Växjö University

School of Mathematics and Systems Engineering

Reports from MSI - Rapporter från MSI

Dynamic and Static Approaches for Glyph-Based Visualization of Software Metrics

Raja Majid Mehmood

Dec
2008

MSI
Växjö University
SE-351 95 VÄXJÖ

Report 08120
ISSN 1650-2647
ISRN VXU/MSI/DA/E/--08120/--SE

Abstract

This project presents the research on software visualization techniques. We will introduce the concepts of software visualization, software metrics and our proposed visualization techniques: Static Visualization (glyphs object with static texture) and Dynamic Visualization (glyphs object with moving object). Our intent to study the existing visualization techniques for visualization of software metrics and then proposed the new visualization approach that is more time efficient and easy to perceive by viewer. In this project, we focus on the practical aspects of visualization of multivariate dataset. This project also gives an implementation of proposed visualization techniques of software metrics. In this research based work, we have to compare practically the proposed visualization approaches. We will discuss the software development life cycle of our proposed visualization system, and we will also describe the complete software implementation of implemented software.

Keywords: Software Visualization, Software Metrics, Glyphs, 2D Glyph-Based Grid, SWT, Static Visualization, Dynamic Visualization, Texture, Animated Software Metrics, Multivariate Dataset, Data Object.

Acknowledgement

First of all I am very thankful to my supervisor Dr. Andreas Kerren for introducing me to software visualization and for all the help along the way. I have a sincere thanks to Mr. Ilir Jusufi for the many discussions, advice and proof-readings. Many thanks are also directed to my teacher Dr. Jonas Lundberg for great collaboration and many interesting discussions during my Master degree in Software Technology. Thanks as well to all of my colleagues and friends at Vaxjo University whose support through all the ups and downs has been so very much appreciated. Finally, I would like to thank my family for always supporting and encouraging me to make my studies possible.

Table of Contents

List of Figures

List of Tables

1. Introduction

Systems can contain a big amount of software entities linked together by different kinds of dependencies. Software visualization tools are used by software designers to raise the level of abstraction and reduce the amount of information to the one needed. Mostly these are stand-alone programs, which force the user to switch between different windows and contexts. The development of software visualization frameworks is a significant step to bring visualization tools in the forward engineering process [1], [2] and [3].

1.1. Background

This section describes the background of our research. It includes a brief explanation of software visualization, software metrics, why we use visualization in computer science or other fields, and multivariate dataset.

1.1.1. Software Visualization

Software Visualization is concerned, among other things, with static or dynamic (animated) 2D or 3D [1] visual representations of information about software systems based on their structure, size, or behavior. Often, the information used for visualization is software metric data from measurement activities. Visualization is inherently not a method for software quality assurance but can be used to manually discover anomalies similar to the process of visual data mining [35]. The actual objectives of software visualizations are to support the understanding of software systems (i.e., its structure) and algorithms (e.g., by animating the behavior of sorting algorithms) [37] as well as the analysis of software systems and their anomalies (e.g., by showing classes with height coupling) [7], [35], [36]. Software Visualization uses graphical representation to show code, control flows, classes, data and dependencies among them. It exploits the human visual perception system in order to provide what's lacking in a sequential, text-based representation. Therefore the programmer and software designer would have software visualization instead of lines of code displayed in a textual editor [3].

1.1.2. Software Metrics

A software metric is a measure of some property of a piece of software or its specifications. Since quantitative methods have been proved so powerful in other sciences, computer science practitioners and theoreticians have worked hard to bring similar approaches to software development. Tom DeMarco stated, "You can't control what you can't measure" [8].

There are many examples of software metrics: lines of code, cyclomatic complexity, function point analysis, bugs per line of code, code coverage, number of lines of customer requirements, number of classes and interfaces, Robert Cecil Martin's software package metrics cohesion or coupling [8].

Management methodologies, such as the Capability Maturity Model or ISO 9000, have therefore focused more on process metrics which assist in monitoring and controlling the processes that produce the software [8].

Industrial experience suggests that the design of metrics will help certain kinds of behavior from the people being measured. The common phrase applied is "you get what you measure" (or "be careful what you wish for") [2]. A simple example that is actually quite common is the cost-per-function-point metric applied in some software process improvement programs as an indicator of productivity [2]. The simplest way to achieve a lower cost-per-FP is to make a function points arbitrarily smaller. Since there is no standard way of measuring function points, the metric is wide open to gaming – that is, cheating [2].

One school of thought on metrics design suggests that metrics communicate the real intention behind the goal, and that people should do exactly what the metric tells them to do. This is a spin-off of test driven development, where developers are encouraged to write the code specifically to pass the test. If that is the wrong code, then they wrote the wrong test. In the metrics design process, gaming is a useful tool to test metrics and helps make them more robust, as well as for helping teams to more clearly and effectively articulate their real goals [8].

One way to avoid the "be careful what you wish for" trap is to apply a suite of metrics that balance out each other. In software projects, it's advisable to have at least one metric for each of the following: schedule, size/complexity, cost, and quality [8].

1.1.3. Why Visualization Can Help

Visualization is an area that presents data in a visual form to facilitate rapid, effective, and meaningful analysis and interpretation. Example application domains include scientific simulations, land and satellite weather information, geographic information systems, and molecular biology [4]. Visualization is also used in more abstract settings, for example, software engineering, data mining, and network security. A key challenge is designing visualizations that are effective for the user's data and analysis tasks [9].

The Software Visualization approach constructs visual representations that harness the strengths of the low-level human visual system. These perceptual visualizations display the data in ways that allow items of interest to capture the user's focus of attention [4].

It is a very critical situation to analyze the large size of the average dataset. The desire to extract knowledge efficiently motivates the need for an effective visualization system. A dataset's size is made up of three related properties:

- Number of elements stored in the dataset
- Number of attributes represented within the dataset
- Range of values possible for each attribute

Visualization tools are used to visualize these properties with effective representation (easy to perceive by viewer). These properties are represented in visualization known as information content and it is a combination of following properties:

- Number of elements
- Number of attribute values per element
- Range of different attribute values being visualized

The new technique described in this project seeks to increase information content by focusing on the last two properties, dimensionality and range [6].

1.1.4. Multivariate Dataset

In this thesis, a multivariate data object, d, is defined as d = {*d1, d2, . . . ,* d_N}, where d_i is a scalar and N is the number of attributes (N ≥ 2). A multivariate dataset [47] is then one comprising M data objects (instances of d), where M ≥ 2. The values of M and N vary widely depending on the application area. An illustration of a multivariate dataset containing shows five data items in table 1.1.

Maintainability	CMM_1_0_CBO	CMM_1_0_CYC_Classes	CMM_1_0_DAC
0.049088	3	1	3
0.17815	1	1	1
0.322364	1	1	2
0.117407	1	1	1
0.117407	1	1	1

Table 1.1. Multivariate Dataset for software components

The objects here are software components where each is represented by a number of attributes (CMM_1_0_CBO, CMM_1_0_CYC_Classes, CMM_1_0_DA) which describe different properties of the five software components. Here multivariate dataset is represented in tabular way, where a data object (or simply an item) corresponds to a row and each attribute corresponds to a column, is common and the one used throughout this thesis.

1.2. Problem Description

A lot of data is produced due to the rapid advances in computer technology. Mainly, the data comes from statistical data gathering, automated measurements and simulations, and this will become the primary source for a large dataset, which contains a large number of items, variables and time steps. It is very difficult to display a huge number of attributes in a small graphical representation. Our intent is to construct a software metrics that is useful to display as much as possible attributes within small drawing canvas with high quality display and time efficiency.

In this thesis, we have to visualize the multivariate dataset, which contains either too many data objects or attributes for a visualization to be efficiently carried out using a propose techniques, is referred to as being large. Depending on the application area, efficient can have different meanings. In general, representations that enable efficient information visualization are those which convey information about data in a clear and interpretable way, and in a reasonable time. Therefore, we need to develop an effective (time efficient, low memory consumption, accuracy) approach to visualize software metrics for a large multivariate dataset.

1.3. Goals

This project aims to the development of visualization techniques for software metrics that allows viewers to visually analyze, explore, and compare a multivariate dataset. We will introduce a technique that visualizes the data along a traditional two dimensional space-filling grid. We use "glyphs" (simple graphical objects) [39] that vary in color, placement, texture and motion properties to represent the attribute values contained in a data element. When shown together, the glyphs form visual patterns that support exploration, facilitate discovery of data characteristics, and highlight trends and exceptions**.** In the following, we identify four important goals for our research:

- To design graphical glyphs that support flexibility in their placement and in their ability to represent multivariate datasets.
- To implement a visualization technique that is user friendly and time efficient for a large multivariate dataset.
- To display the attributes values using static texture and animation.
- To focus on practical comparison for both approaches (static and dynamic) of visualization.

Our intent is to display the intersection of multiple datasets, and also show the specific details about the structure of data objects. This information can be critical to understand how the original data objects are related to each other. To our best knowledge, the combination of multivariate display techniques, perception, and animation for direct comparison of different perspectives into a dataset is a useful and novel contribution to the field of software visualization.

Our primary objective in this project is to design a graphical glyph that supports flexibility in its placement and in its ability to represent multivariate data objects. We focus to accommodate a high number of attributes by a small graphical representation and also try to display complete dataset in effective way.

The main part of this project is to develop a software tool that can accept any type of multivariate data and represent it graphically. Our software is able to read the dataset from some external data source; it can also be a excel file.

1.4. Outline

Chapter 2 describes previous research work related to visualization, software visualization and visualization of software metrics. This chapter also includes our proposed techniques for the visualization of software metrics. In Chapter 3, we describe the research and development methodology has been used in this project, also describe the analysis and design of the proposed system; it includes function and nonfunctional requirements, use cases, and class diagrams of our visualization system. In Chapter 4, we describe our implemented visualization techniques. This chapter also includes complete description of static and dynamic visualization approaches in our software. It contains detailed overview of our software interaction and utilities too. Chapter 5 is a comparison of both static and dynamic visualization approaches. In Chapter 6, we describe our conclusion and future work related to this research work.

2. Visualization and Proposed Techniques

In this chapter, we will discuss relevant research related to our problem. We will present some information about the literature covering the different visualization techniques in general. Mainly, we have used a book of Colin Ware [40] that covers the many topics of information visualization (texture, glyph, animation). We will also propose our own approach to visualize the software metrics.

2.1. Related Work

We will discuss previous work in short that is directly related to our proposed techniques, and highlight areas where we may possibly able to offer improvements over existing methods.

2.1.1. Textures

Textures are used in many disciplines, like computer vision, human visual psychophysics, and computer graphics. Though, each filed has different types of problems:

- texture segmentation and classification in computer vision.
- modeling the low-level human visual system in psychophysics.
- information display in computer graphics.

All of these groups need proper methods for the texture patterns being classified, modeled, or displayed. A review of texture segmentation and feature extraction techniques [12] describes two general classes of texture representation:

- statistical models use convolution filters other techniques to measure variance, inertia, entropy, or energy, and
- perceptual models that identify underlying perceptual texture dimensions like contrast, size, regularity, and directionality.

Here, texture studies focus on the perceptual features that make up a texture pattern. In our project, we express that we can use texture properties to assist in visualization, producing displays that allow users to fastly and correctly explore their data by analyzing the resulting texture patterns.

Different methods have been used to identify and investigate the perceptual features inherent in a texture pattern. Julesz has conducted several psychophysical experiments to study how a texture's first, second, and third-order statistics affect discrimination in the low-level visual system [13], [14]. This led to the texton theory [15], which proposes that early vision detects three types of features:

- elongated blobs with specific visual properties (like: hue, orientation, length).
- ends of line segments, and

- crossing of line segments.

Now, we will discuss several methods in computer graphics used to construct different textures. These texture patterns can perform different types of visualization tasks, like

- representation of flow patterns
- identifying spatially coherent regions in multi dimensional data
- representing surface shape and extent, and
- visualization of multivariate dataset.

During this research, I have studied few texture base visualization techniques [16], [17], [18], [19], [20], [21], [22] and [23] that are helpful to construct the texture for our proposed visualization system. Schweitzer used rotated discs to display the shape and orientation of a 3D surface [16]. Grinstein et al. developed a visualization tool called EXVIS that uses "stick-men" glyphs to produce texture patterns that display spatial coherence in a multivariate dataset [17]. Ware and Knight used Gabor filters to construct texture patterns; attributes of current dataset are used to modify the size, orientation, and contrast of the Gabor elements during visualization process [18]. Turk and Banks described an iterated technique to place streamlines for visualization of two-dimensional vector fields [19]. Interrante visualized texture strokes to show three-dimensional shape and layered transparent surfaces show the depth [20]. Salisbury et al. used texturing methods to build up computer-generated pen-and-ink drawings that express a realistic sense of shape, depth, and orientation [21]. Finally, Laidlaw described two methods for visualizing a two-dimensional diffuse tensor image with seven different values at each spatial location [22]. The first method used ellipsoids to show individual tensor values. The second method used oil painting [23] technique to visualize all seven values simultaneously.

After research on these existing texture based visualizations, we concluded a static texture in geometry object for our proposed static visualization. Although our method produces results that are glyph-like in appearance, but we differ from existing methods, both in how we construct our patterns, and in the end result that we generate. We introduce a texture pattern by collection of pixels, and this texture is placed within glyph object (geometry shape). The placement of pixels within texture pattern can be,

- random texture
- organized texture,
- random texture with placement by relative dataset value
- organized texture with placement by relative dataset value

We introduced the two types of textures: one is random and other is organized. Random texture is placed anywhere in glyph object, but organized texture is

placed in glyph object with proper sequence. Our proposed texture pattern is used to represents the strength of an attribute value. Dense texture pattern shows the high attribute value and sparse texture pattern shows the low attribute value. You can find more explanation about our proposed texture patterns in Chapter 4.

2.1.2. Motion

Motion is a very useful visualization technique, like the animation of objects, dye, or glyphs to represent the direction and magnitude of a vector field (e.g., fluid flow visualization). As with other visualization properties like (color, shape, texture), our objective is to use the property of motion to get an effective (easy to perceive by viewer) visualization display. There are three common properties described in the perceptual literature:

- flicker
- direction of motion, and
- velocity of motion.

Flicker is used as a visualization technique to perceive the discrete on-off pattern of some object by the viewer. It is normally measured as the frequency of repetition F in cycles per second (cps). A common use of flicker research in computer graphics is the critical flicker frequency (CFF), the rate at which images must be redrawn to appear continuous [28], [29].

Direction and velocity of motion are also used to create an attractive (understandable) visualization display. Tynan and Sekuler describe that the viewers responded rapidly to view a target object in the periphery (border or side-line) by 200-350 milliseconds, and 200 to 310 milliseconds for targets in the center of focus [30].

After study of previous research work [28], [29], [30] and [31], we propose the dynamic visualization that show moving object with constant speed in specific direction. As concern in our visualization of software metrics, we focus to the last two properties of motion that is direction and velocity of a moving object.

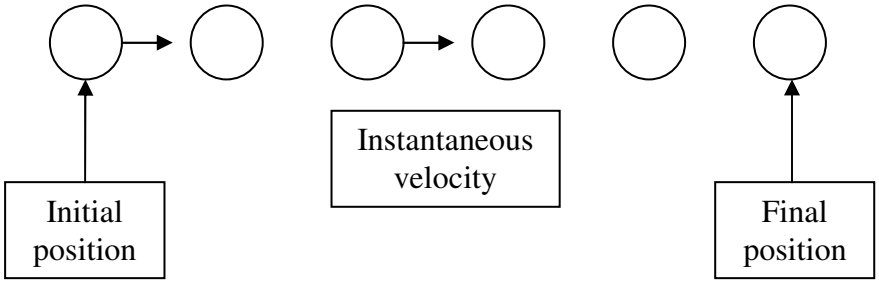


Figure 2.1. Motion of circle in Glyphs

We can see in Figure 2.1, where circle (geometry object) is moving with constant speed in one direction, either left-right or right-left. Instantaneous

velocity shows the strength of attribute, high instantaneous velocity shows high value of attribute and low instantaneous velocity shows low value of attribute.

2.1.3. Glyphs

Glyphs are graphical objects used to visualize the data of any type. A glyph may have properties of orientation, scaling, translation, deformation, size, placement, and color, etc. These properties are used to represents the input data. Glyphs are influenced by attributes of the current dataset. Here, we will discuss several glyphs based visualization techniques.

Chernoff developed iconic representation of the human face [24], [25]. The face elements like nose, eyes, eyebrows, mouth, and jowls are changed according to input data values. Foley and Ribarsky have created a visualization tool called *Glyphmaker* that is used to create visualization of multivariate datasets [26]. This tool uses a glyph editor and a glyph binder to create glyphs, to arrange them spatially, and to bind attributes to their visual properties. Levkowitz represented a visualization of colored squares to produce patterns to represent multivariate datasets [27].

Glyphs are also suitable to visualize software metrics. These glyphs can be used to improve the visualization for a larger dataset, for example, scaling shows the attribute strength, or coloring to identify attribute. After analyzing different types of glyphs, we proposed two types of glyphs in this project, one with a static texture and another with moving object, and they are used to visualize the data of double type. These proposed glyphs will more describe in the following section.

2.2. Proposed Visualization Approaches

In this research thesis, we have studied many existing software visualization techniques to display a multivariate dataset. These visualization tools have used a different type of visualization features like, texture, animation, and flow visualization in 2D or 3D environment. Mainly I have chosen the latest software visualization tools like [32], [33], [34] and [38] have visualized the multivariate dataset. After research on existing visualization tools we have found some deficiencies:

Display Complete Dataset

As we found, almost all the visualization tools have displayed a complete dataset in single display. If the dataset contains few data objects than it is fine and it can be useful to visualize the dataset. But on other hand, this type of visualization is not suitable for large dataset, where number of data objects in thousands and millions. In our research, we have proposed a sliding technique to visualize the complete dataset, where dataset is divided in multiple slides and each slide can show set of data objects. User can view any slide in constant time O (1), and it shows the time efficiency of the software visualization tool.

Display Complete Data Object
Similarly, we have found that existing visualization tools have displayed a complete data element or data objects in single appearance. This visualization technique is only worth full for low number of attributes. Our proposed technique enables to visualize the any number of attributes of data object; here we also used a slider technique to visualize the attributes of data object, where each slide contains the set of attributes. We have also introduced a scrolling option to visualize the attributes of data object, where user can view an attribute forward or backward by pressing the scroll bar. Here user also access any attribute of data object in constant time O (1).

Texture
As we discussed in above section 2.1.1, that many computer graphics researchers introduced different type of texture based techniques [16], [17], [18], [19], [20], [21], [22] and [23]. The existing tools are commonly used the texture having the properties of height, density, orientation, regularity, and placement. We analyzed that the texture pattern with many properties become more complex to perceive by viewer, and in these visualization techniques viewers have a difficulty to perceive the texture patterns. As we studied these techniques but we differ from existing methods, both in how we construct our patterns, and in the end result that we generate. Although, we have developed texture pattern contains only two properties: density and placement, and it is already discussed in section 2.1.1 with more details.

Animation
We studied many existing tools to visualize the software metrics. But there is only visualization tool [32] exist with property of animation. As we found that the animation is very complex to perceive in existing tools. Our propose animation to visualize the software metrics is very simple to perceive and easy to understand. There is a very smooth motion of object with constant speed and direction to show the strength of an attribute.

3D Environment
We analyzed 3D visualization [41], [42] is more complex and also expensive to render the resulting image or display. We concluded to construct the visualization tool for display of software metrics in 2D (two dimensional) environment.

Here, we will describe two different approaches for the visualization of software metrics: one is static and another is dynamic.

2.2.1. Static Approach

In this case, we use static objects (glyphs).

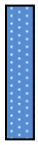


Figure 2.2. Static glyph or Vertical bar

Above, Figure 2.2 shows the static glyph with a texture pattern. This shows the static display of software metrics. We can see the following graphical properties of glyph object (Figure 2.2):

- shape (rectangle),
- background color,
- border, and
- texture pattern

In the graphical representation (Figure 2.3), we will use a proposed software metrics to create a sketch of complete multivariate dataset in single representation. We have a 2D grid that contains a number of 2D boxes, where each rectangular box in 2D grid represents a single data object. Each box contains a different number of vertical bars (glyph) with different color, texture and placement properties, where each glyph shows the attribute of current data object.

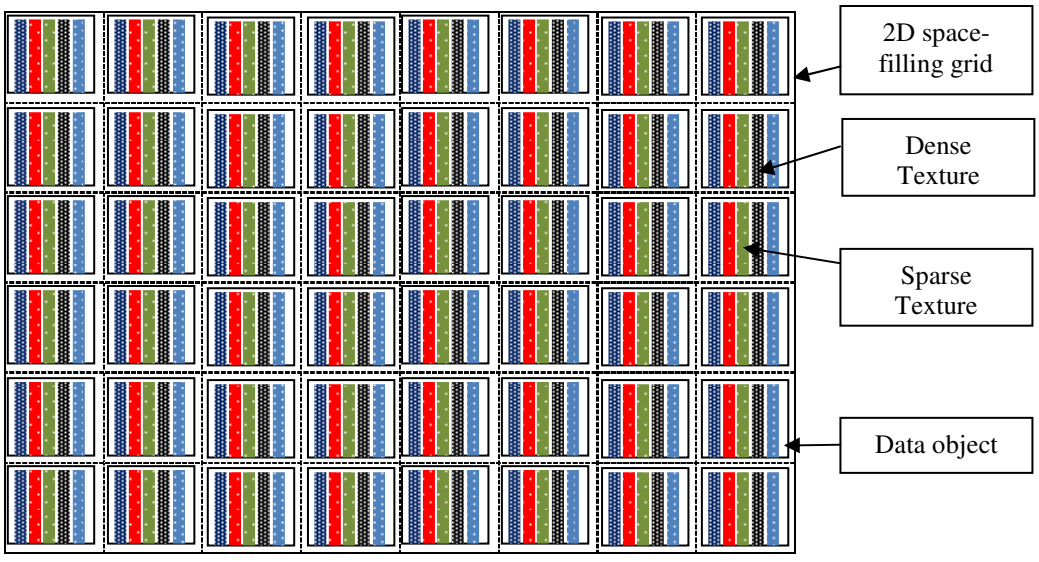


Figure 2.3. Graphical representation in 2D-Grid using static approach

In above Figure 2.3, we focus on three properties of glyph object; color represents the attribute itself. Here, texture represented by number of dots and it shows a representation of an attribute value, in above Figure 2.3, a dense texture (high number of dots) shows high value of that attribute, similarly in same Figure 2.3 sparse texture (low number of dots) shows the low value of that attribute, we will discuss the property of texture in Section 4.1.1.b in more detail. The placement of the bar identifies the sequence of attribute in each data object.

2.2.2. Dynamic Approach

In this approach, we use glyphs (geometry objects) that have properties of color, placement, and animation. Here, we use the technique of moving objects, which allows the motion of circle in glyph object. The main focus of our approach is to animate the circle in glyph object with a specific speed (animation). The speed of a moving object represents value of an attribute.

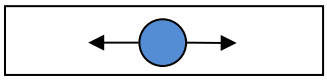


Figure 2.4. Dynamic glyph with moving circle

Above, Figure 2.4 shows the dynamic glyph with a moving object. This shows the dynamic display of software metrics. We can see the following graphical properties of dynamic glyph object (Figure 2.4):

- shape (Rectangle),
- background color,
- moving object, and
- direction, either left or right

In the following graphical representation (Figure 2.5), we will use the proposed dynamic software metrics (Figure 2.4) to visualize the complete multivariate dataset. We have a 2D grid that contains a number of 2D boxes, where each rectangular box represents a single data object. Each box contains a different number of horizontal bars (glyph) with different color, placement and animation (moving circle) properties, here each glyph is showing an attribute of current data object. In the following (Figure 2.5), we will focus on three properties of glyphs:

- color represents the attribute,
- animation shows attribute value, circle moves from left to right shows the positive value of attribute and similarly if circle moves from right to left shows the negative value of attribute, and
- placement of circles show the organized arrangement of attributes for each data object

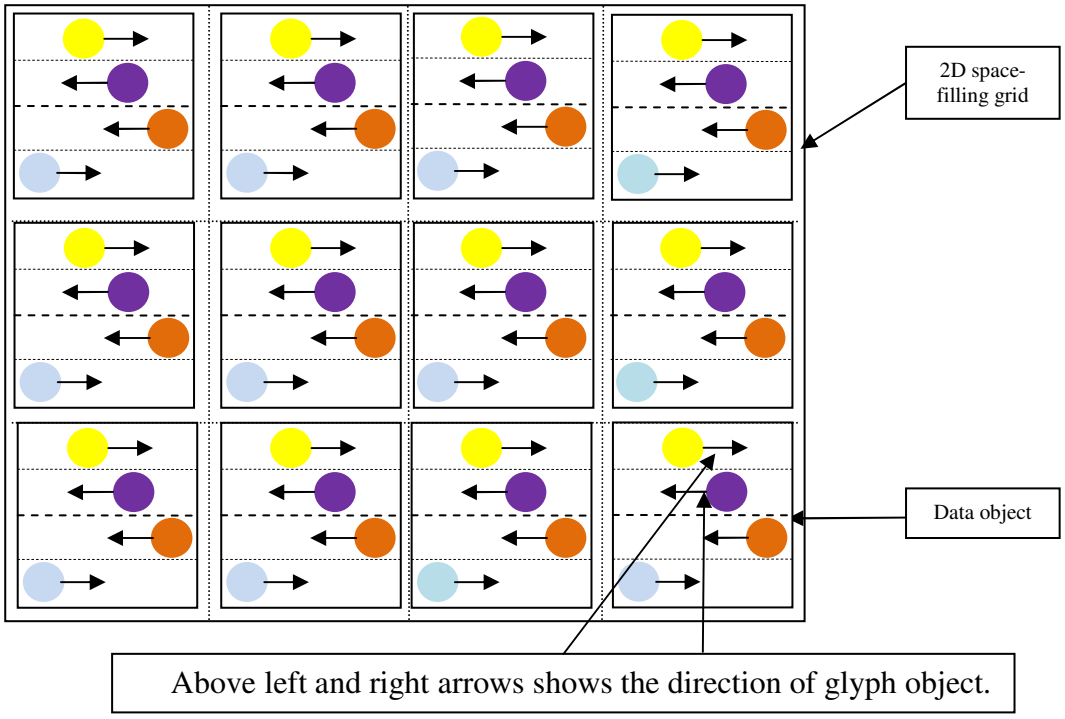


Figure 2.5. Graphical representation using dynamic approach

2.3. Summary

In this chapter, we studied different visualization techniques by different researchers. After detailed research of many visualization techniques of software metrics for multivariate dataset, we finalized our proposed techniques for visualization of software metrics in two-dimensional grid. At this point our idea is clear and now we are going to develop a proposed visualization tool for multivariate dataset. The next step is to analyze the requirements of proposed system and then we will design the proposed visualization tool. We will describe the requirement analysis and design in next chapter. That chapter also explains the software development methodology that will use in this project. Furthermore, the implementation of our proposed techniques will more describe in chapter 4.

3. Requirement Analysis and Design

This chapter includes the development methodology and requirements of our proposed system. Furthermore, we will describe the interaction between user and system through the *Use Case* diagram.

3.1. Methodology

In this section, we will describe the software development methodology which we have used in our research project. There are many software development methodologies. From them, we have chosen the Iterative and Incremental Development Methodology to develop software of high quality.

3.1.1. Iterative and Incremental Process Model

In an iterative and incremental lifecycle (Figure 3.1), the development proceeds as a series of iterations that evolve into the final system. Each iteration consists of the following process components: planning, requirements, analysis & design, implementation, testing and evaluation. The developers do not assume that all requirements are known at the beginning of the lifecycle; indeed change is anticipated throughout all phases [48].

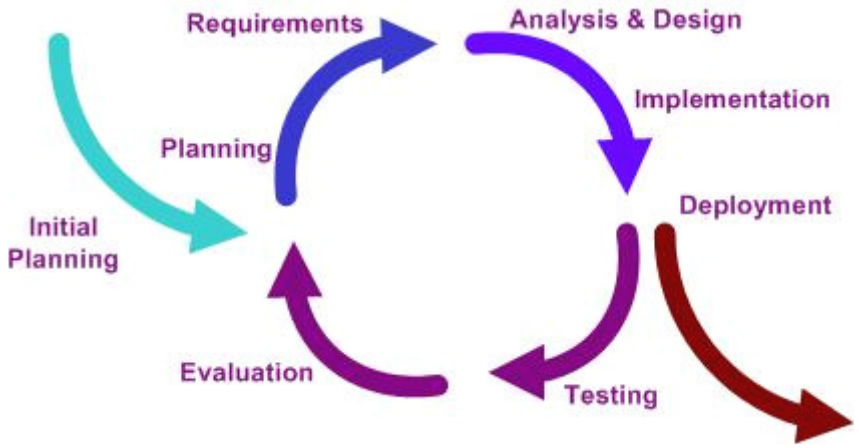


Figure 3.1. System Development Life cycle [53]

The Rational Objectory Process defines the control for an iterative and incremental lifecycle. It defines an extensive set of guidelines that address the technical aspects of software development focusing on requirements analysis and design [10]. This process is structured along two dimensions:

- time and,
- process

The Time dimension is structured as follow:

- **Inception:** It is a specification of the project vision. In this phase, we have to focus on the actual problem, requirements gathering, and research work.
- **Elaboration:** Planning the necessary activities and required resources; specifying the features and designing the architecture.
- **Construction:** Building the product as a series of incremental iterations.
- **Transition: S**upplying the product to the user community (manufacturing, delivering, and training).

The Process dimension includes the following activities:

- **Project Management:** It includes planning of project development, development cases, description and guidelines.
- **Requirements:** Requirement specification of proposed system.
- **Analysis & design:** Analyze and describe the main functionalities of the proposed system, and also design the system for implementation phase.
- **Implementation:** Code generation and programming that will result in an executable system.
- **Test:** Verification of the entire system.
- **Deployment:** Deliver to end user in an operating environment.

Each activity of the process component dimension is applied to the each phase of the time based dimension. Figure 3.2 shows how the process components are applied to each time based phase.

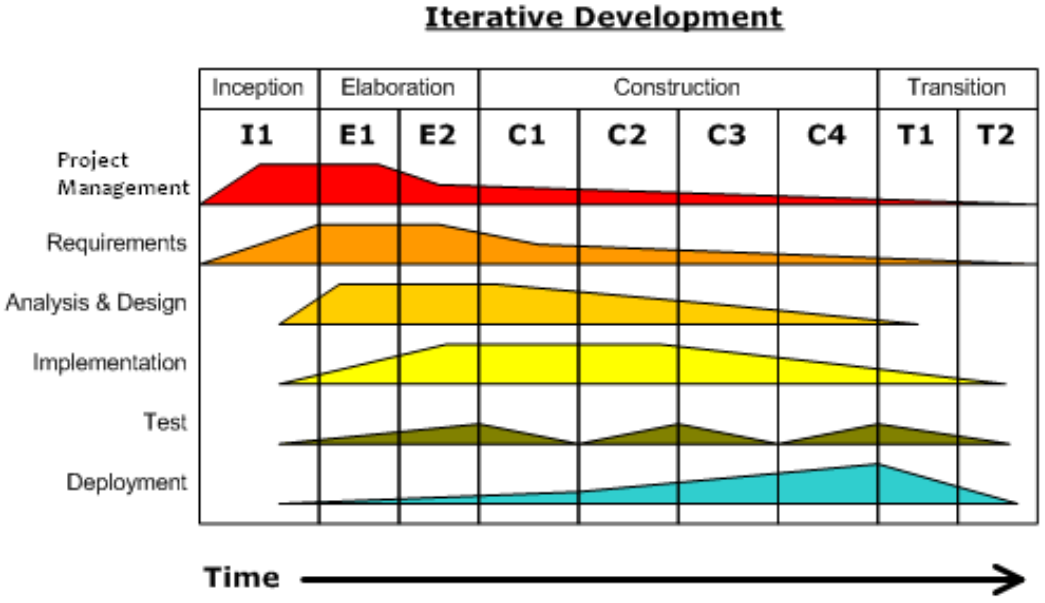


Figure 3.2. Overview of Iterative and Incremental Model [53]

As we described the life cycle and the Rational Objectory Process of the Iterative and Incremental process model, we used the same development methodology in our research project. Similar to the proposed model, we

- get the requirements at first,
- analyze the old visualization research of software metrics,
- design the architecture of proposed system,
- develop the code, and
- test the entire program

All above steps are described in this report in detail. In the next section, we will define the requirement specification and use cases, which help us further to design the proposed system.

3.2. Requirements

A requirement specification helps us to define the scope and boundaries of the proposed system. It leads to a better understanding of overall system. There are two types of requirements: functional requirements and non-functional requirements.

3.2.1. Functional Requirements

This subsection describes the functional requirements of proposed system. It defines the main functions or components of a software system, and each of them includes the required function of the proposed system, its importance, and a brief description.

FR-01	Define Data Source Name
Importance	Essential
Description	To setup a DSN (data source name) ODBC bridge for external data source (excel file)
FR-02	Load Dataset
Importance	Essential
Description	To load the dataset FR-01 some external data source (Excel sheet, through configured JDBC-ODBC Bridge). At the same time, it constructs internal data structure of same dataset for further use.
FR-03	Attribute Definition
Importance	Desirable

Description	The default definition of attribute is constructed during the loading process of dataset. But a user has facility to update or change the definition of attributes.
FR-04	Display Settings
Importance	Desirable
Description	This setting panel is setup in the start of main program. The user has access to change or update these settings for static and dynamic display of *2D-SFGrid*. She/he has options of scrolling and sliding the attributes in each data object for both static and dynamic display.
FR-05	Static Display
Importance	Essential
Description	This display gives us a static behavior of software metrics. In this display, each attribute is represented by glyph (geometry object, rectangle). Here, the texture on glyph shows the strength of attribute value, the glyph border shows the positive value of attribute, otherwise negative, the glyph background color represents the color of a specific attribute.
FR-06	Dynamic Display
Importance	Essential
Description	The user can get the dynamic display of software metrics. In this display, software metrics contain properties of background color and motion of a glyph object. The background color represents the specific attribute, the speed of moving object shows the strength of attribute value, and the direction of moving object represents the positive or negative value of attribute (left to right motion shows a positive value of an attribute, right to left motion shows a negative value of an attribute).
FR-07	Mouse Click on Glyphs
Importance	Desirable

Description	The user can also get the complete information of any cell in data object, just by clicking the right mouse button on desired attribute.
FR-08	Data Object Slider for Scrolling
Importance	Desirable
Description	This function enables the user to view all attribute values in each data object through slider bar. In this option, the user can view one attribute at same time to move the slider either forward or backward.
FR-09	Data Object Slider for Sliding
Importance	Desirable
Description	This option enables the user to view attributes in set of slides for each data object.
FR-10	View Different Number of Attributes
Importance	Desirable
Description	To view only selected attributes, user need to change the attribute definition, only select the desired attributes and save the attribute settings at the end.
FR-11	Attributes Comparison Slider
Importance	Desirable
Description	To compare the attribute values in currently displaying data objects.
FR-12	Data Set Slider
Importance	Desirable
Description	To view all data objects in dataset by scrolling the slider. This control displays all data objects in set of slides. By default, one slide may contain maximum 24 data objects.

Table 3.1. Functional requirements

3.2.2. Non-Functional Requirements

These requirements are often described as usability, reliability, performance, and substitutability requirements. We will discuss non-functional requirements in this section together with their importance and a brief description.

N-FR-01	Performance
Importance	Essential
Description	The system should work fast.
N-FR-02	Efficient Display of 2D-SFGrid
Importance	Essential
Description	To compare the attribute values in currently displaying data objects.
N-FR-03	Efficient Data structure
Importance	Essential
Description	Data structure must be efficient to get fast results.
N-FR-04	Application Maintenance
Importance	Essential
Description	Application can be maintainable, well documented.
N-FR-05	Application Scalability
Importance	Desirable
Description	The system should be scalable. It should be helpful for developers to add the new features and functionalities.
N-FR-06	Platform Independent
Importance	Desirable
Description	The system should work on all computer machines.

Table 3.2. Non-Functional requirements

3.3. Use Cases

In this section, we will describe the use cases of our proposed system. This technique is used in system development to retrieve the functional requirements of a system. Use cases describe the interaction between a primary actor (external user) and the system itself, represented as a sequence of simple steps. Actor may be end users, hardware devices or other systems. Each use case is a complete series of events, and it is described from reference of the actor [11].

UC-01	Configure DSN
Goal	To make a JDBC-ODBC connection
Pre-Condition	The system must have JDBC and ODBC drivers and also have data source (excel file).
Post-Condition	Connection established.
Triggered Event	The user needs to configure the Administrative tools.
Description	This help us to load the dataset from excel file.
UC-02	Execute the Application
Goal	To visualize the software metrics
Pre-Condition	JVM (Java Virtual Machine) must be installed.
Post-Condition	The *Main Controller* window will open.
Triggered Event	The user opens the executable file (.jar).
Description	The user runs the executable program and application will come up with *Main Controller* window, where user can control the application completely.
UC-03	Load Dataset
Goal	To load the dataset from external source and construct the data structure for further use.
Pre-Condition	DSN (JDBC-ODBC bridge) must be configured of external data source (excel file).

Post-Condition	All other application controls will be enabled.
Triggering Event	The user clicks the button (*Load Data Set*) in *Main Controller* window.
Description	This is very first step for the user to get visualization of a software metrics. This is performed only once in an entire program. When user clicks this button, he/she will get the complete dataset from external data source into the current running application. During this loading process program also constructs internal data structure for further use.
UC-04	Open the Attribute Definition Window
Goal	To view or change the attributes details.
Pre-Condition	Dataset must be loaded.
Post-Condition	The user can view the complete details of all attributes of dataset. Also he/she can change the color of any attribute. Here, the user can select or deselect the attributes in further visualization.
Triggering Event	Open the *Attribute Definition* window by clicking the mouse button in *Main Controller* window. View all the attributes by scrolling the slider bar by just pressing the mouse left button. Select or Deselect the attributes by pressing the mouse button on the check boxes.
Description	This window shows the complete detail of each attribute. It includes a color rectangle, RGB color code, name, maximum positive value, minimum positive value, maximum negative value, minimum negative value of each attribute of dataset.
UC-05	Open Static Display of Software Metrics
Goal	To view the software metrics with static texture.
Pre-Condition	Dataset must be loaded.
Post-Condition	*2D-SFGrid* window will open, that shows the software metrics with static texture.

Triggering Event	The user clicks the button (*Static Display*) in *Main Controller* window.
Description	This *2D-SFGrid* window shows the software metrics with static texture, which helps the user for analyzing the contents of dataset. Here, texture on the glyph shows the strength of an attribute value, the glyph border shows the positive value of an attribute, otherwise negative, and the glyph background color represents the color of a specific attribute.
UC-06	Open Dynamic Display of Software Metrics
Goal	View the software metrics with motion of glyph object.
Pre-Condition	Dataset must be loaded.
Post-Condition	The *2D-SFGrid* window will open, that shows the software metrics with moving glyph objects.
Triggering Event	The user clicks the button (*Dynamic Display*) in *Main Controller* window.
Description	The *2D-SFGrid* window shows the software metrics with dynamic behavior, which helps the user for analyzing the contents of dataset. In this display, software metrics contains properties of a background color and a motion of glyph object. Background color represents a specific attribute, speed of a moving object shows the strength of a attribute value, and direction of a moving object represents the positive or negative value of an attribute (left to right motion shows a positive value of an attribute, right to left motion shows a negative value of an attribute).
UC-07	Visualize the Attributes Detail
Goal	To get the information of specific software metrics.
Pre-Condition	2D space-filling grid must be open.
Post-Condition	The user can see the tooltip text.
Triggering Event	The user clicks the left mouse button on desired glyph object to view the detail of that attribute.

Description	Here, user can view the name, maximum positive value, minimum positive value, maximum negative value, minimum negative value, actual value and texture value (property of texture only use in static display of software metrics. High texture value shows that, the actual value is close to maximum value of this attribute and vice versa) or delay value (property of delay only use in dynamic display of software metrics. High delay value shows the actual value is close to minimum value of this attribute and vice versa) of an attribute.
UC-08	Sliding the Data Object
Goal	To view the all attributes values in data object.
Pre-Condition	Must have some attributes and data objects in dataset.
Post-Condition	The user can view different attributes values of same data object.
Triggering Event	Mouse left click event is triggered to move the data object slider.
Description	In this case, the user can view the set of glyph objects in each slide, and he/she can move the slider either forward (next set of attributes) or backward (previous set of attributes) direction to view either next set of attributes or previous set of attributes respectively.
UC-09	Scrolling the Data Object
Goal	To view the all attributes values in data object.
Pre-Condition	Must have some attributes and data objects in dataset.
Post-Condition	The user can view different attribute values of same data object.
Triggering Event	To view the all attributes values of any data object by scrolling the slider bar in *Data object Composite*.
Description	The user can view the set of glyphs in each slide, and he/she can move the slider either forward (next attribute) or backward (previous attribute) direction to view either next or previous attribute respectively.

UC-10	Sliding the Dataset
Goal	To view the all data objects in dataset.
Pre-Condition	Must have some data objects in dataset.
Post-Condition	The user can view the other data objects in dataset.
Triggering Event	To view the different data objects by scrolling the bottom slider bar in *2D-SFGrid* window.
Description	The user can view the set of data objects in each slide (each slide contains maximum of 24 data objects), and he/she can move the slider to either forward (next set of data objects) or backward (previous set of data objects) direction to view either next set of data objects or previous set of data objects respectively.
UC-11	Comparing the Values of Data Objects
Goal	To compare all the attributes values of currently displayed data objects.
Pre-Condition	Must have a normalized dataset.
Post-Condition	The user can visualize same attributes in currently displayed data objects.
Triggering Event	The mouse left button is clicked, and it changes the attributes in all displayed data objects.
Description	To compare all the attributes values of currently displayed data objects by scrolling the slider bar at the top of *2D-SFGrid* window.
UC-12	Mouse Over on Data Object Name
Goal	To view the full qualified name of data object.
Pre-Condition	Must have some data object.
Post-Condition	To see the full qualified name of this data object.
Triggering Event	Tooltip comes up on mouse over.

Description	The user can get the *Full Qualified Name* of any data object by just moving the mouse on the rectangle of *Data Object Name* in *Data Object Composite*.
UC-13	View or Update the Display Settings
Goal	To facilitate the user to change as he/she wants.
Pre-Condition	Dataset must be loaded.
Post-Condition	*Display Settings* window comes up.
Triggering Event	The user clicks the left mouse button on *Display Settings* button in *Main Controller* window.
Description	The user can view or change *2D-SFGrid* display settings in *Display Setting* panel. This panel contains two parts: one is *Static Display* and other is *Dynamic Display*.
UC-14	Update the Static Display Settings
Goal	To change the default display settings of *Static Display.*
Pre-Condition	Dataset must be loaded.
Post-Condition	Information message comes up to confirm these settings applied successfully.
Triggering Event	Selection events triggered, when user change the attributes display settings.
Description	Here, the user has few choices to select different types of texture techniques. He/she also has another property; he/she can select only one option of data object attributes display by either scrolling or sliding. The user can realize these changes in *Static Display*.
UC-15	Update the Dynamic Display Settings
Goal	Change the default display setting of *Dynamic Display*.
Pre-Condition	Dataset must be loaded.
Post-Condition	Information message comes up to confirm these

	settings applied successfully.
Triggering Event	Selection events triggered, when user change the attributes display settings.
Description	In this case, the user has a facility to change the display properties of *Dynamic Display* window. One property is *Attributes View Option* that allows the user to select by either scrolling or sliding. Other property is *Number of Attributes*; here user can select any number from option list, and this will affect the number of glyphs will display in each *Data Object Composite*. User can realize these changes in *Dynamic Display* window.

Table 3.3. Use Cases

3.4. Use Case Diagram

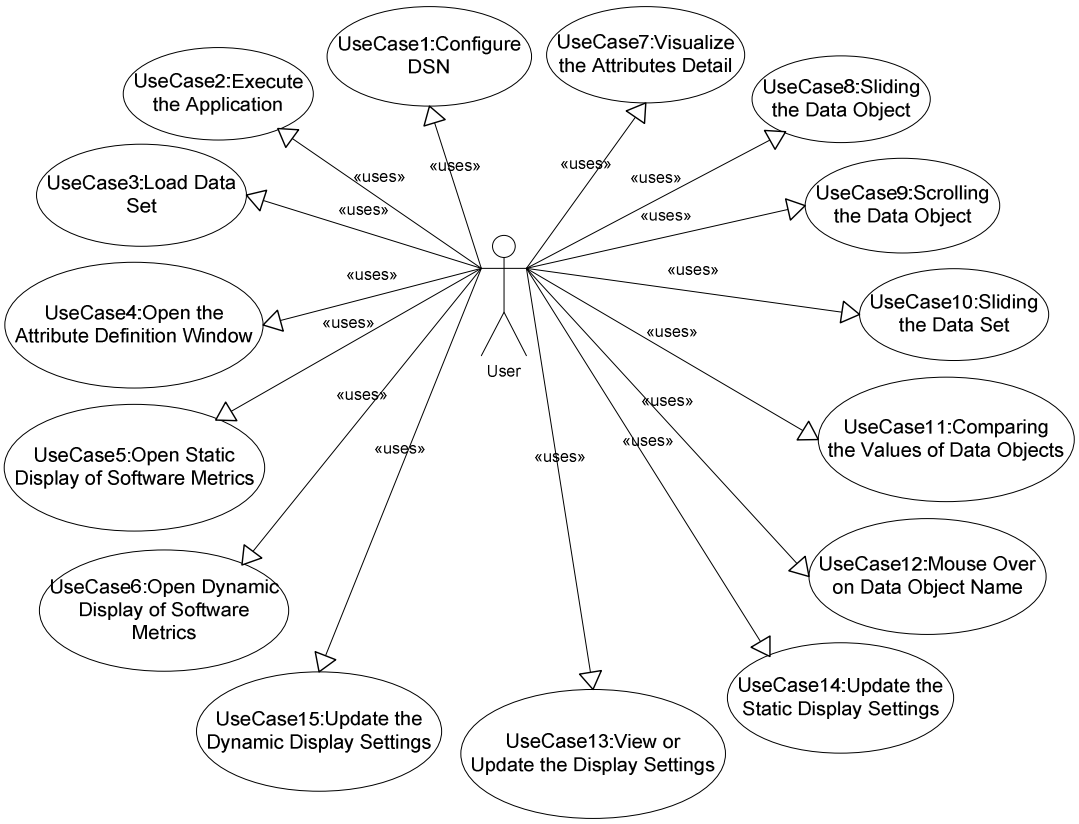


Figure 3.3. Use case for 2D-SFGrid Visualization System

3.5. Summary

In this chapter, we defined the functional and nonfunctional requirements which help us in the design of the use cases. We included use case diagram that shows the interaction of user with the system. In the next chapter, we will discuss the implemented visualization and interaction techniques in our proposed system.

4. Visualization and Interaction Techniques

In this chapter, we will describe the visualization techniques for software metrics implemented in our application (2D-SFGrid). As we proposed, two visualization techniques: one with static textures and another with moving object in glyphs. This chapter also includes explanations of interaction techniques used in this application.

4.1. Visualization Techniques

It is very difficult to accommodate a huge dataset in small graphical representation. Here, we have used a two-dimensional grid to visualize the data objects and their attributes.

4.1.1. Static Visualization of Software Metrics

In this visualization, we define a visual representation of software metrics with following graphical properties,

a) background Color,
b) texture,
c) border,
d) sequence of attributes,
e) placement of texture.

In the following screenshot (Figure 4.1), we see a data object "DefinitionTable" and their attributes. There are many glyph objects (inside rectangles); each glyph represents one attribute of the data object.

Figure 4.1. Screenshot: Data Object (Static Software Metrics)

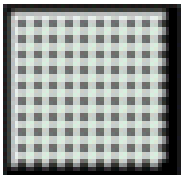
In Figure 4.1, each glyph object corresponds to one attribute with the properties of background color, texture, border, sequence of attributes, and placement of texture.

a) Background Color

This property identifies an attribute itself, and is used to differentiate the attributes in the data object. In the above Figure 4.1, we can see that the background color of all glyph objects represents the attributes of the data object. We also see few glyphs in Figure 4.1 with black background (without texture), which do not have the properties of proposed software metrics. These glyph objects show that there are no more attributes in the data object, and thus they are presented as a blank glyph objects.

b) Texture

This property is used to represent the value of an attribute. Perhaps, high density texture represents a high attribute value and a low density texture represents a low attribute value. Texture is calculated through scaling of actual attribute value. Figure 4.1 shows that many glyphs have the property of texture, where each glyph texture shows the scaled value of each attribute in that data object. The following Figures 4.2 presents different textures for the same attribute. Figure 4.2 (a) shows the actual attribute value is close to highest attribute value. And in Figure 4.2 (b), the actual attribute value is close to the lowest attribute value.



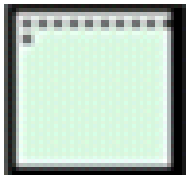


Figure 4.2. (a): High Texture (b): Low Texture

Texture Value?

We will use the following properties to calculate the texture value:

Ma => Max Attribute Value
Mi => Min Attribute Value
N => Number of Parts
A => Attribute Texture Value
Aa => Actual Attribute Value
I => Interval or Partial Differential

$I = (Ma - Mi) / N$
$A = (Aa - Mi) / I$ ..... *Formulas 4.1 (For Positive Values)*

$I = (|Mi| - |Ma|) / N$
$A = (|Mi| - |Aa|) / I$ ..... *Formulas 4.2 (For Negative Values)*

All attribute properties can be seen in Figure 4.3, where its actual value is 6.0. As the value is positive, we use Formula 4.1.

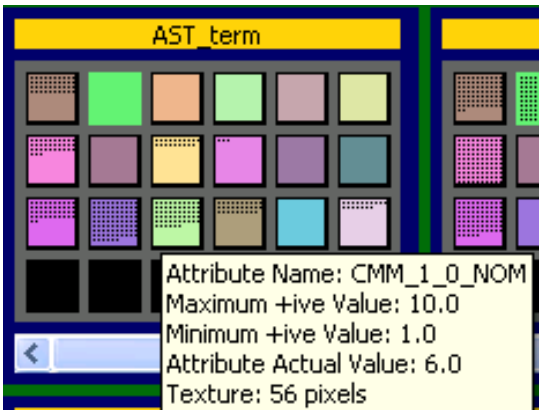


Figure 4.3. Screen Shot: Actual value with attribute information

Now, we will calculate the texture value,

A = attribute_texture_value=?

Consider the values from the above screenshot (Figure 4.3), where actual_attribute_value=6.0, and use them as input values for Formulas 4.1.

I = (10.0-1.0)/100 => 0.09
A = (6.0-1.0)/0.09
A = 55.5555555556 => 56

The calculated texture value is used to display the texture by 56 dots in the glyph rectangle.

c) Border

In Figure 4.1, we can see glyphs with a border property. The Border of a glyph object shows the positive value of attribute, and if attribute value is negative than the glyph object will be borderless. As we can see in Figure 4.1, there is one glyph object without border, which is presented in the following Figure 4.4 (a).

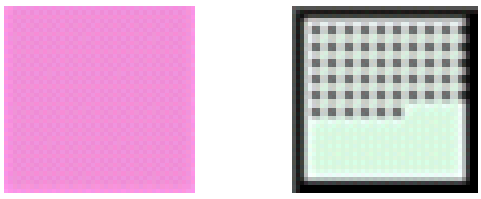


Figure 4.4. (a): Borderless Glyph (b): Glyph with Border

The above glyphs object (Figure 4.4 (b)) has a border, that present the positive value of attribute.

d) Sequence of Attributes

This property of software metrics enables to show the attributes in proper order. As we can see in the following Figure 4.3, where all attributes of both data objects have same placement sequence. This property helps to user to compare the same attribute in different data objects.

Figure 4.5. Screenshot: Sequence of Attributes

e) Placement of Texture

The position of a texture in a glyph represents the relative value of an attribute in the dataset. In the following, we are going to explain, how to place the texture (set of dots) in the glyph. As we can see in the screenshot above (Figure 4.4), the texture is placed on the same position in all glyphs.

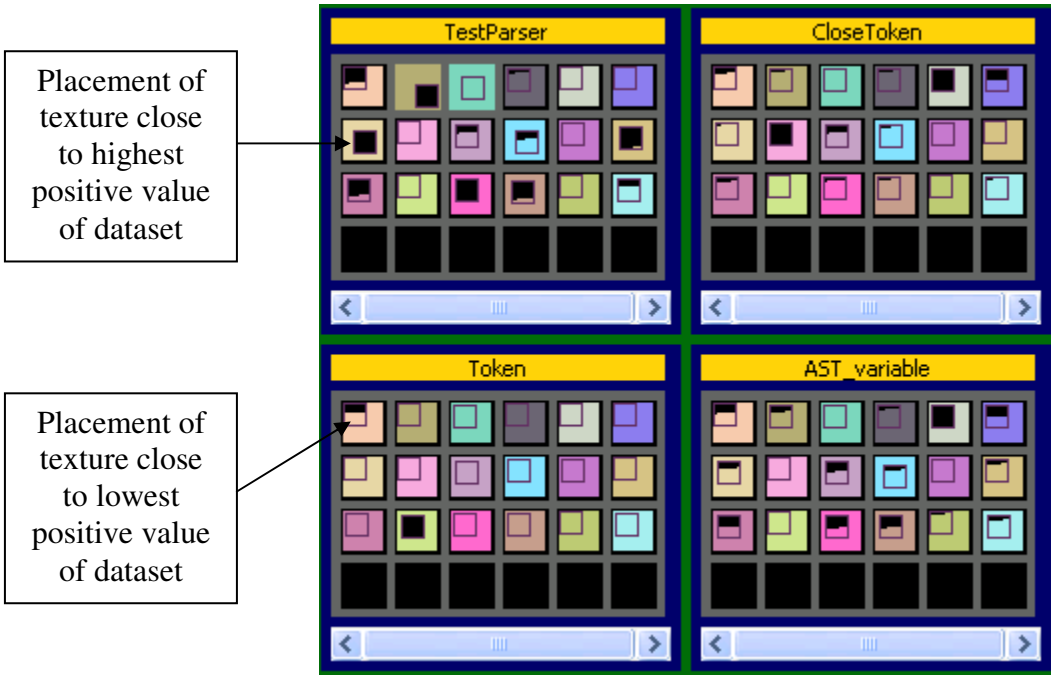


Figure 4.6. Screenshot: Placement of Texture in Glyphs

In Figure 4.6, however the placement of textures is different in all glyphs and it shows the position of attribute's value in dataset. Here, we will explain the placement of texture in more detail.

How to Find Out the Position for a Texture in Glyphs?

We will compute the position of a texture for glyph objects with the help of the following formulas:

$$I = (MaD - MiD) / N$$
$$Tp = (Aa - MiD) / I \qquad \text{..... } \textit{Formulas 4.3 (For Positive Values)}$$

$$I = (|MiD| - |MaD|) / N$$
$$Tp = (|MiD| - |Aa|) / I \qquad \text{..... } \textit{Formulas 4.4 (For Negative Values)}$$

In Formula 4.3 and 4.4, we have some variables like

- *MaD*, maximum value in dataset.
- *MiD*, minimum value in dataset.
- *N*, scaling size or difference of range i.e. 10 for positive value, and 11 for negative value.
- *Aa*, current value of attribute in current data object.
- *I,* is used for interval, or it can be a partial differential.
- *Tp,* is location of texture placement.

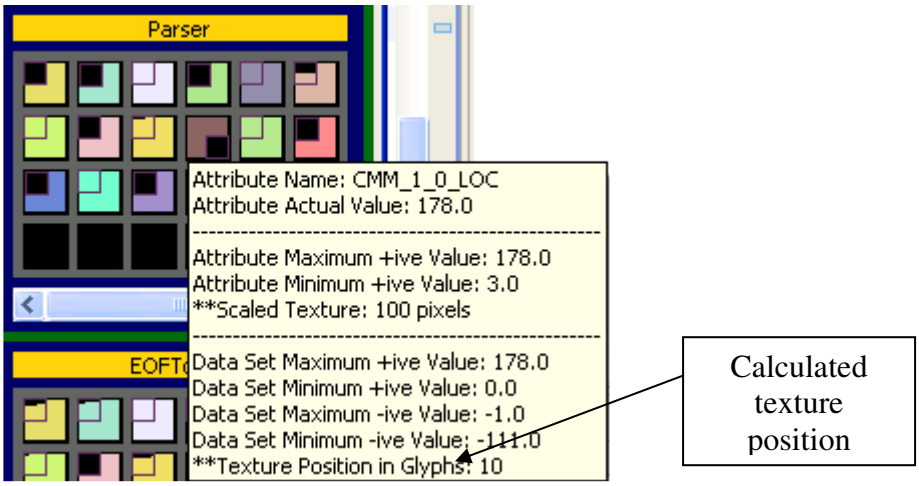


Figure 4.7. Screenshot: Calculated Texture Position in Glyphs

Here, we are only interested to calculate the texture position in glyph:

Tp = texture_placement=?

Consider the values from above screenshot (Figure 4.7), and compute the Tp with the help of Formula 4.3,

I = (178.0-0.0)/10 => 17.8
Tp = (178.0-0.0)/17.8
Tp = 10.0

Hence, we calculated texture position "Tp" is exactly same as in the above screenshot (Figure 4.7).

4.1.2. Dynamic Visualization of Software Metrics

In our second visualization technique, we define software metrics with the following properties:

a) Background Color
b) Animation
c) Sequence of Attributes

The following screenshot shows a data object "AST_term" and its attributes. As we can see, there are many glyph objects and each glyph contains one moving object (circle). As before, each glyph represents one attribute of the current data object.

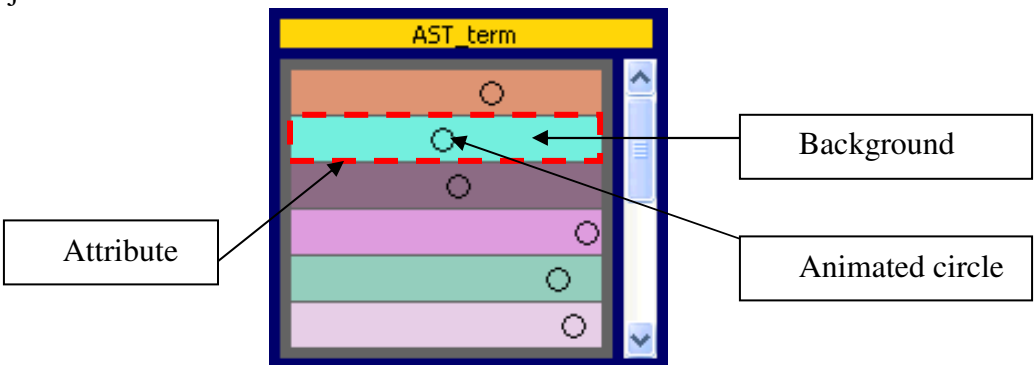


Figure 4.8. Screenshot: Software Metrics in a Dynamic Display

a) Background Color
This property identifies the attribute and is used to differentiate the attributes in a data object. In the above Figure 4.8, we can see that the background color of all glyph objects represents the attributes of a data object.

b) Animation

The moving object (circle) in each rectangular box (Figure 4.8) is used to generate the animation property of software metrics, and it represents the value of an attribute. In the following Figure 4.9, we have two rectangular boxes: each represents one attribute of the data object. The moving objects in these rectangles have a different direction. If the circle is moving from right to left (Figure 4.9 (a)), then this shows the negative value of the attribute. Similarly, we see in Figure 4.9 (b), where circle is moving from left to right that the value of the attribute is positive.

(a): R-L Motion of Circle (b): L-R Motion of Circle

Figure 4.9. Screenshot: Motion of Circle

The speed of moving object shows the actual value of an attribute, i.e., higher speed represents higher attribute value and vice versa.

Here, we need to calculate the delay value against the actual value of the attribute: The delay value shows the speed of a moving object, and it must be between 0-100 (delay value is used to control the multi threading operation, and default range (0-100) of delay is useful for smooth animation of objects). A high delay shows low speed of a moving object and a low delay shows high speed of a moving object. Now, I describe the formula to compute the delay value against the actual value of an attribute.

Delay Value?

$$I = (Ma - Mi) / N$$
$$Ad = N - (Aa - Mi) / I \quad \text{…….. } \textit{Formulas 4.5 (For Positive Value)}$$

$$I = (|Mi| - |Ma|) / N$$
$$Ad = N - (|Mi| - |Aa|) / I \quad \text{…….. } \textit{Formulas 4.6 (For Negative Value)}$$

In above Formula 4.5 and 4.6: we have different variables like,

- *Ma*, maximum value of this attribute in dataset
- *Mi*, maximum value of this attribute in dataset
- *N*, range size or difference of range, i.e., (0-100) => 100
- *Aa*, current value of attribute in current data object
- *I,* is used as a interval
- *Ad,* is a Attribute delay value for moving object.

In the following Figure 4.10, we can see the complete information of the attribute with its actual value.

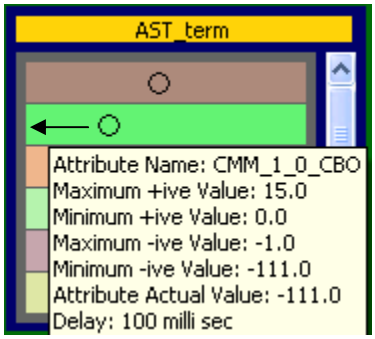


Figure 4.10. Screenshot: Get Delay Value

We see that the actual value is exactly equals to the lowest value of attribute. As we know the range of delay value is 0 to 100, we can imagine that the delay value should be 100 in this case. Let's go to compute stepwise:

Consider values from above screenshot (Figure 4.10), here Aa = -111.0; put required values into Formula 4.6.

I = (-1*(-111)-(-1)*-1)/100 => 1.1 => 1
Ad = 100-(-1*(-111)-(-111)*-1)/1;
Ad = 100-0 => 100

The above value 100 shows the delay time of a moving object from one state to another state. The delay value is different than the texture value, because this is used to control the speed of moving object, i.e., a high delay presents s low attribute value and a low delay value presents a high attribute value.

c) Sequence of Attributes

This property of software metrics enables to show the attributes in a proper order. As we can see in the following Figure 4.11, all attributes of both data objects have the same placement sequence. This property helps the user to compare the same attributes of different data objects.

Figure 4.11. Screenshot: Sequence of Attributes in Two Data Objects

4.2. User Interaction and System Dialogs

In this section, we will discuss possible interaction techniques for users; these techniques allow the user to analyze a large dataset and also facilitate him/her to visualize software metrics through static and dynamic display. We also describe few important controls of application in the following sections.

4.2.1. Main Controller Window

This is the main interface for users to operate with the complete application. As we can see, the *Load Data SET* button enables the user to enter the *DSN* (Data Source Name) and excel sheet number.

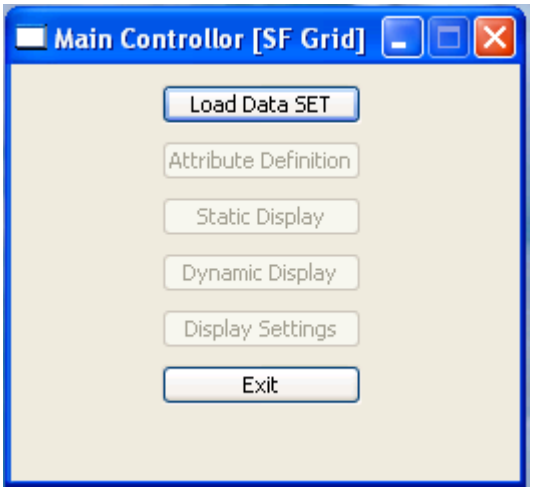


Figure 4.12. Screenshot: Main Window Before Loading of a Dataset

a) Excel File Format

We used a excel file system as an input source. Our application can only allow the following file format.

R1C1 | fx | FullQualifiedName

	1	2	3	4	5	6	7
1	FullQualifiedName	name	Maintainability	CMM_1_0	CMM_1_0	CMM_1_0	CMM_1_0
2	lambda.semantics.D	DefinitionT	0.049088167	3	1	3	0
3	lambda.parser.AST_	AST_term	0.178150281	-111	1	1	0
4	tests.TestParser	TestParser	0.322363911	-1	-55	2	0
5	lambda.scanner.Clo	CloseToke	0.117407103	1	1	1	1
6	lambda.scanner.Dot	DotToken	0.117407103	1	1	1	1
7	tests.TestScanner	TestScann	0.136637872	2	1	2	0
8	lambda.scanner.Lar	LambdaTo	0.117407103	1	1	1	1
9	lambda.utils.AST_E	AST_Even	0.136637872	0	1	0	0
10	lambda.scanner.Tok	Token	0.178150281	0	1	0	0
11	lambda.parser.AST_	AST_variat	0.185726039	4	1	2	1

Table 4.1. Excel File Format

Above Table 4.1 shows a multivariate dataset. Here, each cell in top row represents the name of each column. All other rows in this table represent the data objects of the dataset. First column is *Full Qualified Name* and second column is *Actual Name* of each data object respectively. After that all columns represent the attributes of that data object, and data type of attribute's value is *double.*

To proceed further, it is necessary to configure first, the DSN (Data Source Name) Bridge in the administrative tools of operating system.

b) DSN Input Dialog

Here, users have options to enter the desired *DSN* and excel sheet name, and then to proceed by pressing the *OK* button.

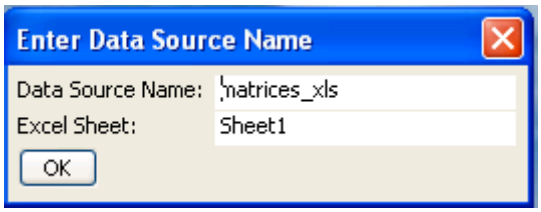


Figure 4.13. Screenshot: DSN Dialog

After pressing the *OK* button, we will get the following screenshot, where other controls are enabled, and now a user can access these controls easily.

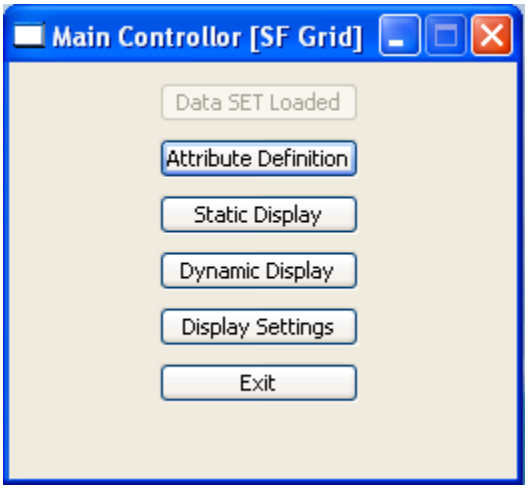


Figure 4.14. Screenshot: Main Controller After Loading of a Dataset

4.2.2. Attribute Definition Window

We can get the following visual interface (Figure 4.15) by pressing the *Attribute Definition* button (Figure 4.14). This window gives the complete information of all attributes of a dataset. The user has an option to choose only desired attributes by selecting the respective check boxes.

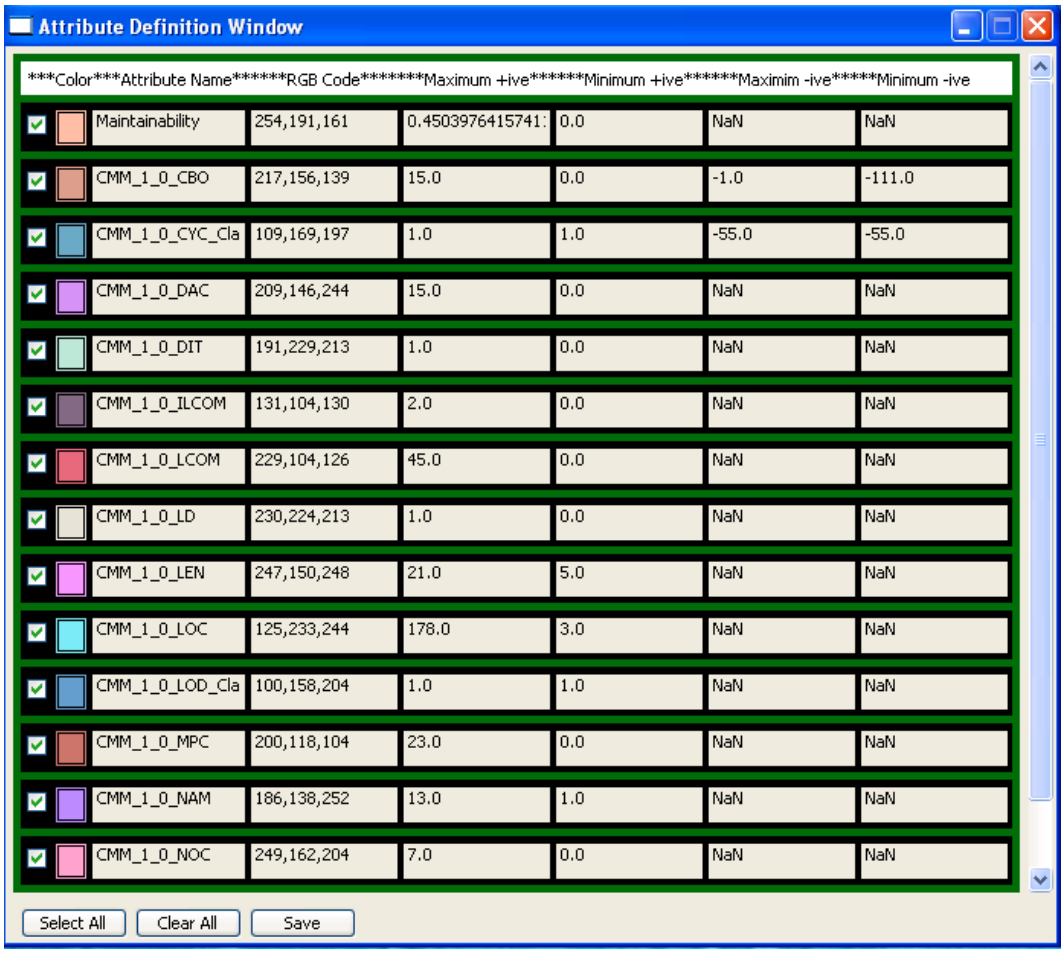


Figure 4.15. Screenshot: Attribute Definition Window

The user also has the option to select or deselect all visible attributes by pressing the *Select All or Clear All* buttons (Figure 4.15). In Figure 4.16, the user unselects all the attributes at once by pressing the *Clear All* button.

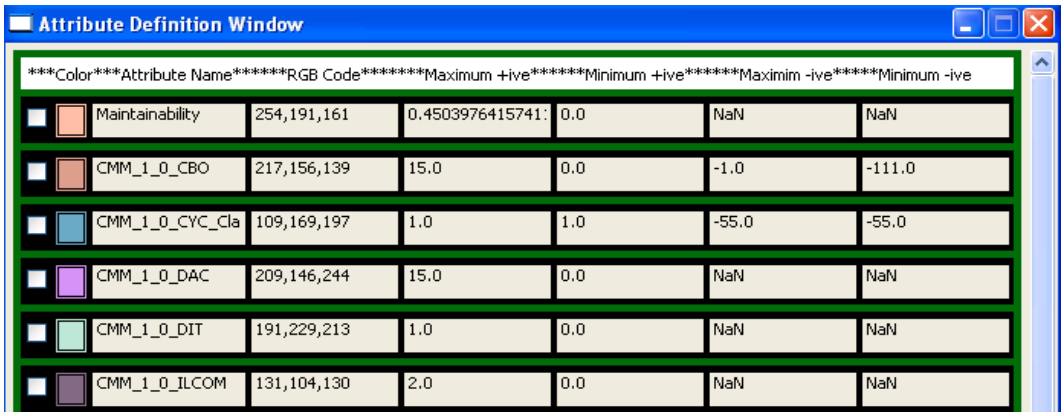


Figure 4.16. Screenshot: Showing Clear All Operation

If user wants to view same settings in the visualization window (*2D-SFGrid*), then he/she must save these changes by pressing the *Save* button. The user also has an option to change the color of an attribute by pressing the mouse button on the color of that attribute. If a user presses the button, the color dialog will come up, as we can see in the following Figure 4.17.

Figure 4.17. Screenshot: Change the Attribute Color

4.2.3. Static Visualization in 2D-SFGrid

The user can get static visualization of software metrics by pressing the *Static Display* button (Figure 4.14).

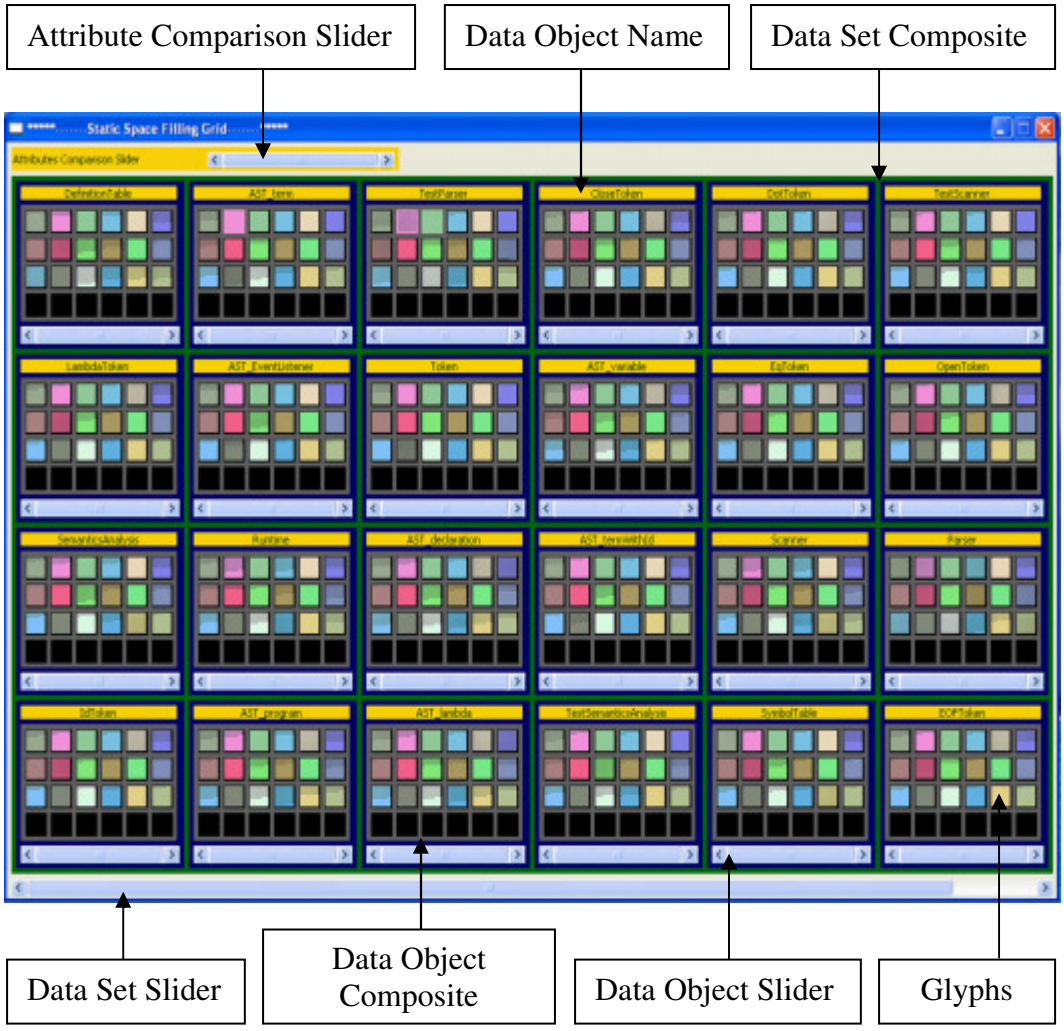


Figure 4.18. Screenshot: Static Display of Software Metrics

In this window (see Figure 4.18), the user has different types of controls for the visualization of software metrics. We will describe above Figure 4.18 as follows,

Controls	Description
Data Set Composite	2D Grid with green background (one in window).
Data Object Composite	Placed 24 Data Object Composites in one Data Set Composite, this grid has a grey background. Each represents one data object of a dataset.
Glyphs	Placed 24 glyphs in each data object composite.

Attribute Slider Bar	Placed at top of the window.
Data Object Slider Bar	Placed at bottom of each Data Object Composite (inner grid with grey background).
Data Set Slider Bar	Placed at bottom of the window.
Data Object Name	Place at top of each Data Object Composite, this is a Label control with yellow background.

Table 4.2. Components of Static 2D-SFGrid

4.2.4. Dynamic Visualization in 2D-SFGrid

The user can get the dynamic visualization of software metrics by pressing the *Dynamic Display* button (Figure 4.14).

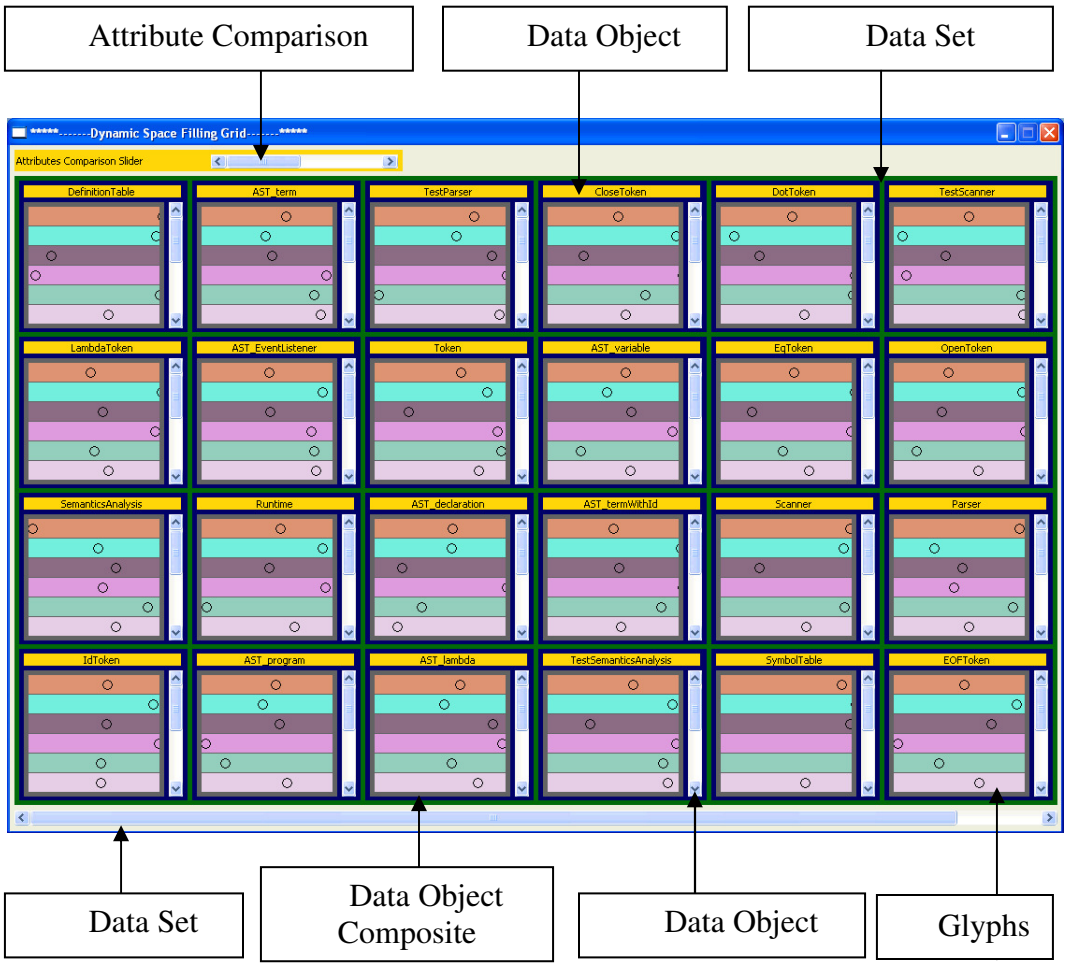


Figure 4.19. Screenshot: Dynamic Display of Software Metrics

As in above screenshot, we have different types of controls for the visualization. Here, we will describe these controls,

Controls	Description
Data Set Composite	2D Grid with green background (one in window).
Data Object Composite	Placed 24 Data Object Composites in one Data Set Composite, this grids have grey background.
Glyphs	Placed in each data object composite, by default each Data Object composite contains 6 Glyphs, user can change this setting in Display Setting Panel.
Attribute Slider Bar	Placed at top of the window.
Data Object Slider Bar	Placed vertically at right of each Data Object Composite (inner grid with grey background).
Data Set Slider Bar	Placed at bottom of the window.
Data Object Name	Place at top of each Data Object Composite, this is a Label control with yellow background.

Table 4.3. Components of Dynamic 2D-SFGrid

4.2.5. Display Settings

In our application, the user also has an option to change the default settings; here we will discuss these settings practically.

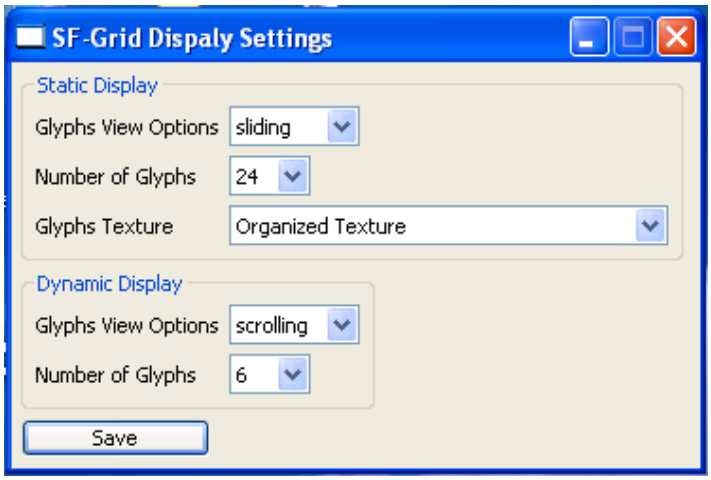


Figure 4.20. Screenshot: Default View for Both Visualizations

a) Sliding

By default, this option is active. User can get the sliding option to view the different attributes of same data object.

(a) (b)

Figure 4.21. Screenshot: Sliding View

Here, we can see in Figure 4.21 (a), that shows the first slide of attributes (max 24 in each slide). By pressing the slider bar, we can see in Figure 4.21 (b) the next slide will come with two more attributes. In Figure 4.21 (b), the slider bar shows the last slide (set of attributes), and there are no more attributes in this data object.

b) Scrolling

If user wants to view only one attribute forward or backward in same data object, then he/she needs to select the scrolling option from *SF-Grid Display Settings,* and save these settings by pressing the *Save* (Figure 4.22) button.

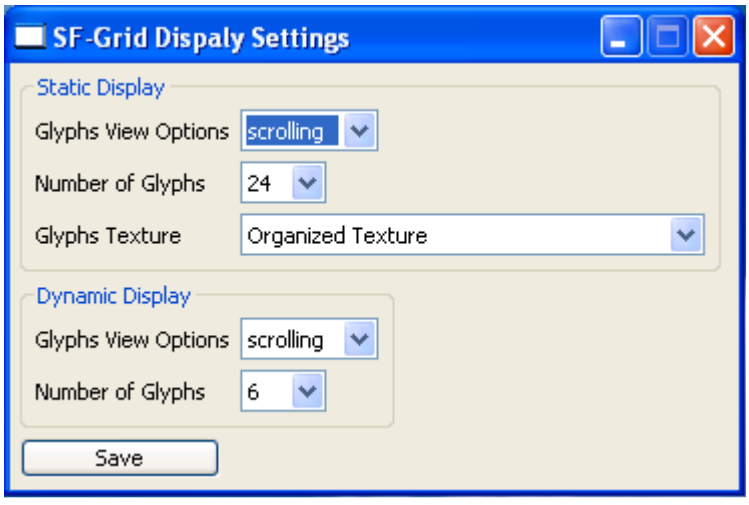


Figure 4.22. Screenshot: Set Scrolling Option for Static Display

Figure 4.23 (a) shows the initial state of scrolling. In the Figure 4.22 (b), we can see the next attribute by pressing the slider bar to the right hand side.

(a) (b)

Figure 4.23. Screenshot: View the Software Metrics in 2D-SFGrid

As we see in Figure 4.22 (b), a new attribute comes at the end (at 24th position in grid), and all old attributes are moved one step backward.

c) Glyphs in Data Object Composite

This option is only available in dynamic display of software metrics. It enables the user to view a desired number of *Glyph*s objects in *Data Object Composite*.

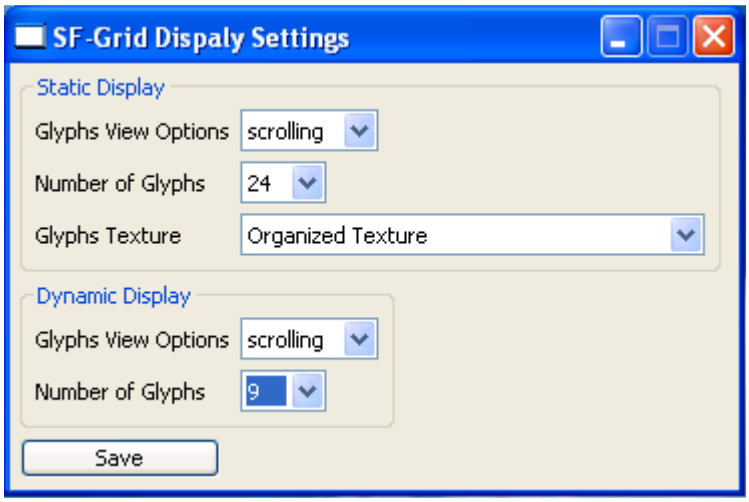


Figure 4.24. Screenshot: Select Number of Glyphs for Dynamic Display

In Figure 4.24, the user selects "9" for number of glyphs to display. In the following Figure 4.25, that each *Data Object Composite* shows the nine glyphs.

Figure 4.25. Screenshot: View the Effect in the Dynamic Display

d) Texture Choices

By selecting this option, the user can get the organized glyphs texture without the property of *Placement* (see section 4.1.1).

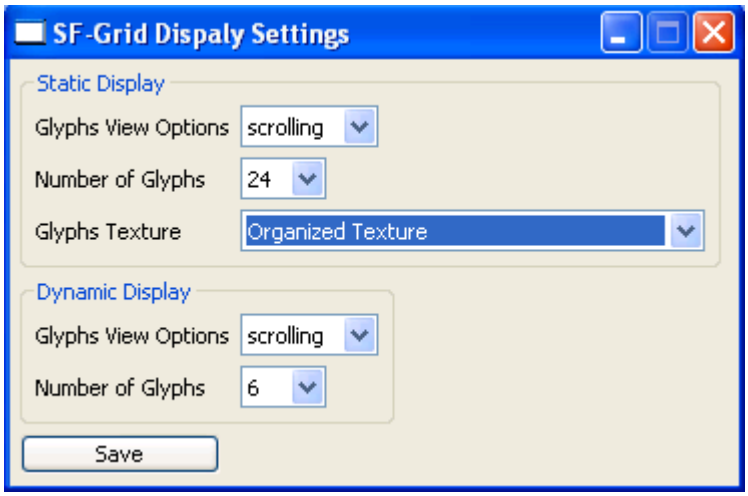


Figure 4.26. Screenshot: Change the Texture Property in Display Settings

In the following Figure 4.27, we see that the textures are organized with a fixed placement (all textures start from the zero location in the glyphs).

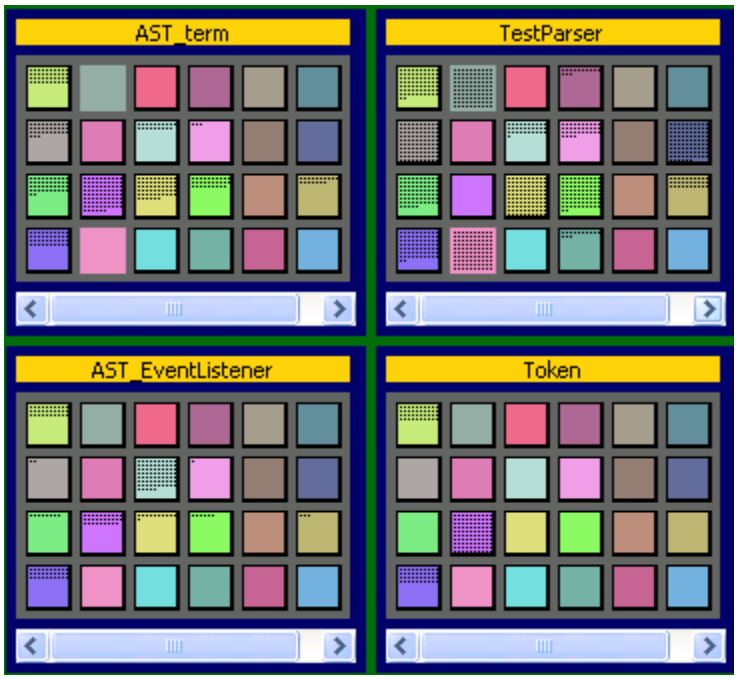


Figure 4.27. Screenshot: Get Organized Textures in the Static Display

We also have an option of a random texture without the property of *Placement*.

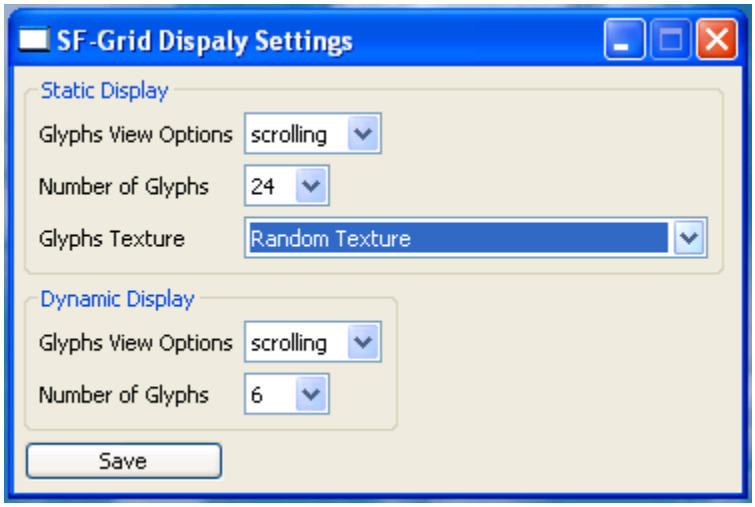


Figure 4.28. Screenshot: Change Texture Option in Display Setting

Figure 4.29 shows random textures in all glyphs objects. Both techniques give the same meanings in different ways.

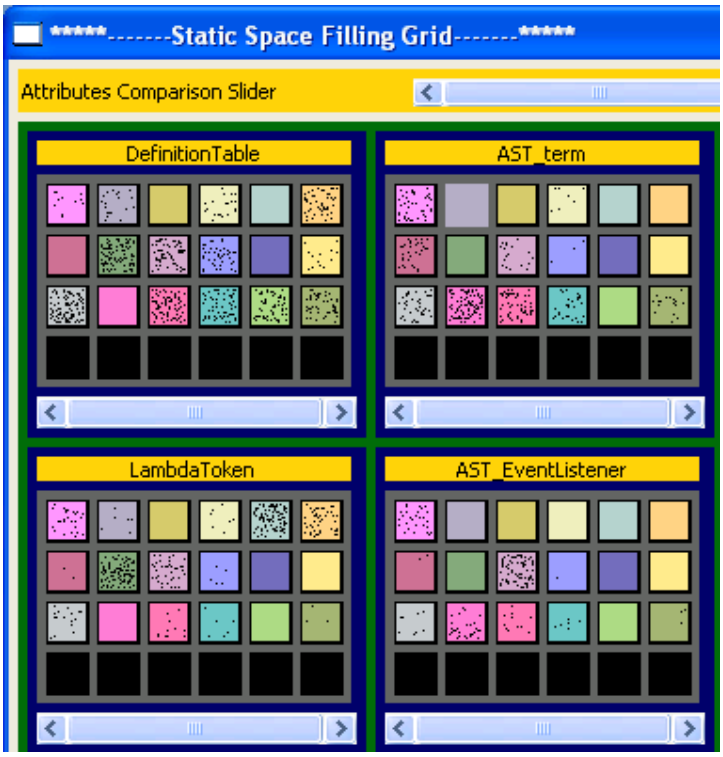


Figure 4.29. Screenshot: View Random Textures in the Static Display

If the user wants to view software metrics with the feature of *Placement*, then he/she needs to select the option *Organized Texture with Scaled Positioning* from the option list (Figure 4.30).

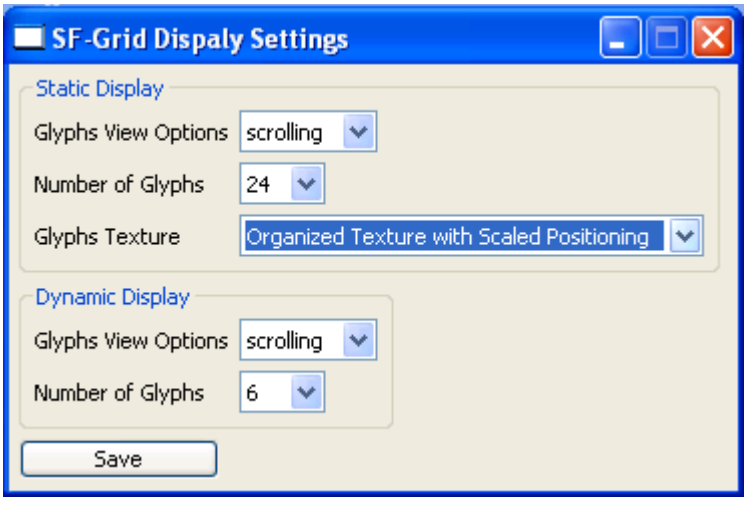


Figure 4.30. Screenshot: Select Organized Texture with Placement Property

We can get the following display (Figure 4.31) by selecting the above option. All glyphs objects (rectangles) contain a *Texture* at different positions.

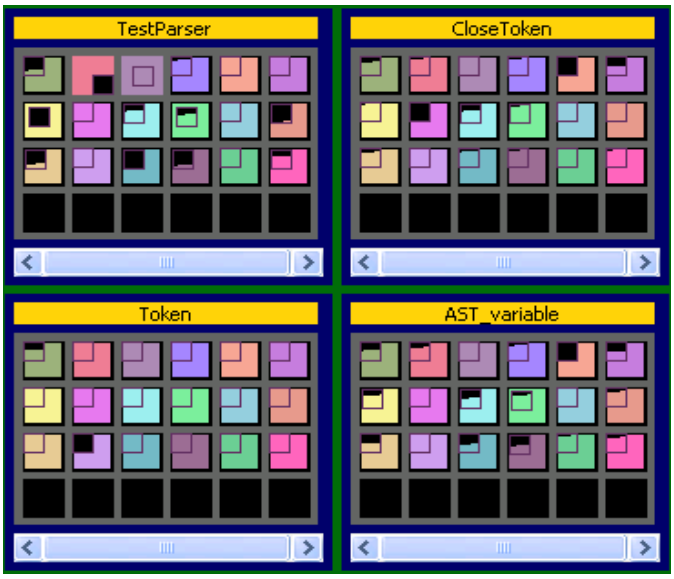


Figure 4.31. Screenshot: View Results of Last Setting in Static Display

4.2.6. Data Set Slider

Figure 4.32. Screenshot: Operate Data Set Slider

It helps, to scroll the data objects forward or backward. As we can see the slider bar at bottom of following Figure 4.32, let see if user press right end of slider.

Here, we can see in the next slide (Figure 4.33), where *2D-SFGrid* shows the next set of *Data Objects*. This slide (Figure 4.33) shows only three *Data Objects*, and rest of *Data Object Composites* displayed as blank. This shows that there are no more *Data Objects* in current *Dataset.*

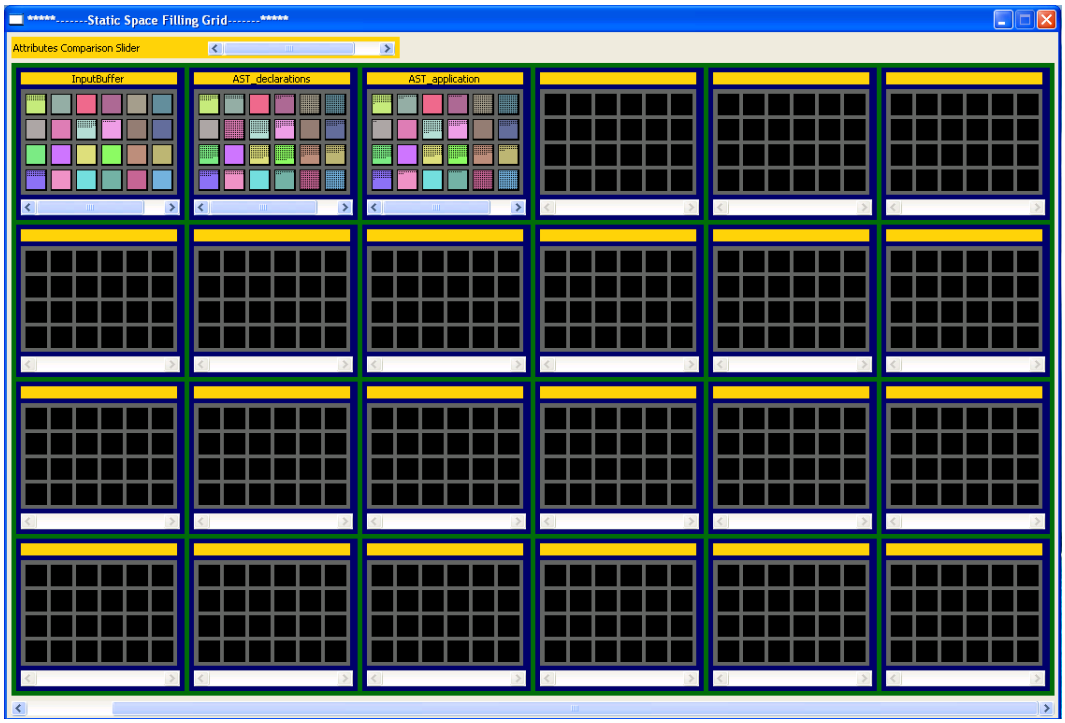


Figure 4.33. Screenshot: Get Next Slide by Pressing the Slider's Right End

4.2.7. Attribute Comparison Slider

This control is placed at the top of *2D-SFGrid*. If user wants to compare the same attributes in different *Data Objects*, he/she has the option to move the slider forward or backward to view next or previous attributes in different *Data Objects Composites*.

In the following Figure 4.34, the *Attribute Comparison Slider* is at start position, and all *Data Objects* are also at initial position.

Figure 4.34. Screenshot: Data Objects at Initial Position

When a user moves the *Attribute Comparison Slider* to the right, the next attribute comes up in all *Data Objects Composites* (Figure 4.35).

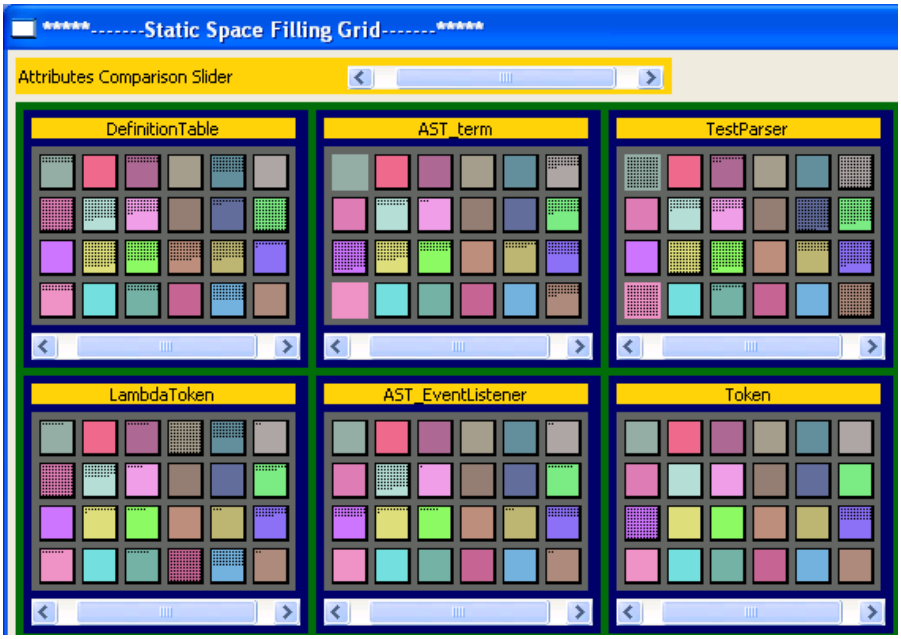


Figure 4.35. Screenshot: Next Attribute show in All Data Object Composite

4.2.8. View Information of Attribute

The user can get the complete information of the desired software metrics just by pressing the left mouse button on a glyph. We can see the information of attribute (CMM_1_0_CBO) in the following figure (Figure 4.36)

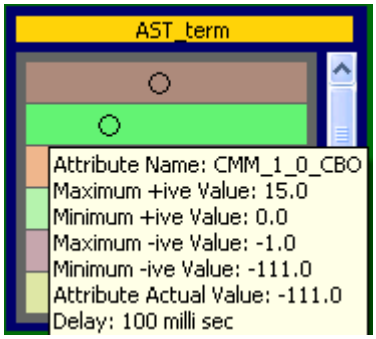


Figure 4.36. Screenshot: Information of Software Metrics by Mouse Click

4.2.9. View Full Qualified Name of Data Object

The user has also the option to get the *Full Qualified Name* of the desired *Data Object*; he/she needs to move the mouse over on *Data Object Name* (Figure 4.37).

Figure 4.37. Screenshot: Get Full Qualified Name of Data Object

4.3. Summary

In this chapter, we have explained the different visualization and interaction techniques that we have been proposed in Section 2.2. We also described in all properties of both static and dynamic visualization techniques. We have discussed all interaction techniques that are linked with functional requirements (Table 3.1) of chapter 3. Next chapter is all about Comparison of our proposed visualization approaches.

5. Comparison of Static and Dynamic Approaches

In this chapter, we will discuss practical results of both visualization approaches (static and dynamic). This section involves practical comparison; we will analyze the performance of both approaches through direct physical interaction. We will explain the performance comparison of our developed software (*2D-SFGrid*) by its major functions. Our developed software program (*2D-SFGrid*) is a visualization tool used to visualize the software metrics by two different approaches,

- Static Visualization
- Dynamic Visualization

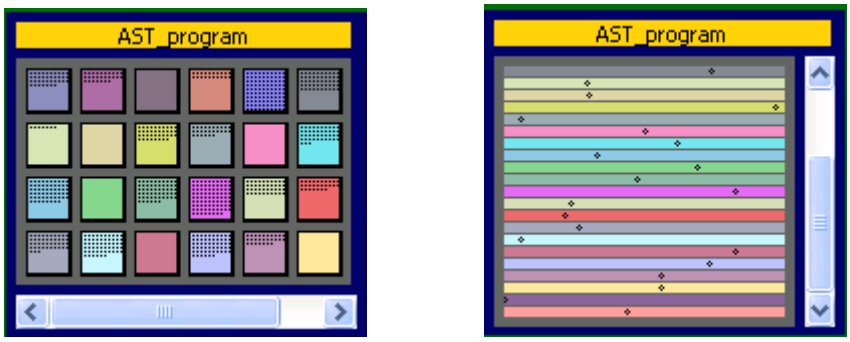
Both approaches are used to visualize software metrics for multivariate datasets. We have used the same data structure for both visualization techniques. The user can load a complete dataset once in the program. Then, the program builds a common data structure at loading time. When the dataset is loaded completely, then the user can use the same dataset several times for different type of visualizations (static and dynamic). Therefore, the time complexity of the dataset loading process is the same for both visualization approaches. It depends on the size of the dataset, and it is linear in O (D), where D is the number of data objects in the dataset.

5.1. Practical Comparison by Test Persons

Here, we have used a different approach to analyze our visualization tool (*2D-SFGrid*). Our main focus in this comparison test is perception and better understanding of software metrics. As the *2D-SFGrid* is a visualization tool for multivariate datasets, we should also analyze the performance of our system by direct human interaction. In this case, I have chosen three friends of mine from the group of Software Technology, MSI, Vaxjo University to compare the visualization performance of both approaches. They have analyzed both visualization techniques, and finally all of them agreed that the dynamic visualization is easy to perceive and better to understand than static visualization. They also recommend that dynamic display is only suitable for low number of attributes of data object. Here, we will discuss some scenarios in both approaches.

Compare Rendering feature in the both approaches

Here, we compare the rendering of software metrics in both visualization approaches. We will use the two different screenshots from both visualizations.



(a). Static approach (b). Dynamic approach

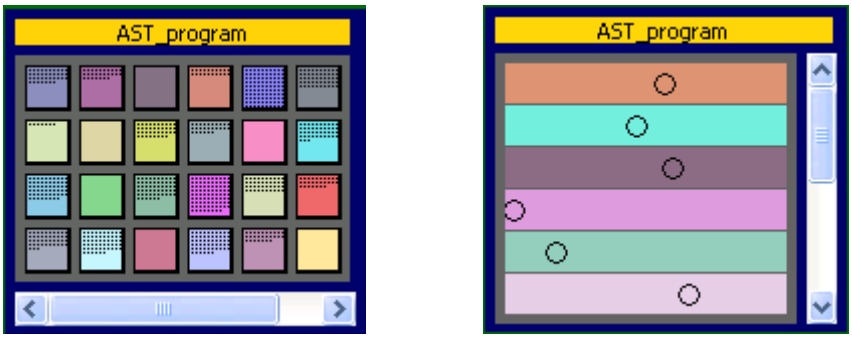
Figure 5.1. Comparison of rendering feature in both approaches

In Figure 5.1 (a) and (b), we have two different visualization results through both visualization approaches respectively. Our dynamic approach used an animation of moving object within glyph object. This cause a system performance problem, it is difficult to execute and manage all threads at same time. We examine that the dynamic approach cause the rendering problem by visualizing the more number of attributes and animation result is not reliable in this case.

On the other hand, we have a static approach, where software metrics is displayed by static glyph, and there are no runtime active threads for animation. We analyzed that, this approach is not affected by visualizing more number of attributes. Comparatively a static approach utilizes low system resources than dynamic approach, and it is better choice for the user to visualize more number of attributes rather than dynamic approach.

Compare property of Perception in both approaches

Perception is our main objective in this research work. We can see the visualization results of both approaches in the following Figure 5.2.

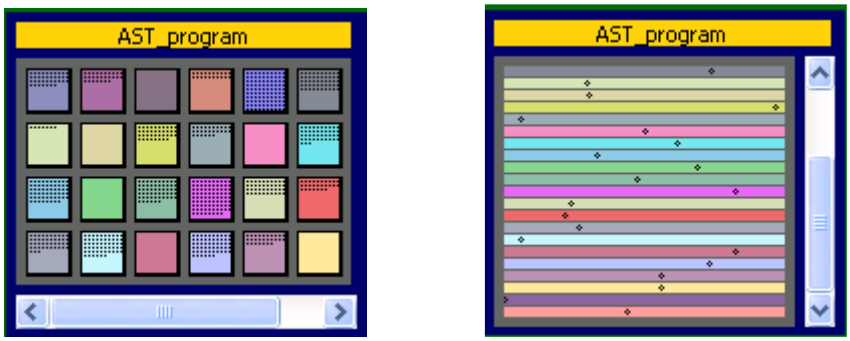


(a). Static approach (b). Dynamic approach

Figure 5.2. Case A: Comparison of perception property in both approaches

We can examine that the dynamic approach is easily perceive than static approach, but above Figure 5.2 shown that number of attributes are not same.

Now we change the scenario, and visualize the more number of attributes in dynamic approach same like static approach.



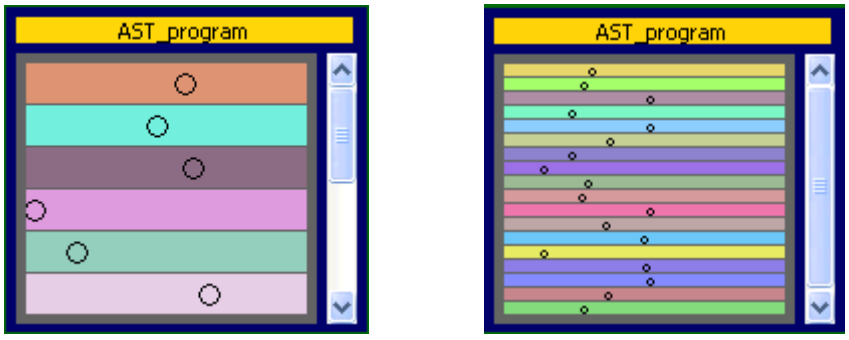
(a). Static approach (b). Dynamic approach

Figure 5.3. Case B: Comparison of perception property in both approaches

We can see in above Figure 5.3, where attributes are almost same in both approaches. In Figure 5.3 (b), shows more moving objects in a dynamic approach. It is very difficult to perceive some information from this dense animation and also very difficult to compare the same attribute in different data objects. After comparison of visualization results (Figure 5.3) from both approaches, we conclude that the results in static approach are more perceivable than dynamic approach.

Properties of Rendering and Perception in the dynamic visualization

We used a moving object in the dynamic visualization approach that creates an animation feature in glyphs objects. This approach is very fast and perceivable quickly for a low number of attributes. But, if we want to see more attributes in the same display then the number of moving objects are increased and the height of glyphs are decreased: this leads a performance and understanding problem for viewer.



(a) (b)

Figure 5.4. Comparison: Two different views in dynamic visualization

As we can see in Figure 5.1 (a) and (b), where six and eighteen glyphs objects are shown respectively in two different data object composites. These dynamic visualization results show that the user can better perceive the software metrics in Figure 5.1 (a) than those in Figure 5.1 (b).

The test persons also agreed that the static visualization is better for displaying more glyphs in a 2D grid. The static visualization is very fast in rendering the display of software metrics in a 2D grid with low utilization of resources, and this is suitable for slow machines (processors).

5.2. Summary

This chapter is all about the performance comparison of both visualization (static and dynamic) approaches. We have analyzed the performance of both techniques by practical comparisons. The next chapter concludes of our work.

6. Conclusion

Our intent in this thesis was to research and develop visualization techniques to visualize software metrics. In this chapter, we will discuss our achievement of this research project. We will also explain future work that is related to our proposed visualization techniques.

6.1. Results

In this thesis, we have presented two approaches for visualization of software metrics, i.e., a multivariate dataset is the main input of these visualizations. Both visualization techniques are useful in different perspectives. We have introduced a 2D space-filling grid for the visualization of software metrics, and it is used for both visualizations (static and dynamic). Our approach for the visualization of software metrics is different from other visualization systems. Normally, other visualization tools like [32], [33], and [34] have visualized the complete dataset at once in visualization display. We introduced a sliding technique to visualize the complete dataset rather than zooming technique. Each slide contains a set of data objects, the user can access any slide (set of data objects) of dataset in constant time O (D), where D is number of data objects. These results have shown the better performance (rendering and data access) of our proposed visualization than other existing visualization techniques. Here we will describe the solution of our defined goals in this project.

6.1.1. Visualization Techniques

A very first goal of this project was to implement a visualization technique that is user friendly and time efficient for a large dataset. We developed a visualization of software metrics using a 2D grid. As in Chapter 2, we discussed previous visualization work for multivariate datasets. In our study, we found a research work related to visualization of software metrics through static texture, and we were decided to develop a static visualization tool for software metrics.

Similarly, we also found some research work related to motion of objects to represents the software metrics. After analysis on previous research work of both static texture and motion of glyphs, we have decided to develop both visualization techniques (static and dynamic) for the same multivariate dataset.

As we introduced a sliding technique to visualize the dataset and data object with in 2D-Grid. This technique makes user friendly and time efficient access to get the any data object or attribute value in constant time O (1).

6.1.2. Graphical Glyphs

As a very important goal in this project, we defined graphical glyphs objects that support flexibility in their placement and in their ability to represent multivariate datasets. The complete discussion of this goal is already mentioned in Chapter 2, where we discuss previous research on glyphs objects, and also describe the proposed glyphs objects for this visualization project. The practical solution of

graphical glyph was described in Chapter 5, where each glyph object represents an attribute value of a data object.

6.1.3. Software Metrics

As we discussed software metrics in Chapter 1, it represents the measurement of a software component or its specification. In this project, we constructed two types of software metrics visualization: one is static and other is dynamic. Both type of software metrics are used to represents the attributes values using static texture and animation respectively. A static software metrics has used the glyphs object with static texture and on other hand; we have used a glyph object with moving object for dynamic software metrics.

6.1.4. Comparison of static and dynamic visualization approaches

As a last goal of this project, our focus is on practical comparison of our both visualization approaches. Practically, loading process time of dataset in both visualization approaches is same, but dynamic approach is more expensive than static approach, because in the dynamic approach, motion of object within glyphs is continuously in process. On other hand in static approach, all glyphs objects are loaded at once in start with static texture.

As we know the processing of dynamic approach is more expensive than static, but we have focused on the perception performance of visualization. We have already discussed the detail comparison of both approaches in chapter 5, where we concluded that the perception of dynamic visualization is more effective and better understandable than static visualization. But dynamic visualization is only useful for low number of glyphs objects in each data object. On other side texture based or static visualization technique is more convenient to display large number of glyphs in *Data Object Composite*.

6.2. Future Work

This thesis presented two different techniques for displaying software metrics in a 2D space-filling grid. After the completion of this project, we conclude with the following future work:

- To load large datasets into the program memory faster than now.
- Improve the data structure to accommodate the dataset.
- Introduce the zooming feature in our visualization application.
- Visualize the complete dataset in single display.
- Extend to flexible displays, especially for large screens, projectors etc.
- Our intent to further research on both approaches to visualize the complete multivariate dataset in a single display. It can be controlled by zooming and scrolling options.

7. Glossary

2D	2 Dimension
2D-SFGrid:	2 Dimensional Space-Filling Grids
API:	Application Programming Interface
AWT:	Abstract Window Toolkit
EMF:	Eclipse Model Framework
GEF:	Graphical Editing Framework
GUI:	Graphic User Interface
IDE:	Integrated Development Environment
JDK:	Java Development Kit
JDT:	Java Development Tools
LWS:	Light Weight System
MVC:	Model-View-Controller
OMT:	Object Modeling Technique
OOSE:	Object-Oriented Software Engineering
PDE:	Plug-in Development Environment (PDE)
SDK:	Software Development Kit
SWT:	Standard Widgets Toolkit
UML:	Unified Modeling Language
JVM:	Java Virtual Machine

8. References

[1] DeMarco, Tom. *Controlling Software Projects*: Management, Measurement and Estimation. ISBN 0-13-171711-1.

[2] Douglas Hubbard, *The IT Measurement Inversion CIO Magazine*, 1999.

[3] Diehl, S. (2002). *Software Visualization. International Seminar*. Revised Papers (LNCS Vol. 2269), Dagstuhl Castle, Germany, 20-25 May 2001 (Dagstuhl Seminar Proceedings).

[4] *Visualizing Multidimensional Query Results Using Animation*, Amit P. Sawanta and Christopher G. Healey, North Carolina State University, Department of Computer Science, Raleigh, NC, USA.

[5] Keim, D. A. (2002). *Information visualization and visual data mining*. IEEE Transactions on Visualization and Computer Graphics, USA * vol 8 (Jan. March 2002), no 1, p 1 8, 67 refs.

[6] Visualization Handbook (Hardcover) by Charles D. Hansen, Chris Johnson, Academic Press (June, 2004).

[7] Wikipedia. en.wikipedia.org/wiki/Software_visualization, 2008.

[8] Wikipedia. en.wikipedia.org/wiki/Software_metric, 2008.

[9] Andreas Kerren, Achim Ebert, and Jörg Meyer (Eds.). Human-Centered Visualization Environments. Volume 4417 of Lecture Notes in Computer Science (LNCS) Tutorial, Springer, 2007.

[10] Rational Software Corporation. *The Objectory Process--Introduction*, 1996.

[11] Aurum A. Cox, K. and Jeffery. *An experiment in inspecting the quality of usecase descriptions.* Journal of Research and Practice in Information Technology, 36(4):211–229, 2004.

[12] Reed, T. R., and Hans Du Buf, J. M. A review of recent texture segmentation and feature extraction techniques. *CVGIP: Image Understanding 57*, 3 (1993), 359-372.

[13] Julesz, B. Textons, the elements of texture perception, and their interactions. *Nature 290* (1981), 91-97.

[14] Julesz, B. A theory of preattentive texture discrimination based on first-order statistics of textons. *Biological Cybernetics 41* (1981), 131-138.

[15] Julesz, B. A brief outline of the texton theory of human vision. *Trends in Neuroscience 7*, 2 (1984), 41-45.

[16] Schweitzer, D. Artificial texturing: An aid to surface visualization. *Computer Graphics (SIGGRAPH 83 Conference Proceedings) 17*, 3 (1983), 23-29.

[17] Grinstein, G., Pickett, R., and Williams, M. EXVIS: An exploratory data visualization environment. In *Proceedings Graphics Interface '89* (London, Canada, 1989), pp. 254-261.

[18] Ware, C., and Knight, W. Using visual texture for information display. *ACM Transactions on Graphics 14*, 1 (1995), 3-20.

[19] Turk, G., and Banks, D. Image-guided streamline placement. In *SIGGRAPH 96 Conference Proceedings* (New Orleans, Louisiana, 1996), H. Rushmeier, Ed., pp. 453-460.

[20] Interrante, V. Illustrating surface shape in volume data via principle direction-driven 3d line integral convolution. In *SIGGRAPH 97 Conference Proceedings* (Los Angeles, California, 1997), T. Whitted, Ed., pp. 109-116.

[21] Salisbury, M., Wong, M. T., Hughes, J. F., and Salesin, D. H. Orientable textures for image-based pen-and-ink illustration. In *SIGGRAPH 97 Conference Proceedings* (Los Angeles, California, 1997), T. Whitted, Ed., pp. 401-406.

[22] Laidlaw, D. H., Ahrens, E. T., Kremers, D., Avalos, M. J., Jacobs, R. E., and Readhead, C. Visualizing diffusion tensor images of the mouse spinal cord. In *Proceedings Visualization 98* (Research Triangle Park, North Carolina, 1998), pp. 127-134.

[23] Meier, B. J. Painterly rendering for animation. In *SIGGRAPH 96 Conference Proceedings* (New Orleans, Louisiana, 1996), H. Rushmeier, Ed., pp. 477-484.

[24] Bruckner, L. A. On Chernoff faces. In *Graphical Representation of Multivariate Data*, P. C. C. Wang, Ed. Academic Press, New York, New York, 1978, pp. 93-121.

[25] Chernoff, H. The use of faces to represent points in k-dimensional space graphically. *Journal of the American Statistical Association 68*, 342 (1973), 361-367.

[26] Foley, J., and Ribarsky, W. Next-generation data visualization tools. In *Scienti_c Visualization: Advances and Challenges*, L. Rosenblum, Ed. Academic Press, San Diego, California, 1994, pp. 103-127.

[27] Levkowitz, H. Color icons: Merging color and texture perception for integrated visualization of multiple parameters. In *Proceedings Visualization '91* (San Diego, California, 1991), pp. 164-170.

[28] Brown, J. L. Flicker and intermittent stimulation. *Vision and Visual Perception*, C. H. Graham, Ed. John Wiley & Sons, New York, New York, 1965, pp. 251–320.

[29] Meier, B. J. Painterly rendering for animation. *SIGGRAPH 96 Conference Proceedings* (New Orleans, Louisiana, 1996), pp. 477–484.

[30] Tynan, P. D. and Sekuler, R. Motion processing in peripheral vision: Reaction time and perceived velocity. *Vision Research 22,*1 (1982), 61-68.

[31] A. J. van Doorn and J. J. Koenderink. Temporal properties of the visual detectability of moving spatial white noise. *Experimental Brain Research*, 45:179–188, 1982.

[32] Huber, D. E. and Healey, C. G. "Visualizing Data with Motion." In Proceedings IEEE Visualization 2005 (Minneapolis, Minnesota), pp. 527-534.

[33] Healey, C. G. and Enns, J. T. "Large Datasets at a Glance: Combining Textures and Colors in Scientific Visualization." IEEE Transactions on Visualization and Computer Graphics 5, 2, (1999), 145-167.
[34] Healey, C. G. and Enns, J. T. "Building Perceptual Textures to Visualize Multidimensional Datasets." In Proceedings IEEE Visualization '98 (Research Triangle Park, North Carolina, 1998), pp. 111-118.
[35] Stephan Diehl (Stephan Diehl, editor. *Software Visualization*, Springer State-of-the-Art Survey LNCS 2269. Springer Verlag, 2002.)
[36] J. T. Stasko, J. Domingue, M. H. Brown, and B. A. Price. *Software Visualization*. MIT Press, 1998.
[37] Algorithm Animation - Chapter Introduction. A. Kerren and J. T. Stasko. Software Visualization, volume 2269 of LNCS State-of-the-Art Survey, pages 1-15. Springer, 2002.
[38] Sawant, A. P. and Healey, C. G. "Visualizing Multidimensional Query Results Using Animation." To appear in *Proceedings Visualization and Data Analysis 2008* (San Jose, California)
[39] Matthew O. Ward , *A taxonomy of glyph placement strategies for multidimensional data visualization*, ISSN:1473-8716, 2002
[40] Colin Ware, *Information Visualization*, Perception for design, 2002.
[41] Liu, G., Enns, J. T., and Healey, C. G. "Sensitivity to 3D Orientation in Textured Surfaces." In Psychonomics 2000 Poster Session #599 (New Orleans, Louisiana, 2000).
[42] St. Amant, R., Healey, C. G., Riedl, M., Kocherlakota, S., Pegram, D. A., and Torhola, M. "Intelligent Visualization in a Planning System." In Proceedings Intelligent User Interfaces 2001 (Santa Fe, New Mexico, 2001), pp. 153-160.
[43] Steve Northover, Mike Wilson. *SWT: The Standard Widget Toolkit*. Addison-Wesley Professional; 1 edition (Jul 8 2004).
[44] Sherry Shavor et al., *The Java Developer's Guide to Eclipse*, Addison-Wesley, 2003.
[45] Berthold Daum, *Eclipse 2 for Java Developers*, Wiley, 2003.
[46] Erich Gamma and Kent Beck, *Contributing to Eclipse*, Addison-Wesley, 2003.
[47] H. Hinterberger. Data Density: *A Powerful Abstraction to Manage and Analyze Multivariate Data*. PhD thesis, Informatik-Dissertationen ETH, Zurich, 1987. No. 4.
[48] Ivar Jacobson and James Rumbaugh. *The Unified Software Development Process*. Prentice Hall, 1999. ISBN 978-0-201-57169-1.
[49] http://www.eclipse.org/ [accessed 15 April 2008].
[50] http://www.eclipse.org/swt/ [accessed 27 April 2008].
[51] http://www.java.com/en/download/index.jsp [accessed 15 April 2008].
[52] http://www.soyatec.com/euml2/installation/ [accessed 22 June 2008]
[53] http://en.wikipedia.org/wiki/Iterative_and_incremental_development [accessed 27 July 2008]

Appendix A: Implementation

In this chapter, we will describe implementation aspects of our visualization system (2D-SFGrid). This chapter includes brief description of, major components of our developed software, platform and environment in which codes are written, and the tools are used in software development.

A. 1. Visualization System Components

In this section, we will discuss some major components of our implemented visualization system. In the following Figure A.1, we can see the main components linked with visualization system.

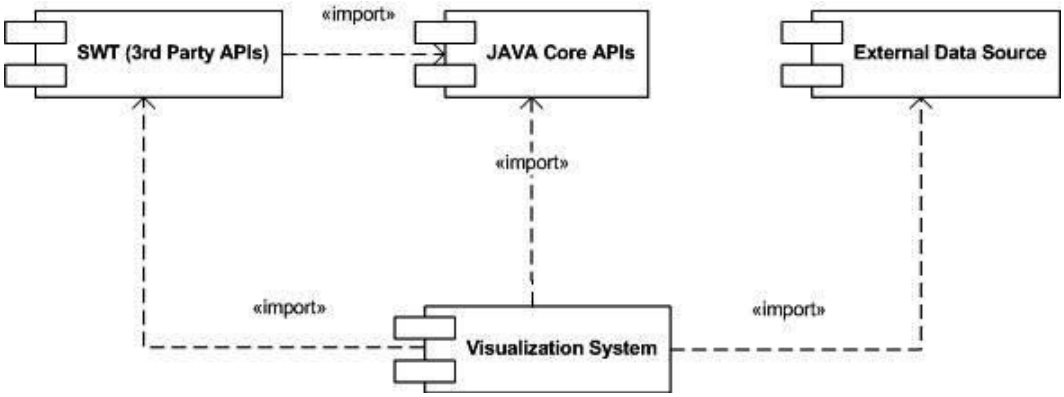


Figure A.1. Component Diagram: Visualization System

We have used a SWT (Standard Widget Toolkit) toolkit in our visualization system for graphical purposes. JAVA core API's are used as a main language of our software development. External Data Source is main source of data for our software; we have used an Excel database as an input source (dataset) for our developed visualization tool (2D-SFGrid).

A. 2. Eclipse Development Environment

We used Eclipse environment in the development of this project. Eclipse is an open source community. Its projects are focused on building an open development platform comprised of extensible frameworks, tools and runtimes for building, developing and managing software across the life style [44], [45]. The Eclipse Foundation is a not-for-profit, member supported corporation that helps cultivate both an open source community and an ecosystem of complementary products and services [46].

Eclipse is available via eclipse main website [49]. For using, Eclipse Java runtime environment (JRE) is needed (Java 1.6 JRE recommended) [51]. We also used an eclipse plug-in [52] to create the UML diagrams.

A. 3. SWT

The Standard Widget Toolkit (SWT) is a graphical widget toolkit use for the Java platform. It was originally developed by IBM systems but now Eclipse Foundation is maintained for Eclipse IDE. It is a substitute to the AWT and Swing Java GUI toolkits provided by Sun Microsystems as part of the Java Platform, Standard Edition J2SE [43].

SWT is written completely in Java. It is very useful to display GUI elements, the SWT implementation accesses the native GUI libraries of the operating system using JNI (Java Native Interface) in a way that is similar to those programs written using operating system required APIs. Programs that call SWT are portable, but the implementation of the toolkit, despite the fact that it is written in Java, is unique for each platform [43]. SWT development toolkit is available for free download on eclipse main website in SWT section [50].

A. 4. Software Development

All codes are programmed in Eclipse programming environment. Java has been chosen for programming language. This project is comprised of a number of classes and interfaces,

Figure A.2. 2D-SFGrid: Class Design Diagram

spacefillinggrid

Class SFGrid

```
java.lang.Object
  └ spacefillinggrid.SFGrid
```

```
public class SFGrid
extends java.lang.Object
```

This class provides 2D space-filling grid display that shows the dynamic and static behavior of glyph objects, and shows the visualization of dataset.

Field Summary	
protected java.util.BitSet	**attributesBitSet**: contains the status of all attributes either true or false, depends on user selection in Attribute Definition Window, by default all true
protected Slider	**dataSetSlider**: dataset slider, setup horizontally at the bottom of SF-Grid window
protected boolean	**dynamicGlyph** property of dynamic glyph is true on user selection otherwise false
protected SFDisplaySettings	**sfDisplaySettings** display settings selected by user
protected boolean	**staticGlyph** property of static glyph is true on user selection otherwise false
(package private) int	**windowHeight** default height of SF-Grid window
(package private) int	**windowWidth** default width of SF-Grid window
Constructor Summary	
SFGrid(int approachValue, SFDisplaySettings sfDSettings) default constructor, needs approach value either static (1) or dynamic (2) and instance of SFDisplaySettings	
Method Summary	
void	**create2DGridDynamic**(Shell shell)
void	**create2DGridStatic**(Shell shell) create and setup 2D static grid environment, glyphs created as a static display
protected boolean	**updateGlyphsBox**(Composite glyphsComposite, Label objectLabel, int dataObjectNumber, int innerSliderPosition, int topSliderPosition, int boxSize) this protected method, only use internally.

Table A.1. *SFGrid: Class Document*

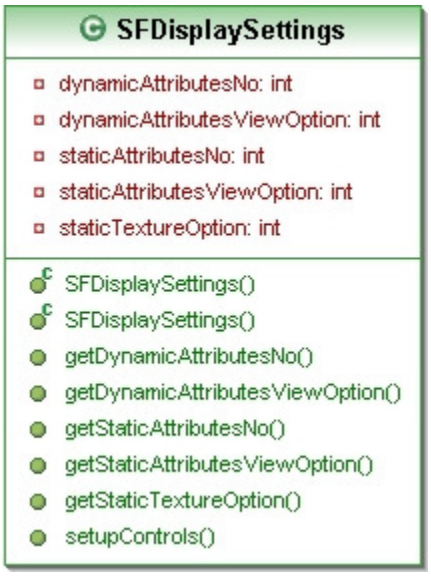


Figure A.3. SFDisplaySettings: Class Design Diagram

spacefillinggrid

Class SFDisplaySettings

```
java.lang.Object
  └ spacefillinggrid.SFDisplaySettings
```

```
public class SFDisplaySettings
extends java.lang.Object
```

This class contains functionality to control the 2D Space-Filling grid display settings. This component gives a visual interface to user; user can set setting for both static and dynamic 2D-SFGrid windows.

Field Summary	
private int	**dynamicAttributesNo** for total number of dynamic attributes of data object to be displayed in SF-Grid window
private int	**dynamicAttributesViewOption** for sliding (1) or scrolling (0) options in SF-Grid window, by default scrolling is set for dynamic view
private int	**staticAttributesNo** for total number of static attributes of data object to be displayed in SF-Grid window
private int	**staticAttributesViewOption** for sliding (1) or scrolling (0) options in SF-Grid window, by default scrolling is set for static view
private int	**staticTextureOption** this is a texture property, by default it is 0 (for organized texture)

Constructor Summary	
SFDisplaySettings()	default constructor
SFDisplaySettings(SFDisplaySettings displaySettings)	this is one argument constructor, this constructor receive last saved display settings.

Method Summary	
int	**getDynamicAttributesNo**() to get the number of glyphs, that user want to display in each data object composite in dynamic grid
int	**getDynamicAttributesViewOption**() to get the dynamic view option either scrolling or sliding
int	**getStaticAttributesNo**() to get the number of glyphs to display in each data object composite in static grid
int	**getStaticAttributesViewOption**() to get the settings by user, either scrolling or sliding to view the more attributes in data object composite
int	**getStaticTextureOption**() to send the texture option, this value help to
void	**setupControls**(SFDisplaySettings savedSetting, Shell shell) to setup the controls properly, it provides the user interaction by selection boxes and buttons

Table A.2. *SFDisplaySettings : Class Document*

Figure A.4. MessageHandler: Class Design Diagram

spacefillinggrid

Class MessageHandler

```
java.lang.Object
  └java.lang.Throwable
      └spacefillinggrid.MessageHandler
```

All Implemented Interfaces:

java.io.Serializable

```
public class MessageHandler
extends java.lang.Throwable
```

Use for exception handling, report error message using dialog window

Field Summary	
private static long	**serialVersionUID**
Constructor Summary	
MessageHandler()	
Method Summary	
static void	**show**(java.lang.Exception e, java.lang.String msgType, java.lang.String msg)
static void	**show**(java.lang.String msgType, java.lang.String msg)

Table A.3. *MessageHandler : Class Document*

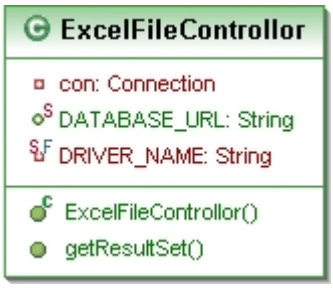


Figure A.5. ExcelFileControllor: Class Design Diagram

spacefillinggrid

Class ExcelFileControllor

```
java.lang.Object
  └ spacefillinggrid.ExcelFileControllor
```

```
public class ExcelFileControllor
extends java.lang.Object
```

This class have link with excel data base through DSN Bridge. We used DSN name: matrices_xls by default.

Field Summary	
private java.sql.Connection	**con** connection object, use to get the connection through ODBC connection
static java.lang.String	**DATABASE_URL** set DSN name, same name use as DSN name or alias
private static java.lang.String	**DRIVER_NAME** driver for JDBC and ODBC connection
Constructor Summary	
ExcelFileControllor() default constructor	
Method Summary	
java.sql.ResultSet	**getResultSet**(java.lang.String sqlQuery)

Table A.4. *ExcelFileControllor: Class Document*

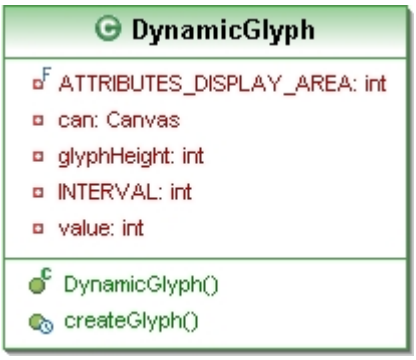


Figure A.6. DynamicGlyph: Class Design Diagram

spacefillinggrid

Class DynamicGlyph

```
java.lang.Object
  └ spacefillinggrid.DynamicGlyph
```

```
public class DynamicGlyph
extends java.lang.Object
```

This class generates the dynamic glyphs, each glyph object represent one attribute of data object. This class uses the internal data structure to display the information of each attribute.

Field Summary	
private int	**ATTRIBUTES_DISPLAY_AREA** display area of all attributes for each data object composite
private Canvas	**can** rectangle, shows the attribute
private int	**glyphHeight** height of each attribute glyph object
private int	**INTERVAL** delay time of circle object motion of each attribute
private final int	**NOA** number of attributes to display
private int	**value** used for position of moving circle in glyph (rectangular box)
Constructor Summary	
DynamicGlyph(Composite composite, int noOfAttributes) constructor, initialize and start the motion of object	
Method Summary	
void	**createGlyph**() this method create and manage the glyph object.

Table A.5. *DynamicGlyph: Class Document*

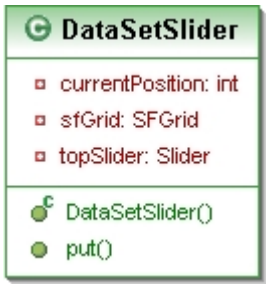


Figure A.7. DataSetSlider: Class Design Diagram

spacefillinggrid

Class DataSetSlider

```
java.lang.Object
  └ spacefillinggrid.DataSetSlider
```

```
public class DataSetSlider
extends java.lang.Object
```

This class creates and manages the horizontal slider to control the dataset. it comes at the bottom of SF grid window

Field Summary	
private int	**currentPosition** to access the current position of slider
private SFGrid	**sfGrid** sfGrid object of SFGrid, to access the main window display
private Slider	**topSlider** SWT slider control, placed at bottom of SFGrid window
Constructor Summary	
DataSetSlider(SFGrid sf) one parameter Constructor, initialize the current position of slider control	
Method Summary	
Slider	**put**(Shell shell) this method create and manage the slider control.

Table A.6. *DataSetSlider: Class Document*

Figure A.8. DataSetManager: Class Design Diagram

spacefillinggrid

Class DataSetManager

```
java.lang.Object
  └ spacefillinggrid.DataSetManager
```

```
public class DataSetManager
extends java.lang.Object
```

This class is used to load the dataset, and construct some tricky data structure for further use for visualization of software metrics in 2D space-filling grid.

Field Summary	
private static java.util.Map<java.lang.String,AttributeManager>	**attMngrMap** map contains the records of all attributes by key (name of attribute)
private static java.util.List<java.lang.String>	**attributeList** contains list of attributes names
static java.util.BitSet	**attributesBitSet** bit set object, use for attributes to set true or false have public access
private static int	**attributeSize** size of attributes in dataset

private static java.util.Map<java.lang.Integer,DataObjectDetails>	**dataSetMap** map contains the records of all data object by key value (number start from zero)
static java.lang.Double	**maxNegAttributeValue**
static java.lang.Double	**maxPosAttributeValue** load the dataset into internal data structure, takes time in loading, it depends on size of dataset
static java.lang.Double	**minNegAttributeValue**
static java.lang.Double	**minPosAttributeValue**
private static int	**objectSize** size of dataset, total number of records or data objects in dataset
protected static java.lang.String	**sheetNo** sheet number of excel files, final access, constant

Constructor Summary

DataSetManager() default constructor

Method Summary

static AttributeManager	**getAttributeDefinition**(java.lang.String key) key is attribute name, it returns the object that contains the definition of attribute
static java.util.List<java.lang.String>	**getAttributeList**() return list of attributes
static int	**getAttributeSize**() return the size of attributes, total number of attributes in data object
static DataObjectDetails	**getDataObjectDetails**(int key) key is number of data object (start from zero), and returns the object that contains the definition of data object
static int	**getObjectSize**() return number of data objects in current dataset
void	**manageDataSet**(Button b) load the dataset into internal data structure, takes time in loading, it depends on size of dataset

Table A.7. *DataSetManager: Class Document*

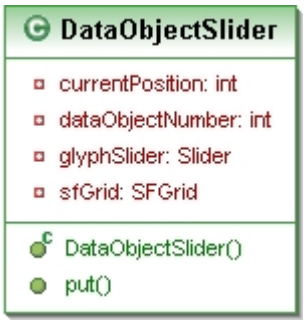


Figure A.9. DataObjectSlider: Class Design Diagram

spacefillinggrid

Class DataObjectSlider

```
java.lang.Object
  └ spacefillinggrid.DataObjectSlider
```

```
public class DataObjectSlider
extends java.lang.Object
```

This class create and manage the horizontal slider for each record (data object) of dataset.

Field Summary	
private int	**currentPosition** to access the current position of slider
private int	**dataObjectNumber** dataObjectNumber unique number for each data object
private Slider	**glyphSlider** slider object, use to view the all attribute values by scrolling the slider
private SFGrid	**sfGrid** sfGrid object of SFGrid, to access the main window display
Constructor Summary	
DataObjectSlider(SFGrid sfGrid, int dataObjectNumber) two parameter Constructor, initialize the current position of slider control	
Method Summary	
void	**put**(Composite composite) this method use to add the slider object for each data object in SFGrid window, this also includes the controller for slider position by user interaction

Table A.8. *DataObjectSlider: Class Document*

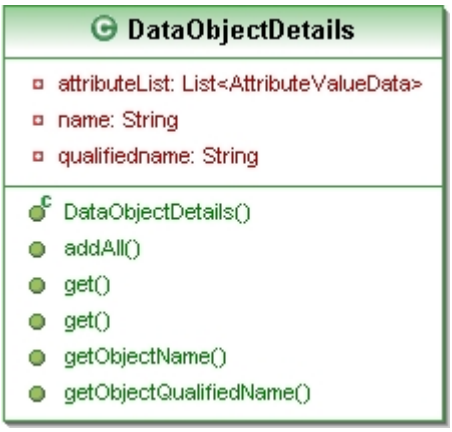


Figure A.10. DataObjectDetails: Class Design Diagram

spacefillinggrid

Class DataObjectDetails

```
java.lang.Object
  └ spacefillinggrid.DataObjectDetails
```

```
public class DataObjectDetails
extends java.lang.Object
```

This class contains the details of each data object

Field Summary	
private java.util.List<AttributeValueData>	**attributeList** this list contains the value of all attributes in this data object
private java.lang.String	**name** name of data object
private java.lang.String	**qualifiedname** qualified name of data object

Constructor Summary
DataObjectDetails(java.lang.String name, java.lang.String qualifiedname) two parameter constructor, initialize the name and qualified name of data object

Method Summary	
void	**addAll**(java.util.List<AttributeValueData> avdList) to add the all attributes values of this data object into list
AttributeValueData	**get**(int index) to get the specified value of attribute by passing the index of the list
java.util.List<A	**get**(int fromIndex, int toIndex)

ttributeValueData>	to get the sublist of attributes values of this data object
java.lang.String	**getObjectName**() to get the name of data object
java.lang.String	**getObjectQualifiedName**() to get the qualified name of data object

Table A.9. *DataObjectDetails: Class Document*

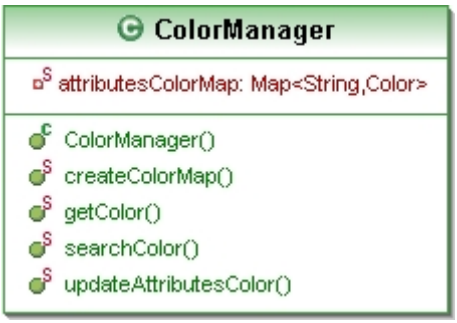


Figure A.11. ColorManager: Class Design Diagram

spacefillinggrid

Class ColorManager

```
java.lang.Object
  └ spacefillinggrid.ColorManager
```

```
public class ColorManager
extends java.lang.Object
```

This class is, to generate the color code for each attribute, and it is manage by Map, we can also access the color of each attribute directly by calling getColor().

Field Summary	
private static java.util.Map<java.lang.String,Color>	**attributesColorMap** contains set of attributes along their colors code
Constructor Summary	
ColorManager() default constructor	
Method Summary	
static void	**createColorMap**(java.util.List<java.lang.String> attributesSet) get the attributes list from MainControllor class, and generate the unique color code for each attribute
static Color	**getColor**(java.lang.String attribute) to get the color of attribute
static boolean	**searchColor**(RGB rgb) to searching of color, found or not
static java.lang.String	**updateAttributesColor**(Color oldAttributeColor, Color newAttributeColor) update color definition in ColorManager

Table A.10. *ColorManager: Class Document*

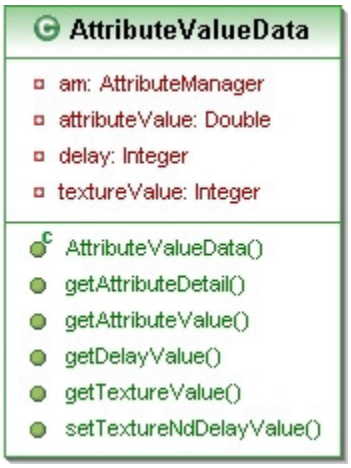


Figure A.12. AttributeValueData: Class Design Diagram

spacefillinggrid

Class AttributeValueData

```
java.lang.Object
  └ spacefillinggrid.AttributeValueData
```

```
public class AttributeValueData
extends java.lang.Object
```

This class contains the value of attribute, means the value of each cell have one {AttributeValueData} object, also defines the texture (0-100, 0 show minimum attribute value and 100 shows maximum attribute value) and delay (0-100, 0 show maximum attribute value and 100 shows minimum attribute value) values by actual attribute value, also contains the {AttributeManager} object.

This class also have two properties of each attribute, one is Texture and another is Delay, Texture: is combination of pixels and used to show the strength of attribute's actual value, Delay: the delay value shows the delay time of glyph object from on state to another state during motion, the delay value is different than texture value, this is use to control the speed of glyphs, high delay present low actual attribute value and low delay value present high actual attribute value.

Field Summary	
private AttributeManager	**am** keep the definition of attribute, its color, name, value's range
private java.lang.Double	**attributeValue** value of attribute in data object
private java.lang.Integer	**delay** this value is calculated through SCALING technique, this is between from 0-100
private java.lang.Integer	**textureValue** it is calculated texture value through SCALING

	technique, this value is between from 0-100, the value is used to present the strength of actual value
Constructor Summary	
AttributeValueData(AttributeManager am, java.lang.Double attributeValue) default constructor, needs the object of AttributeManager and attribute value	
Method Summary	
AttributeManager	**getAttributeDetail**() to get attribute definition of this value
java.lang.Double	**getAttributeValue**() to get actual attribute value
java.lang.Integer	**getDelayValue**() to get the delay value, for dynamic display
java.lang.Integer	**getTextureValue**() to get the texture value, for static display
void	**setTextureNdDelayValue**() set the texture and delay values against original value

Table A.11. *AttributeValueData: Class Document*

Figure A.13. AttributeWindow: Class Design Diagram

spacefillinggrid

Class AttributesWindow

```
java.lang.Object
  └ spacefillinggrid.AttributesWindow
```

```
public class AttributesWindow
extends java.lang.Object
```

This class provides the definition of all attributes in dataset, this class also facilitates the user to view and update the settings of attributes

Field Summary	
private java.util.BitSet	**bitSet** BitSet, this hold the status of attributes, either true or false
private int	**currentPosition** control and access the current position of slider control
Constructor Summary	
AttributesWindow() default constructor, create a instance of AttributeWindow	
Method Summary	
void	**openWindow**() initialize the attribute BitSet, also set the current position of slider, open the attributes definition window, this method contains the layout and display of controls in attribute window
private void	**updatComposites**(Slider sld) private method, use to update the information of visual controls

Table A.12. *AttributesWindow: Class Document*

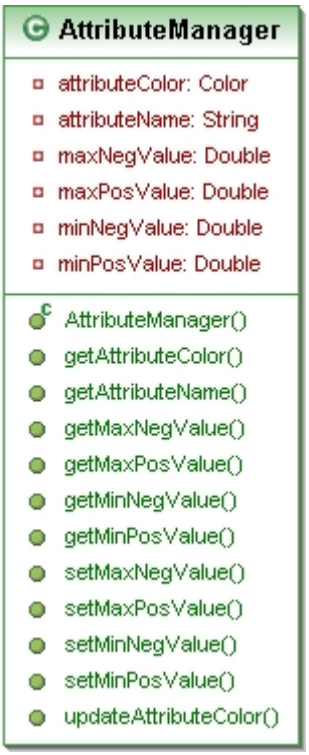


Figure A.14. AttributeManager: Class Design Diagram

spacefillinggrid

Class AttributeManager

```
java.lang.Object
  └ spacefillinggrid.AttributeManager
```

```
public class AttributeManager
extends java.lang.Object
```

This class use to create a one object for each attribute that contains the properties of attribute like, name, color, positive and negative range of values

Field Summary	
private Color	**attributeColor** unique color for each attribute
private java.lang.String	**attributeName** unique name of the attribute
private java.lang.Double	**maxNegValue** maximum negative value of this attribute
private java.lang.Double	**maxPosValue** maximum positive value of this attribute
private java.lang.Double	**minNegValue** minimum negative value of this attribute
private	**minPosValue**

java.lang.Double	minimum positive value of this attribute

Constructor Summary

AttributeManager(Color attributeColor, java.lang.String attributeName) two parameter constructor, attribute color and its name

Method Summary

Color	**getAttributeColor**() this method can use to get the attribute color
java.lang.String	**getAttributeName**() to get the name of this attribute
java.lang.Double	**getMaxNegValue**() to get the maximum negative value of attribute
java.lang.Double	**getMaxPosValue**() to get the maximum positive value of attribute
java.lang.Double	**getMinNegValue**() to get the minimum negative value of attribute
java.lang.Double	**getMinPosValue**() to get the minimum positive value of attribute
void	**setMaxNegValue**(double maxNegValue) useful to display the texture and speed of glyph objects
void	**setMaxPosValue**(double maxPosValue) useful to display the texture and speed of glyph objects
void	**setMinNegValue**(double minNegValue) useful to display the texture and speed of glyph objects
void	**setMinPosValue**(double minPosValue) useful to display the texture and speed of glyph objects
void	**updateAttributeColor**(Color attributeColor) to update the new color in attribute definition

Table A.13. *AttributeManager: Class Document*

Figure A.15. AttributeComparisonSlider: Class Design Diagram

spacefillinggrid

Class AttributeComparisonSlider

```
java.lang.Object
  └ spacefillinggrid.AttributeComparisonSlider
```

```
public class AttributeComparisonSlider
extends java.lang.Object
```

This class contains a functionality of comparison slider; it is top slider in 2D-SFGrid window, to facilitate the user to view all the attributes of displaying objects by scrolling the slider.

Field Summary	
private int	**currentPosition** to access the current position of slider
private SFGrid	**sfGrid** sfGrid object of SFGrid, to access the functionalities of 2D Space-Filling Grid
private Slider	**topSlider** it is SWT slider control
Constructor Summary	
AttributeComparisonSlider(SFGrid sfGrid) default constructor, needs a instance of SFGrid	
Method Summary	
void	**put**(Shell shell) to put the slider object into 2D-SFGrid window

Table A.14. *AttributeComparisonSlider: Class Document*

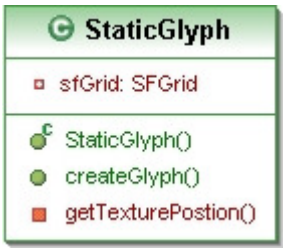


Figure A.16. StaticGlyph: Class Design Diagram

spacefillinggrid
Class StaticGlyph
`java.lang.Object`
`└ spacefillinggrid.StaticGlyph`

`public class StaticGlyph`
`extends java.lang.Object`

This class generates the static glyphs, each glyph object represent one attribute value in data object. This use the internal data structure to display the information of each attribute.

Field Summary	
private SFGrid	**sfGrid**
Constructor Summary	
StaticGlyph(SFGrid sfGrid) default constructor, needs instance of SFGrid	
Method Summary	
void	**createGlyph**(Composite composite) this method create and manage the glyph object, it use the canvas control, put in Composite control with other glyph objects.
private int	**getTexturePostion**(AttributeValueData avd)

Table A.15. *StaticGlyph: Class Document*

A. 5. Class Diagram

Class diagram defines as a static structure diagram in the Unified Modeling Language (UML), which describes the structure of a proposed system by showing the system's classes, their attributes (properties and methods), and the relationships between the classes.

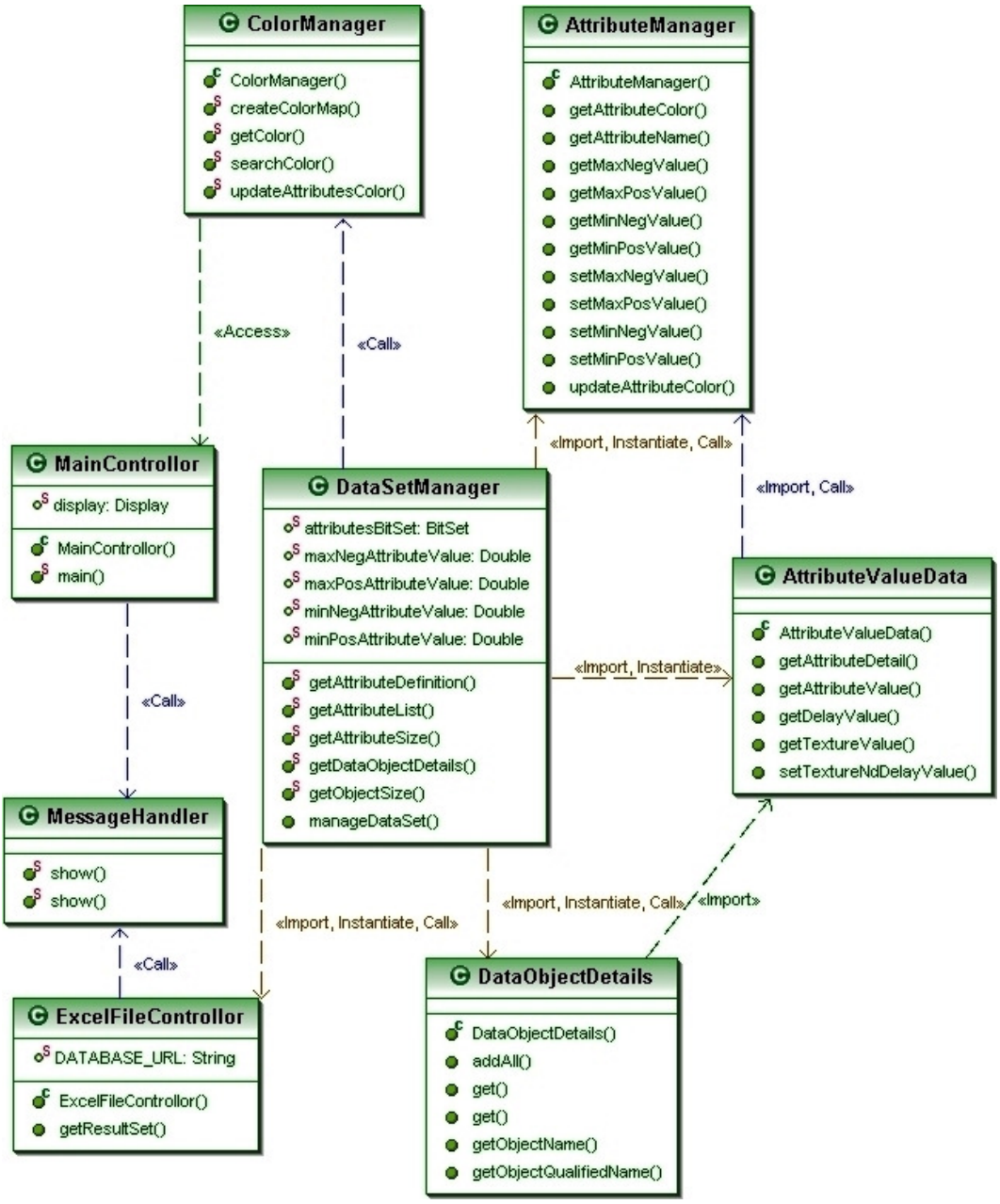


Figure A.17. Class Diagram: Dataset Loader

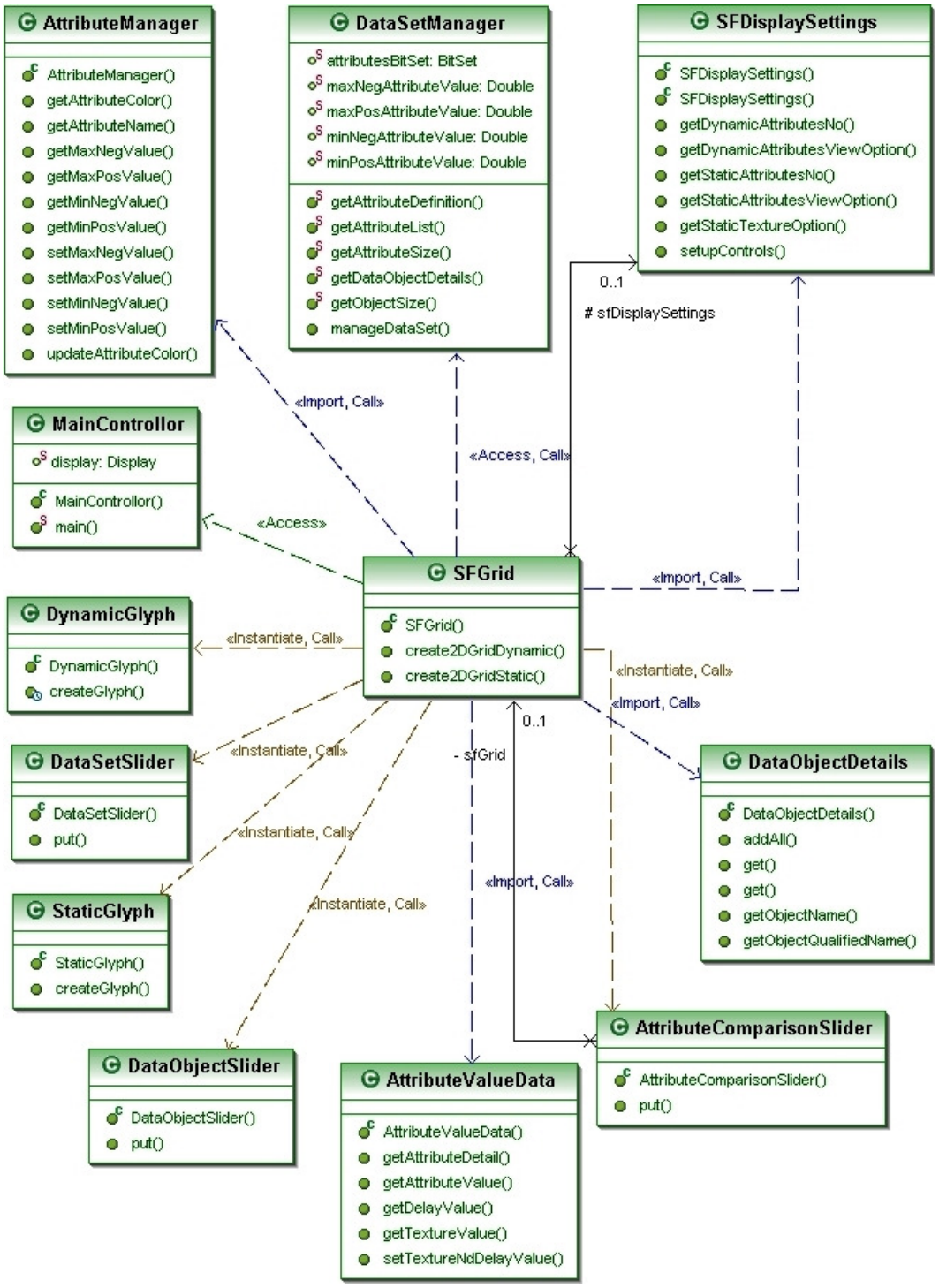


Figure A.18. Class Diagram: Visualization Process

Matematiska och systemtekniska institutionen
SE-351 95 Växjö

Tel. +46 (0)470 70 80 00, fax +46 (0)470 840 04
http://www.vxu.se/msi/