therefore, it is possible to select another node's CDF values to sort by.

(F) Status View: The bottom view informs about the status of the application; errors, progress, and current state. Our tool facilitates asynchronous parallel work done in the background. It is crucial to feedback such information to the user while waiting.

A loaded quality model can be printed to the output area using a button. This printout is a textual overview of the model's hierarchy and an additional illustration to the graph from Subsection V-B. Since the graph allows modifying the model, another button is offered to export the currently loaded model in its present state. The exported data also includes data about the visual representation, such as each node's position. Exporting will download the model as a JSON-file.

C. Summary of Architecture

Our web-based tool facilitates observable data (through *RxJS*[1]) and linking by using *Crossfilter*[2]. While the former is used to propagate that data and a model is available, the latter is used for brushing and linking in the Distributions-, the t-SNE plot- and the Selection-Views.

Visualizing the graph in the Quality Model View is done using *d3.js*[3] and is based on SVG. The plots, however, use an HTML5-canvas, since the performance with thousands of SVG-elements quickly deteriorates. The plots in the Distributions- and t-SNE plots-Views are drawn using *dc.js*[4]. The t-SNE itself is calculated by the library *tsnejs*[5]. To support sorting in the Selection view, the plug-in *tablesorter*[6] for *jQuery* is used.

The main view is visually and logically componentized so that each view is a component of its own. Albeit this application being data-driven, it is not using a related framework, such as *Angular*. Instead, we decided to use vanilla *ES6* where possible. The head's component is the focal point for starting to use the application, by loading a data set and its model.

D. Interactions

Our tool makes extensive use of *linking* and *brushing* features [20] to connect the various views. Again, we refer to the sections of Figure 3.

(A) Manipulating the loaded quality model: Quality models can vary significantly with regard to their design. While one model is shallow and defines a tree-like structure where all nodes point to one single top-level node representing the overall quality, other models may define deeper hierarchies

[1]ReactiveX, An API for asynchronous programming with observable streams, http://reactivex.io/

[2]Crossfilter, Fast n-dimensional filtering and grouping of records, https://github.com/crossfilter/crossfilter

[3]d3, Bring data to life with SVG, Canvas and HTML, https://github.com/d3/d3

[4]dc.js, Multi-Dimensional charting built to work natively with crossfilter rendered with d3.js, https://github.com/dc-js/dc.js

[5]tsnejs, Implementation of t-SNE visualization algorithm in Javascript, https://github.com/karpathy/tsnejs

[6]tablesorter, Flexible client-side table sorting, http://tablesorter.com/

with intermediate nodes that aggregate other nodes and result in several top-level or root nodes. This makes it impractical to use one layout algorithm that satisfies all those different needs. Therefore, we decided to let the user arrange the model manually and to store positional information within the model. However, we also included the option of a force-based graph layout, so that an initial layout can be obtained.

Furthermore, it is possible to disable a node, so that its impact on a parent node can be examined. An edge between two nodes can be removed or established, to create a parent-child relation. However, multiple parents are allowed, while circular references are prohibited. Nodes can be created and deleted. Hence, it is possible to create quality models with our tool and to validate those. It is possible to start with an empty model and to compose it using the available data and controls. While nodes can be dragged, clicking them will select (scroll to) the related plot in the Distributions View.

The rendered graph of the quality model can be panned and zoomed, so that the nodes' extensive details become visible. These interactions allow a closer inspection of nodes and their connection as well as fitting large models into the constrained space of the view.

(B) Selecting entities in the Distributions View: To aid the exploration of large datasets using dynamic queries [29], a user may select data in any plot using a rectangular selection. Since each node may have a different distribution for each entity, this action will highlight the selected entities across all other plots. The user may make additional selections in other plots, to further filter the data. This filtering applies to all plots, including the t-SNE-plot. Instances not currently selected are grayed out. The plots are in descending order by node-depth, as we assume that the higher the nesting, the more aggregated and therefore more relevant to quality a node is.

(C) Dynamic color scale for comparing distributions: One additional element for the linked exploration of multiple distributions is the possibility to dynamically change the color scale according to a distribution. When one of the distributions is selected (by clicking on *color*), the overall color scale of the tool will change to reflect the values of the selected distribution, i.e., the colors will map the same values as the y-axis of the selected distribution. Since all views are connected to the same color scale, the user will then be able to compare the selected probability distribution against all others by inspecting how the colors appear in each chart. Similar color patterns, i.e., the colors increase from left to right, indicate similar distributions, while random color patterns indicate very different ones.

(D) Selecting entities in the t-SNE plot View: The t-SNE scatterplot is an interactive, low-dimensional representation of the software artifacts and their multiple quality measures, generated with the method described in Section III-C. Similarly to the probability distribution charts, the t-SNE scatterplot is also interactive and allows selecting and filtering points and groups of interest. Since all charts are linked, this allows the analyst to explore patterns of each distribution regarding the points or groups of interest.

(E) Sorting and exploring in the Selection View: The Selection View's purpose is to identify selected instances and to order them by their empirical CDF value. It also prints that value so that similar instances can be found. Initially, the first root node's empirical CDF values are used for the selection. However, the filtered entities may be ranked and sorted using any other node value.

E. Implementation Aspects and Design Challenges

Our tool is based on empirical CDF data, which we calculate on the data manually. Our implementation supports uni-, bi- and multivariate data and is unit-tested. Each node that is connected to a column in the loaded data or based on an aggregation has its empirical CDF calculated. The computation is parallelized[7] and uses *WebWorkers* in the browser. Each node's position in the graph is determined by its children and data sources. A unique deterministic ID is computed from that position and used to detect whether a node or its children need to be re-computed. In that way, only changed nodes of the quality model are subject to the computationally expensive computation of the empirical CDF.

The cost for recomputing a node is expressed as *big-O* [30] worst-case amount of operations, which here is $O = mn^2$, with m being the number of columns (here: children) and n being the number of entities. This information is used to display progress to the user while the calculation is ongoing. The computation of a node is done in a depth-first manner.

We support CSV-based data that contains one or more columns to generate an entity-ID from and any number of columns that hold numeric data. The quality model is defined as a plain JSON-file, whose format supports the following features: (a) a list of columns to use for generating an entity-ID and (b) a list of columns to use as nodes in the model. Each node definition can define or reference any other of the defined nodes as children. Also, each node can be disabled and carry optional layout information for the graph-view.

VI. Use Cases

We study two real-world cases to evaluate our prototype. We demonstrate the usefulness and practicality of the proposed approach and tool with a visual exploration of real-world multivariate hierarchical datasets represented by quality models from the domains of technical documentation quality and software maintainability assessment.

A. Technical Documentation

In this data set, the observations correspond to a quality model which follows the *Factor-Metric-Criteria* structure [31]. The quality model relies on 31 metrics that are used to evaluate eight criteria: Cloning Issues, Anti-Patterns, File Complexity, Hierarchy Complexity, Language Issues, Referential Complexity, Text Complexity, and Validity Issues (see Figure 4).

[7]Parallel.js, Easy multi-core processing with JavaScript, https://github.com/parallel-js/parallel.js/tree/master

The metrics' values were collected using *Quality Monitor*™[8]. Details about the analysis engine are available in a previous work [32]. The complete dataset is composed of 33 projects with a total of 21,121 documents. For each metric across all documents, a score was calculated based on an empirical CDF. Each criteria score was quantified as a joint probability distribution calculated as an empirical multivariate CDF of all metrics that contribute to the criterion. The factors' scores are expressed as the multivariate CDF of their criteria.

In order to better illustrate the results while examining this case study, we worked with a subset of ten metrics, three criteria, and a sample of 5,000 files (see Figure 5).

The goal in this case study is to find common flaws and defects, as well as duplicate documentation. Also, we tried to identify documents which share a similar value for text complexity. By looking at the overall quality, it could be determined that documents with lower quality are spread out across almost the whole sample population and its detected clusters, as of the t-SNE (see Figure 6). The plots for Cloning Issues and Anti-Patterns show that with an approximate probability of 60% for the former resp. 85% for the latter, documents with lower joint probabilities can be found. Interestingly, those documents seem to reside in all but one clusters for Cloning Issues and precisely three clusters for Anti-Patterns. From that, it can be concluded that cloning affects almost the entire documentation. Furthermore, about half the files seemingly have the same value for cloning issues, which indicates the same degree of cloning for half the sample population, and hence probably some automatic operation that leads to it, such as automatic generation of documentation from an application programming interface (API).

Since Anti-Patterns are based on metrics such as broken references, it is likely that affected documents are in similar portions of the documentation and that those share some similarity, such as the same editor, type of generator, or were subject to some common error. Also, after having removed two metrics not related to references, we almost obtain the same results for Anti-Patterns. We can conclude that this criterion is measuring too many different things, and that it may be split into different criteria (such as "Reference Issues").

B. Open Source Software (default quality model)

In this data set, observations correspond to a quality model for maintainability assessment based on 17 metrics [33]. The metrics' values were collected using *VizzMaintenance*[9]. The dataset for visualization was composed of source code of the latest version of *JUnit*[10] (last modified April 2018), consisting of 1,119 classes in total. For each class, metrics' scores were calculated as values of the empirical CDF function, then Maintainability was expressed as multivariate CDF of its metrics.

[8]Quality Monitor™, web-based quality analysis tool for software and technical documentation, http://iqm.arisa.se/iqmonitor

[9] VizzMaintenance, Eclipse plug-in supporting Eclipse 3.5 or higher, http://www.arisa.se/products.php

[10]JUnit, A framework to write repeatable tests,https://github.com/junit-team/junit5

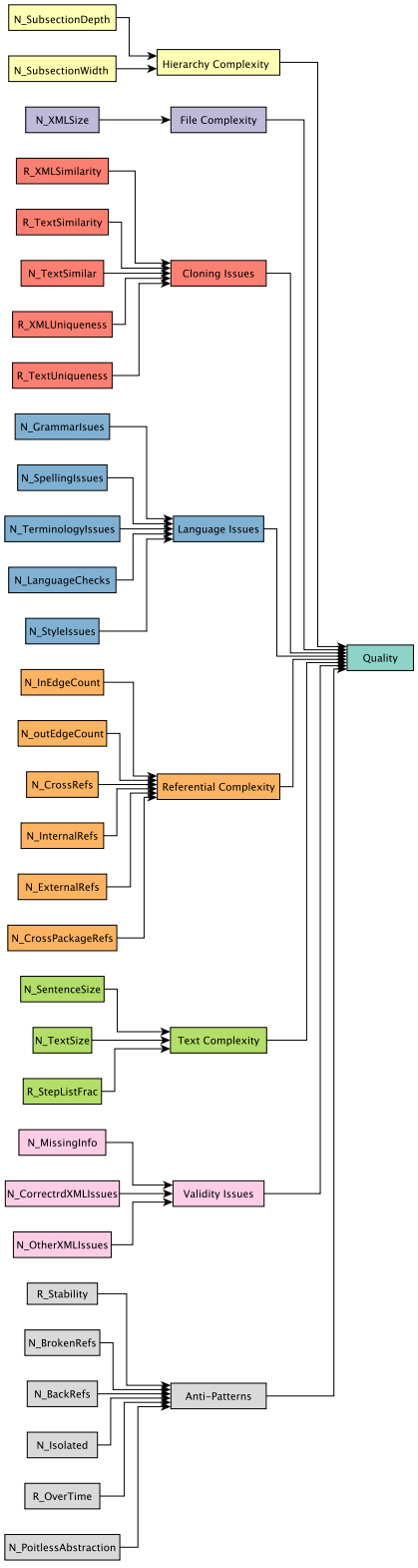

Fig. 4. Quality model for assessing technical documentation quality.

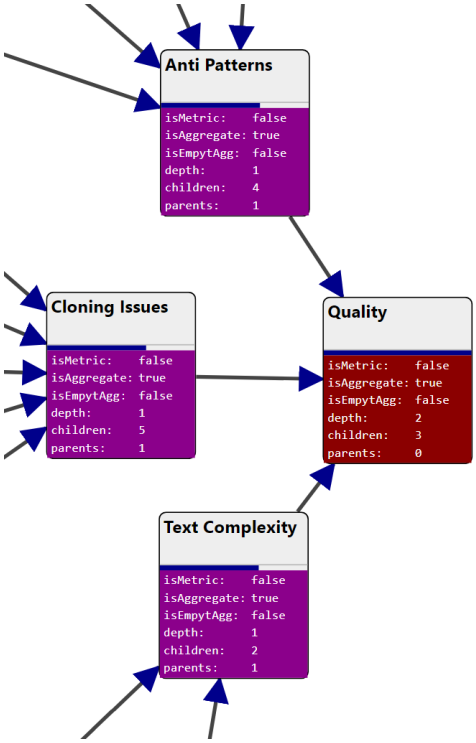

Fig. 5. The limited quality model of the technical documentation use case showing the three aggregation nodes in detail.

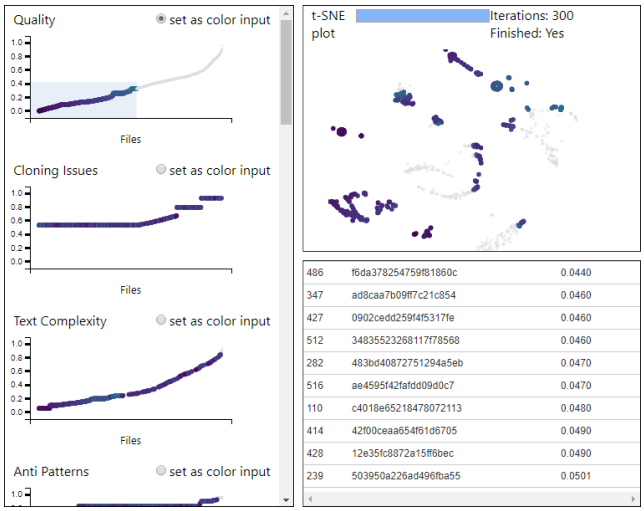

Fig. 6. The low-quality documents of the technical documentation.

This data set's single criterion or score, that all its files are made comparable with, is *Quality* (cf. Figure 9). Note that this use case does not use the intermediate criteria *Maintainability*, *Size* and *Complexity* and its Quality is based on all available metrics directly. The distribution of the empirical CDF for it is rather monotone and linear, only a single spike can be observed, that applies to about 5% of the files. For various runs of t-SNE, we obtain about six to nine clusters, with the Maintainability being spread out almost equally over each. The generated clusters by the t-SNE seem to be rather stable, which speaks in favor of our tool's repeatability and well-chosen parameters for *perplexity* and number of iterations.

When investigating various metrics, it can be observed that none of them is directly correlated to Lines of Code (LOC), which shows an almost perfect linear distribution. That is an interesting finding, as it means that LOC is rather insignificant for any quality assessment in this project. As shown in Figure 7, when selecting entities of Number of Methods (NOM), instances are almost evenly distributed across the whole range of LOC.

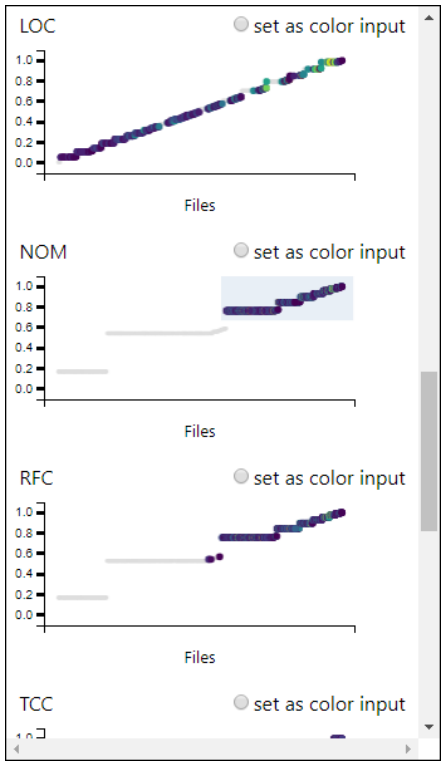

Fig. 7. Almost linear distribution for metric LOC.

We also find that a range of Response For a Class (RFC) is selected. This suggests that the two metrics are connected, which is not surprising in this case, since RFC includes NOM. This use case was therefore more suitable to unveil shortcomings in the underlying quality model, rather than finding files with good or bad quality. The definition of RFC alone is not able to substantiate low understandability. The finding here rather is that RFC and NOM are very similar in terms of their distribution. We can hence conclude that this default model, as given by VizzMaintenance, is not suitable to describe the maintainability of software. By manually engineering another quality model based on the available data, we have revalidated the same software against it in the next section.

C. Open Source Software (engineered quality model)

By reusing the data from the previous use case, which is based on a snapshot of JUnit, we have composed a second quality model to validate it against and is shown in Figure 9. The improved model uses the top-level node Quality, that is an

aggregation of *Size (LEN, LOC, NOC, NOM, WMC)*, *Complexity (CBO, CYC_Classes, DAC, DIT, MPC* and *Maintainability (ILCOM, LCOM, LD, LOD_Class, TCC)*. This engineered model provided much clearer insights into the project and its structure. Through various runs of the t-SNE, five clusters could be determined stably. About 42% of the files with low maintainability are now found in almost only two clusters. Those files also happen to have a lower than average quality, as was found by sub-selecting instances in the quality-plot.

By aggregating the data into a re-engineered quality model, we were able to detect clusters and to identify instances much clearer. Note that some of the metrics are now aggregated to more than one node, such as the Number of Children (NOC), which is an indicator for both *Size* and *Complexity*.

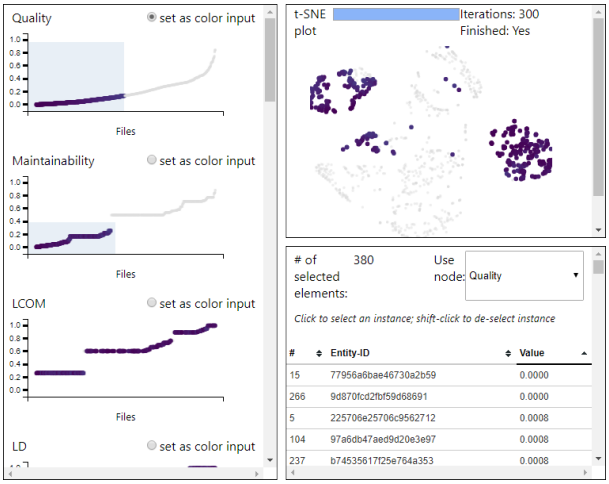

Fig. 8. Clusters of the manually engineered quality model for use case C, showing five clearly discernible clusters and identifying instances with low quality and low maintainability.

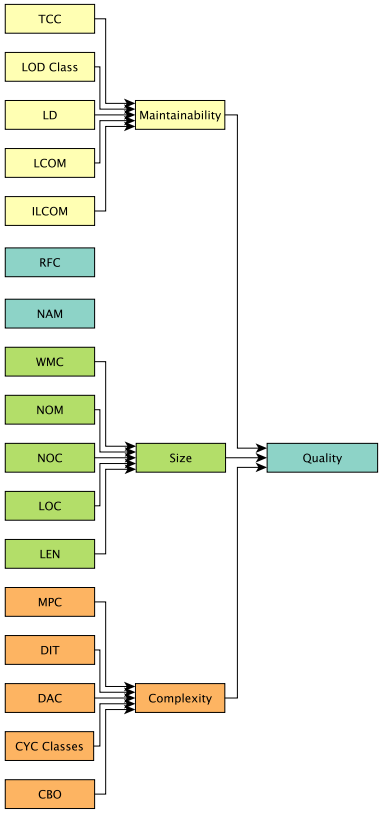

Fig. 9. Engineered quality model for assessing software quality using set of metrics proposed by VizzMaintenance.

VII. Discussion and Conclusion

In this paper, we demonstrate the practicality and usefulness of using joint probabilities and visualization for the quality assessment of software using metrics. Our tool was designed to support and help in understanding both marginal and joint distributions of metrics values and visualizing the similarities between software artifacts based on these distributions. The interactive metric plots (and the t-SNE view) allow the user to inspect and understand the probability distributions of each individual metric, and the probability calculations of the aggregate metrics are directly affected by the organization of the model as the users interact with the visualization. We illustrate the usability of the tool with two case studies of real-world examples: a set of technical documents and an open source project written in Java.

Our main goal was to improve upon classical aggregation techniques and to allow the users to freely explore the composition of the quality model and its effects on the software artifacts. This is different than the more classical data-oriented exploration, i.e., combining the metrics before the visualization, which might work when there is an objective and specific analysis goal and the method for combining the metrics is predefined and strict. Our proposal, on the other hand, gives the user a flexible framework to interactively test "what if" hypotheses when it comes to metric aggregation.

Taking advantage of the fact that the tool allows the users to make changes in the quality model, we consider as one possible avenue for future work to make it possible to compare different quality models. The underlying structure of the systems being assessed is also currently not part of the analysis. Complex software systems are usually organized in, for example, classes, packages, and namespaces. Using our method to find clusters of entities with similar quality assessments, these can be compared to the system's underlying structure to reason about where the quality problems are located, and whether the organization is adequate or there is room for improvement.

Acknowledgments

Part of the research presented in this paper is funded by the Knowledge foundation project "Software technology for self-adaptive systems" (ref. number 20150088). Some of the real-world datasets were contributed by Ericsson, so we acknowledge their involvement and support in this research.

References

[1] N. Fenton and J. Bieman, *Software metrics: a rigorous and practical approach*. CRC Press, 2014.

[2] J. Stasko, *Software visualization: Programming as a multimedia experience*. MIT press, 1998.

[3] A. Tversky and D. Kahneman, "Judgment under uncertainty: Heuristics and biases," *science*, vol. 185, no. 4157, pp. 1124–1131, 1974.

[4] J. H. Friedman and J. W. Tukey, "A projection pursuit algorithm for exploratory data analysis," *IEEE Transactions on computers*, vol. 100, no. 9, pp. 881–890, 1974.

[5] L. Van Der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[6] M. Ulan, S. Hönel, R. M. Martins, M. Ericsson, W. Löwe, A. Wingkvist, and A. Kerren, "Artifact: Quality Models Inside Out: Interactive Visualization of Software Metrics by Means of Joint Probabilities," Jul. 2018. [Online]. Available: https://doi.org/10.5281/zenodo.1311601

[7] S. Wagner, A. Goeb, L. Heinemann, M. Kls, C. Lampasona, K. Lochmann, A. Mayr, R. Plsch, A. Seidl, J. Streit, and A. Trendowicz, "Operationalised product quality models and assessment: The quamoco approach," *Information and Software Technology*, vol. 62, pp. 101 – 123, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584915000452

[8] O. Franco-Bedoya, D. Ameller, D. Costal, and X. Franch, "Queso a quality model for open source software ecosystems," in *2014 9th International Conference on Software Engineering and Applications (ICSOFT-EA)*, Aug 2014, pp. 209–221.

[9] S. Martínez-Fernández, A. Jedlitschka, L. Guzmán, and A. M. Vollmer, "A quality model for actionable analytics in rapid software development," *arXiv preprint arXiv:1803.09670*, 2018.

[10] L. Merino, M. Ghafari, and O. Nierstrasz, "Towards actionable visualisation in software development," in *2016 IEEE Working Conference on Software Visualization (VISSOFT)*, Oct 2016, pp. 61–70.

[11] L. Merino, M. Lungu, and O. Nierstrasz, "Explora: A visualisation tool for metric analysis of software corpora," in *2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT)*, Sept 2015, pp. 195–199.

[12] M. Termeer, C. F. J. Lange, A. Telea, and M. R. V. Chaudron, "Visual exploration of combined architectural and metric information," in *3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, 2005, pp. 1–6.

[13] M. Lungu, M. Lanza, T. Grba, and R. Robbes, "The small project observatory: Visualizing software ecosystems," *Science of Computer Programming*, vol. 75, no. 4, pp. 264 – 275, 2010, experimental Software and Toolkits (EST 3): A special issue of the Workshop on Academic Software Development Tools and Techniques (WASDeTT 2008). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167642309001221

[14] G. Langelier, H. Sahraoui, and P. Poulin, "Visualization-based analysis of quality for large-scale software systems," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*. ACM, 2005, pp. 214–223.

[15] D. Reniers, L. Voinea, O. Ersoy, and A. Telea, "The solid* toolset for software visual analytics of program structure and metrics comprehension: From research prototype to product," *Science of Computer Programming*, vol. 79, pp. 224 – 240, 2014, experimental Software and Toolkits (EST 4): A special issue of the Workshop on Academic Software Development Tools and Techniques (WASDeTT-3 2010). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016764231200086X

[16] U. Chotjaratwanich and C. Arpnikanondt, "A visualization technique for metrics-based hierarchical quality models," in *2012 19th Asia-Pacific Software Engineering Conference*, vol. 1, Dec 2012, pp. 733–736.

[17] ISO/IEC, "Iso/iec 25010 system and software quality models," Tech. Rep., 2010.

[18] M. O. Ward, G. Grinstein, and D. Keim, *Interactive data visualization: foundations, techniques, and applications*. AK Peters/CRC Press, 2015.

[19] L. Van Der Maaten, E. Postma, and J. Van den Herik, "Dimensionality reduction: a comparative," *J Mach Learn Res*, vol. 10, pp. 66–71, 2009.

[20] D. A. Keim, "Information visualization and visual data mining," *IEEE transactions on Visualization and Computer Graphics*, vol. 8, no. 1, pp. 1–8, 2002.

[21] J. I. Maletic, A. Marcus, and M. L. Collard, "A task oriented view of software visualization," in *Visualizing Software for Understanding and Analysis, 2002. Proceedings. First International Workshop on*. IEEE, 2002, pp. 32–40.

[22] A. Kerren and F. Schreiber, "Toward the role of interaction in visual analytics," in *Proceedings of the 2012 Winter, Simulation Conference (WSC)*. IEEE, 2012, pp. 1–13.

[23] W. Wright, D. Schroh, P. Proulx, A. Skaburskis, and B. Cort, "The sandbox for analysis: concepts and methods," in *Proceedings of the SIGCHI conference on Human Factors in computing systems*. ACM, 2006, pp. 801–810.

[24] W. S. Cleveland and R. McGill, "Graphical perception: Theory, experimentation, and application to the development of graphical methods," *Journal of the American statistical association*, vol. 79, no. 387, pp. 531–554, 1984.

[25] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *The Craft of Information Visualization*. Elsevier, 2003, pp. 364–371.

[26] M. Harrower and C. A. Brewer, "Colorbrewer.org: an online tool for selecting colour schemes for maps," *The Cartographic Journal*, vol. 40, no. 1, pp. 27–37, 2003.

[27] C. Ware, *Information visualization: perception for design*. Elsevier, 2012.

[28] M. Wattenberg, F. Viégas, and I. Johnson, "How to use t-sne effectively," *Distill*, vol. 1, no. 10, p. e2, 2016.

[29] B. Shneiderman, "Dynamic queries for visual information seeking," *IEEE software*, vol. 11, no. 6, pp. 70–77, 1994.

[30] P. E. Black, "big-o notation," *Dictionary of Algorithms and Data Structures*, vol. 2007, 2007.

[31] J. McCall, P. Richards, and G. Walters, *Factors in software quality: Final report*. General Electric Company, 1977.

[32] M. Ericsson, A. Wingkvist, and W. Löwe, "The design and implementation of a software infrastructure for iq assessment," *International Journal of Information Quality*, vol. 3, no. 1, pp. 49–70, 2012.

[33] R. Lincke and W. Löwe, "Compendium of software quality standards and metrics – version 1.0," http://www.arisa.se/compendium/, Tech. Rep., 2007.