UniVis - a 3D software system visualization using natural metaphors

Dimitar Ivanov, Milena Lazarova, Haralambi Haralambiev and Delyan Lilov[1]

Abstract **—** The development of large software system frequently involves team participants with varying number and different level of competence. The understanding of the system structure and functionalities depends not only on the preliminary knowledge for the data, but also on its representation. The following paper presents UniVis - software visualization tool aiming to facilitate the processes of orientation and comprehension of complex software systems. By using natural and familiar metaphors, UniVis ensures the understandability of the visualized software system. The integrated navigation approaches attempt to ensure natural mechanism for manipulation of the result visualization by combining techniques for interaction with alternative effects on the visualization elements. The end product of the applied approaches is aesthetically appealing software visualization providing visually accessible amount of knowledge for the presented system.

Keywords **—** information visualization, software visualization, software comprehension, 3D.

I. Introduction

Every software developer is facing the challenges in getting acquainted with unfamiliar software system. Frequently, the effort needed to gain satisfying understanding of the structure and the functioning of the system is proportional to its size and design. Panas, Berrigan, and Grundy discusses the process of extension, reuse and support of industrial size software systems and present summarized statistics, claiming that 50% to 75% of the time and resources are invested in software comprehension as well as 47% to 62% of the time for actual correction and enhancement tasks is spent on comprehension activities. Moreover, the task of understanding the target system is qualified as the first step in the processes of software development and support [28]. These facts express the need of quality instruments for software comprehension.

Another shared problem in the process of software comprehension is its time consumption. Several works consider it as relevant in the context of getting acquainted with large software systems, due to their inadequate or sparse documentation ([28], [37], [33]).

Furthermore, Rilling and Mudur define reverse engineering as "the process of analyzing subject system components and their interrelationships to create a higher level of abstraction and to understand the program execution and the sequence in which it occurred" [36]. That definition emphasizes the linkage between the software systems comprehension and the reverse engineering, also discussed by Ramkumar and Indumathi [33]. This respectively gives a good reason for mutual usage shared approaches, which decrease the time interval needed to gain knowledge for the system.

In order to achieve deep understanding of given software system, we search for possible methods for its representation. Bonyuet, Ma, and Jaffrey suggest that the developers prefer to see the information for the software system visually instead of numerically [6]. We also believe that the best method for getting familiar with considerable amount of information is to visualize it adequately. According to Shneiderman the "users can scan, recognize and recall images rapidly" and detect visualization properties changes – in size, color, shape, movement, etc. [38].

The information needed to get acquainted with given information systems, varies according to the role, which the participants in a software project take. In general, the software engineers are interested in information about functional and nonfunctional issues of the system. The main aim of their tasks is modification of the system in order to improve it. That is why their target can be generalized in modifications, which will improve some aspect of the existing system [28] and gaining of initial perception of the software structure in order to communicate the development [41]. On the other hand, project managers have to focus their attention on system components with a global impact on the maintenance – hot spots in the system, where it is frequently modified and key places, where it can be restructured to obtain performance or reliability [28]. Software architects strive for identifying the structural and functional linkage between the system components. They also follow different indicators for the system quality – code metrics, resource usage diagrams, performance, etc. [5]. The focus of the

[1] Dimitar Ivanov is with the Applied Research and Development Center at Musala Soft, World Trade Center, 36, Dragan Tsankov blvd., 1113, Sofia, Bulgaria (e-mail: dimitar.ivanov@musala.com).

Milena Lazarova is with Systems Department, Technical University of Sofia, 8 Kliment Ohridski blvd, 1756, Sofia, Bulgaria (e-mail: milaz@tu-sofia.bg).

Haralambi Haralabiev is with the Applied Research and Development Center at Musala Soft, World Trade Center, 36, Dragan Tsankov blvd., 1113, Sofia, Bulgaria (e-mail: haralambi.haralambiev@musala.com).

Delyan Lilov is with the Applied Research and Development Center at Musala Soft, World Trade Center, 36, Dragan Tsankov blvd., 1113, Sofia, Bulgaria (e-mail: delyan.lilov@musala.com).

current work includes analysis of such requirements and identification of the most commonly needed features in order to build stable and usable visualization, appropriate for as much users as possible.

One of the popular works, concerning the information visualization defines a simple rule - "overview first, zoom and filter, then details on demand" [38]. The present article describes an abstraction and visual metaphors, used to achieve software visualization, applying this principle to a maximum extent. The abstraction is needed to overcome the details on displaying the source code and to improve the understandability of the software system [10]. The used metaphors are natural and publicly familiar, ensuring that every element of the software system has corresponding representation. The visualization uses innovative approach for visual clustering, achieved by using natural way of grouping elements into visual clusters – the bloom effect [17].

The above-mentioned properties of the target software visualization are applicable for either two or three dimensions. There are vast amount of publications, discussing the advantages and disadvantages of the data representations in two or three dimensions. Marcus, Feng, and Maletic analyze several works, concerning information visualization in 2D and 3D and in conclusion choose 3D visualization [25]. Rilling and Mudur emphasize the use of 3D space, providing shapes and other configurations, which help the users to link certain design features from the code with geometrical objects [36]. With regards to the aesthetic part of the visualization, Teyseyre and Campo claims that the inclusion of three dimensional aesthetically appealing elements can increase the intuitiveness, memorability and the whole human perception for the visualization [41]. Based on this research, the work presented in this paper uses three dimensional visualization as an environment for demonstrating different approaches for representation of given software system. We believe that the balance between the visually attractive and information richer software visualization can be achieved only in three dimensions.

The task of visualizing complex information requires significant visualization frameworks know-hows as well as mathematical skills. The work of Teyseyre and Campo summarizes considerable number of development tools, used for building three dimensional graphic applications [41]. Another point of view is presented by Satish and Raghuveera – they discuss the advantages of 3D over 2D from clearness and usability point of view [37].

The specifics of the selected visual metaphor and the bloom effect (see III.C.2)) calls for the use of development tools with more complete application programming interface (API), which provides easy integration with variety of 3D widgets [29]. That is why, this project develops software visualization system on OpenGL [39] as a multiplatform rendering framework for visualizations in two and three dimensions. Another obstacle, met in the development of complex visualization systems are the differences between the perception of the system users and its designers [47]. Working in the current context of software system visualization, the developers are familiar with the software development processes, which decrease the impact of such perception problems to minimum.

II. Related work

There are various approaches for representation of related data in the 3D space. In order to use the third dimension effectively, Rekimoto and Green present universal three dimensional visualization techniques for hierarchical information, called "The information cube" [35]. Gall, Jazayeri, and Riva represents the time in the third dimension of their visualization of software releases history [10]. Similar to this idea, Radfelder and Gogolla uses the 3D space to represent more complex and detailed sequence diagrams [31].

The present work aims to visualize software systems in three dimensions and to ensure the fast orientation and navigation by using defined metaphor. To illustrate the related literature better, we use the work of Rilling and Mudur, who divide the visualizations in static and dynamic [36]. The next two subsections include examples of visualizations, which uses abstractions and metaphors, similar to the presented in the current paper.

A. Dynamic visualizations

Greevy, Lanza, and Wysseier focus their analysis on combination of static and dynamic analysis of the system features. The third dimension is used as a supplement for interactive representation of the dynamic system information, providing the ability to explore the execution traces ([13], [14]). The authors apply static analysis to create the model of the source code and combine it with dynamic information for the basic runtime operations – object instantiations and message sends. The visualization reacts relevantly to the changes in the presented information by using animations (Fig 1). The navigation is implemented with the classical operations for pan, zoom and rotate. The work also presents proof of concept, visualizing real working software system, which gives the ability to focus on a user defined features of interest.

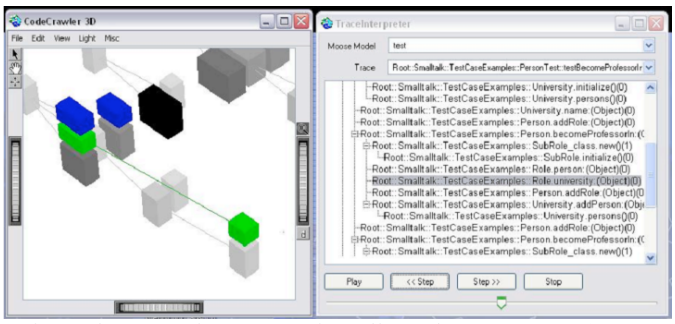

Fig 1. The Dynamic Feature View allows the user to step (source: [14])

Other authors try to use metaphors, similar to nature objects. Malloy and Power use spring embedding algorithm to aid the comprehension of given software system by visualization of object diagrams as molecules [24]. This approach is used in order to decrease the size of the object graph, extracted from Java applications. The needed information is taken by dynamically instrumenting the bytecode and collecting the trace data. As a final step, the trace data is analyzed and visualized in three dimensions using VRML (Fig 2).

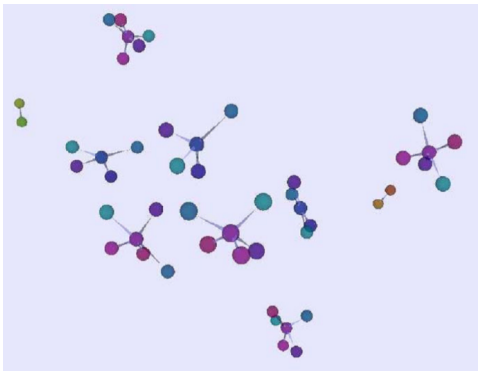

Fig 2. Full view of object model (source: [24])

In order to present another point of view in the software visualization, Rilling and Mudur experiment in applying metaballs metaphor for representation of software systems. The visualization presents software entities and their mutual influence (Fig 3), which form "a constantly moving micro-universe of entities (metaballs)" [36]. The approach gives the opportunity for dynamic alternation to the model program parameters as well as navigation through the representation for different purposes such as design evaluation, reverse engineering, testing, maintenance, etc.

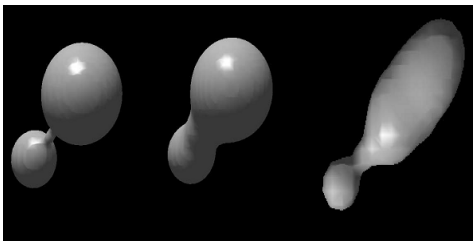

Fig 3. Metaballs in visualization of software interactions (source: [36])

B. Static visualizations

To use the benefits of the third dimension, Alfert and Fronk combine it with information, taken from the syntax graph, generated for given software system ([2], [3]). The authors describe several properties of the 3D space visualization – motion, transparency and positioning of the objects and use it appropriately to present the target system in an adequate view. The semantic grouping of the elements is achieved by the use of information cubes (Fig 4).

The metaphor described by Ploix is used to represent Lisp programs as a solar system. Such representation transfers "syntactic and behavioral components of a textual programming language (Lisp) to a graphical representation" [30]. The visualized solar system contains suns and planets, connected with directed links, which represent the direction of the calls between the functions. The "bottom up" evaluations are represented as orbits of the planets, which actually represents the functions. The additional data used by functions is represented as moons (Fig 5). Graham, Yang, and Berrigan present very similar idea for a solar system, but for representation of object oriented software system metrics [12].

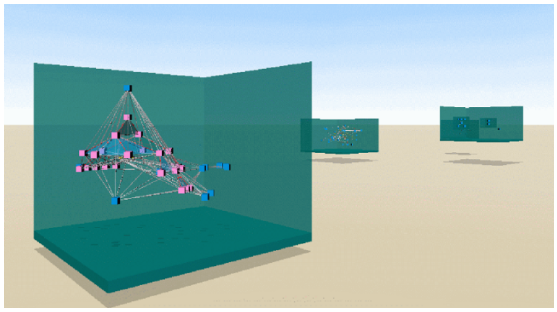

Fig 4. Information cubes, representing software system (source: [3])

Balzer and Deussen present more practical software system representation as a result from exploratory study for visualization of the static structure of real-world systems [4]. The landscape metaphor is used to present three dimensional images of the landscape elements, positioned with custom layouts. The links between the elements are represented as hierarchical connections, forming interconnection networks. The clustering of the elements is achieved by using transparent hemispheres, used to group semantically near elements (Fig 6). The navigation is facilitated by dynamic transparencies enabling the viewer to switch the detailed and the general views easily.

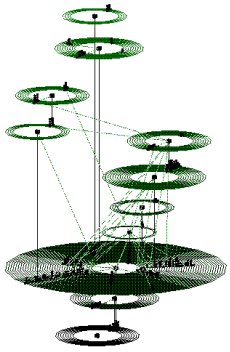

Fig 5. Solar systems of a lists drawing Lisp program (source: [30])

Code Mapping presents the software structure in a three dimensions using an atomic model – the elements are represented as spheres and the relations as lines [6]. The basic feature of the proposed visualization is the representation in virtual reality, combined with advanced user interactions (Fig 7). All the abstract information is extracted from the system using scanner and parser, which process the source code and provide the information to the visualization.

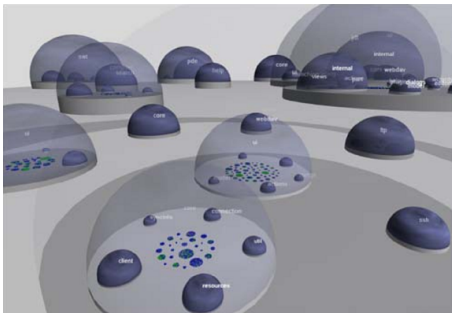

Fig 6. Hierarchy based 3D representation of a software system (source: [4])

Panas, Berrigan, and Grundy present clear and understandable metaphor for program visualization, concerning the static and dynamic aspects of the analyzed Java code [28]. The idea is to represent every package as a three dimensional city, whereas its components (the classes) are represented as its buildings. The authors describe this approach and how it helps the system maintainers and managers to decide the level of importance of every component of the system.

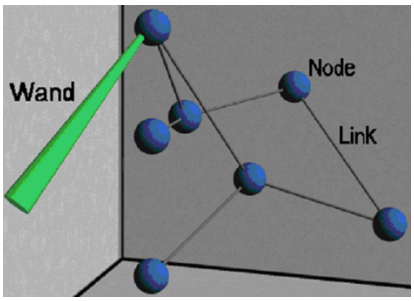

Fig 7. Atomic model in Code Mapping visualization (source: [6])

III. UniVis in General

Our analysis and works on providing comprehensible and clear representation of software system are integrated in UniVis – a prototype of software visualization tool. The general description of the basic aspects of UniVis adheres to the six key areas of interest proposed by Young and Munro [46].

A. Representation

The representation area concerns the graphical representation of the software system's components and its maximum information volume – how much information can be encoded in the representation. The representation is also defined as one of the most important aspects of the visualization.

According to Teyseyre and Campo the effective 3D visualization of graph-based representations should consider three main aspects [41]:

- positioning of the elements, achieved by usage of specific layout algorithm;
- comprehensible representation of the relationships;
- appropriate usage of clustering.

In order to adhere to these recommendations, UniVis integrates several alternative software system representations, containing the proposed three aspects. The elements' layout is customized in such a way to achieve semantic ordering. The links are represented as straight lines or curves, according to the metaphor used. The clustering is developed as a natural way of grouping elements into visual clusters – the bloom effect [17].

Marcus, Feng, and Maletic enumerate a number of representation forms (source code, tables, diagrams, etc.) and attributes (interactive, static, dynamic, etc.) of software visualization [25]. By using their proposal we identify the UniVis as a virtual world dynamic offline representation of software system using one abstract level and drill-down capabilities. UniVis visualization also experiments in constructing drawings of the system graph, using several metaphors borrowed from the nature and the surrounding world. The following sections describe the used metaphors and their elements in detail.

1) Space metaphors

The space metaphors used in UniVis borrows the visual concept from the widespread popular science materials, concerning the known space universe. The correspondence between the software systems and the universe is made visually and semantically (Fig 8).

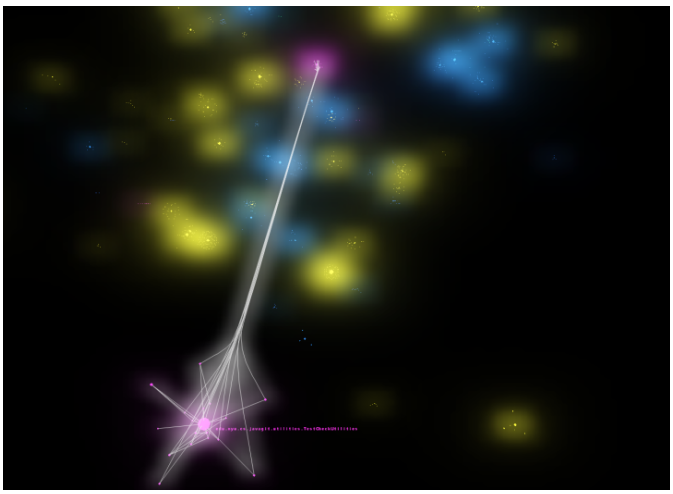

Fig 8. Software visualization in 3D using space metaphor

The semantic correspondence is straightforward - as the planets and their satellites form systems, the systems form galaxies, situated in the unlimited universe, so the statements form methods, the methods form classes, which are part of components, combined in a software system. To attain the ordering of the objects in the space, UniVis integrates custom dynamic layout algorithm, developed to adhere to the space concepts such as gravity or more generally - force-directed layout algorithm. Thus, the related elements are situated near to each other and distributed based on their semantics (e.g. the private methods are nearer to the class representation, whereas the public methods are in the outer orbits).

The visual correspondence between UniVis and the universe aims to achieve comprehensible clustering of the

elements, depending on the distance to the viewer. In real environment, such separation is formed naturally by the light of the space objects – emitted or reflected. This concept is developed in the UniVis as the bloom effect (see III.C.2)). The viewer perception is very close to a light, emitted from the objects, but in order to form distinguishing clusters, the radiance of the elements is artificially enlarged.

2) Geographic metaphors

In the context of orientation, the most popular association refers to the geographical objects and their mutual connections. If the third dimension is involved, the most reasonable metaphor appears to be the geographical globe. Such ideas are also inspired by the use of hyperbolic layouts for representation of connected data [21]. UniVis transfers these ideas to the subject of the software systems and represents them by using geographic metaphor - visualizing the whole software system, projected on the surface of an adequately sized sphere (Fig 9). The result visualization resembles to a three dimensional world map, displaying the cities and their road links.

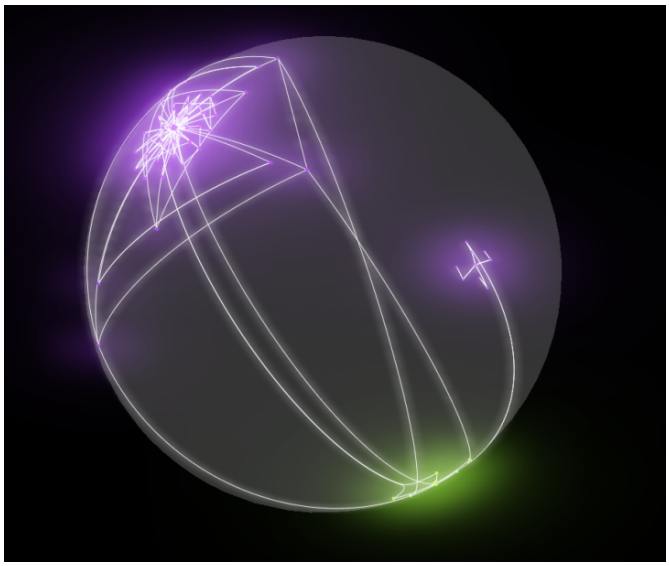

Fig 9. Software system, represented on a surface of a sphere

3) Combined metaphors

The improvement of the orientation in representation of given data is not only a question of aesthetically appealing visualization. The third dimension brings orientation problems, related to the absence of a starting point such as coordinate system and its center. Another problem in this context is the amount of information, needed to represent comprehensible coordinate system. In case of large graphs, such information can cause an information overload. To reduce the presented information as much as possible, UniVis also integrates different ordering of the visualization elements by applying a combination of the concepts, introduced in the space and geographic metaphors.

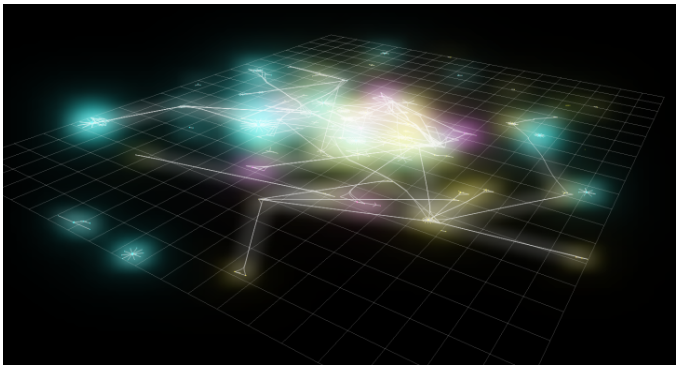

Fig 10. Software system, represented on a plane in the space

a) System plane

To facilitate the information representation and comprehension, UniVis uses simpler flat version of geographic metaphors, enriched with three dimensional graph nodes. The whole software system is presented as graph with three dimensional nodes and edges, situated on a single plane (Fig 10). Since the applied navigation is intended to operate in three dimensions the result visualization look like a 3D view of a geographic map.

b) Component planes

Washizaki, Takano, and Fukazawa introduce the idea, that the software maintainer understanding of the system is simplified to "collection of components" [43]. By taking the latter and the system plane as base ideas UniVis integrates an approach for representation of software system, which places the system components in separate planes (Fig 11). The planes are distributed in the space by following the same force-directed layout principles as for the atomic elements. To represent the relations clearly, UniVis uses an algorithm for force-directed edge bundling (see III.C.3)). The edges between the different components are gathered in bundles, representing general view of the components' relations.

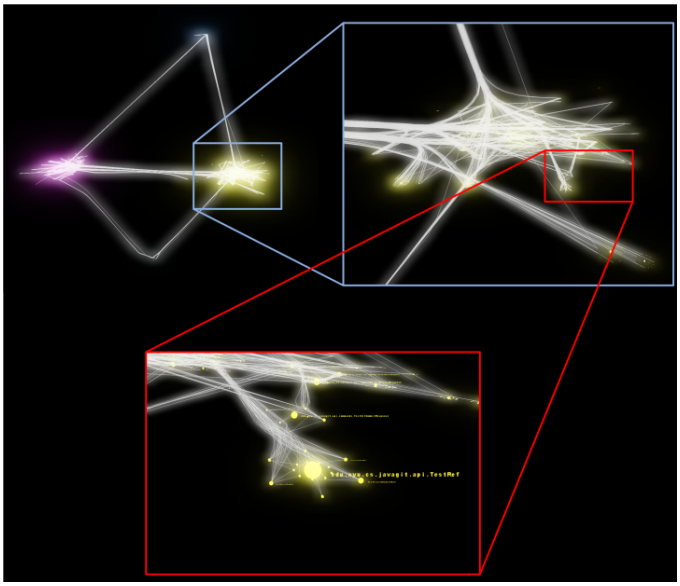

Fig 11. Software system components, situated in separate planes

The most powerful feature of the component planes approach is its ability to represent structures with unlimited level of depth. The methods can be applied as the complexity of the represented system grows or even on representation of multiple related systems.

B. Abstraction

The abstraction is the process of extracting the information away from the low level detail. In order to abstract the domain specific format of the data [23], reduce the mapping and interpretation load [32] and overcome the lack of standard "physical corpus" [2] of a software system, UniVis uses knowledge discovery meta-model [27]. The extraction of software system information from the source code is applied using the work of Yanakiev, Haralambiev, and Kraichev for gaining entity-relationship model of the source code [45]. The key advantage of this method is its independence from the used programming language, providing the ability to represent large number of software systems.

Generally, the abstraction level in the UniVis is high and the amount of details, available to the user is reduced to minimum in order to achieve better comprehension [36]. The visualization represents only top level code elements – classes, interfaces and methods, including their relations – extends, implements, uses, etc. All other details about the software system elements are shown on demand.

C. Navigation

The volume of the represented information presumes corresponding size of the visualization. The navigation is a set of approaches for guiding the users through the visualization without getting them lost or disoriented. The following section describes several methods, which was experimentally integrated in UniVis and intended to facilitate the navigation.

1) Semantic coloring

The number of the elements, contained in a software system requires a good approach for distinguishing them from a distance. There is wide variety of methods for altering the visualization appearance without affecting the source code [3]. The most noticeable properties of a visual element are its shape and its color. Aginsky and Tarr discuss the advantages and disadvantages of different methods, used to facilitate the visual search tasks [1]. Unfortunately, the shape is not considered reliable, whereas the color is described as "perceiving natural scenes" property of the target visual elements. Moreover, when using a shape as distinguishing property, the elements appear with reduced size from a distance and therefore - losing its shape outline. The distance reduces the size of the elements, but their color remains unchanged. That is why UniVis uses special mechanism of color determination in order to provide stable semantic coloring of the visualization elements.

The proposed approach for distinct color generation implements an algorithm for generation of distinct colors in HSV (hue, saturation, value) model and converting them in RGB (red, green, blue) model [40]. The basic idea is to generate visually close nuances for the semantically connected elements and distinguishing colors for the unrelated ones. The principle is to divide the color spectrum for the main components of the system and then – to split the result intervals according to the number of the owned elements of each component (Fig 12). The separation of the color space is developed differently according to the used model – the RGB space represents the color on a plane, whereas the HSV model uses a cylinder.

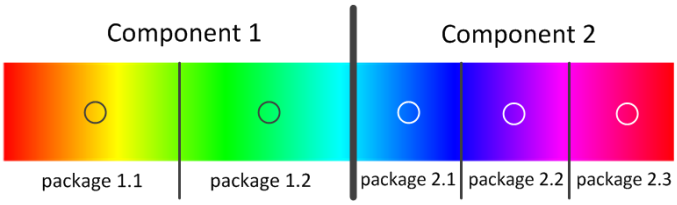

Fig 12. Visual illustration of the distinct color generation

The method of distinct color generation creates visual mapping between the responsibility of a system element and the color of its representation in the visualization. This approach tries to accelerate the process of full visual identification of an element by its representation.

2) Bloom effect

One of the key properties of UniVis is the used approach for semantic visual clustering – the bloom effect [17]. Since the described metaphors use the abstraction of light and its color as base characteristic of the elements, the bloom effect is used to artificially enhance the effect.

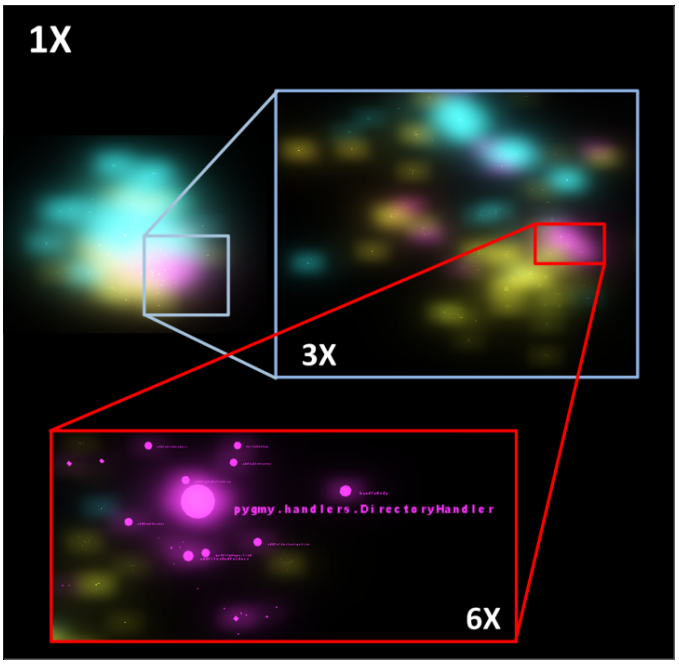

Fig 13. Illustration of the bloom effect in three scale coefficients

The size of the bloom is dynamically calculated according to the distance from the camera (the viewer) to the objects in the space. The used calculations provide a size of the bloom, which visually forms a kind of super nodes. These super

nodes represent group of ordinary nodes. Most frequently, the represented groups are the system components. This technique of hiding the elements when they are not needed is also called elision [29].

The advantage of the bloom effect approach is its unlimited level of depth. In other words, such representation of group of nodes can provide unlimited number of drill-downs and thus introducing the opportunity for visualization of single systems as well as groups of related systems.

The bloom effect also provides a resolution for the basic search task in collection of items having multiple attributes [38]. The grouping of the elements, satisfying values of a set of attributes is achieved visually with the color and the size of the bloom effect (Fig 13).

3) Edge bundling

The relations of the software system graph represent the linkages between the software system components, such as "calls", "uses type", "implements", etc. Their number usually grows exponentially and turns the graph into an incomprehensible tangle. The types of the relations also vary and the grouping of the edges by certain type doesn't give a satisfying result. In order to reduce the overload and provide clearer system visualization, UniVis applies force-directed edge bundling algorithm, introduced by Holten and Wijk [16].

The application of the bundling over the software system graph increases its readability in the general view (Fig 8, Fig 11) and also contributes to the natural form of some metaphors [36]. For example, when the bundling is applied over the edges of the graph, represented by using the geographic metaphor (Fig 9, Fig 10), the bundles resembles to a roads and highways between towns and cities.

The bundling of the edges also provides information about the semantics of the relations between the different system components. When the edge bundle is thick it is compounded of more relations and indicates tight coupling between the components, whereas the thin bundles represents loosely coupled system components.

Like the bloom effect, the bundling approach also provides a kind of unlimited drill-downs. The edge bundling algorithm is force-directed and therefore gathers the near edges together. Theoretically, when using the same algorithm for representation of related systems, the bundles should also be grouped into large bundles, illustrating the relations between the visualized systems.

4) Layout

The graph layout gets into the navigation section, since Parker, Franc, and Ware defines it as a non-spatial navigation together with the dynamic querying capabilities [29]. Teyseyre and Campo introduce different types of information visualization layouts – tree layout, cone layout, orthogonal layout, etc. [41]. In order to present the software system in the best way, the UniVis uses custom force-directed layout, developed by Iliev, Haralambiev, Lazarova, and Boychev [18]. All the metaphors use this layout, but each of them modifies its product before or after the layout application.

The space metaphors (see III.A.1)) use a three dimensional modified version of Iliev's layout and the nodes are placed in the space following the same concept as used in two dimensions - the methods are placed around the class in concentric spheres, the component are differentiated in clusters in the space (Fig 8). The edges of the graph are also situated in the space and bundled in order to avoid information overload.

The geographic metaphors (see III.A.2)) use the planar version of the layout and add post-processing step, consisted of inversed stereographic projection, which places the layout on the surface of a sphere. The curves of the edges are also following the sphere outlines, instead of crossing its interior.

The system plane approach (see III.A.3)a)) uses the original version of the layout and places all the nodes and relations of the graph in a single plane, which respectively means that the bundling algorithm is also used in its original version.

The component plane approach (see III.A.3)b)) applies the "divide and conquer" strategy and uses the original version of Iliev's layout in several separate planes, containing the representation of the components of the system (Fig 11). The planes are distributed in the space by following the same layout principles for three dimensions. The result representation gives a very good overview of the visualized system, against the expectations [15].

D. Correlation

The correlation is the linkage between the visualization and the represented information store. UniVis analyzes the current state of the source code and constructs entity-relationship model, describing it. The model is then represented as three dimensional graph, using different metaphors. Every element of the visualized graph has a reference to its corresponding source code. Unfortunately, the used model construction approach does not provide an opportunity for iterative building of the model and thus the modifications, made in the code, are not reflected in the visualization. The latter mean that the correlation in the current state of UniVis is at its base level and the development of this aspect should be included as future work (see VII.B).

E. Automation

The automation defines how much of the visualization is generated automatically without need of user interaction in the entire process. The effort needed to work with UniVis is reduced to minimum. The initial properties that must be set from the user are the location of the source code and the type of the view – standard or stereo. All the other properties of the visualization can be optionally controlled by the user – the type of interaction, used metaphor (respectively layout), elements shape and color, etc.

F. Interaction

Interaction concerns the quality and quantity of the needed user activities, in order to use the visualization sufficiently. The following paragraphs summarize the basic interaction issues, integrated into UniVis.

1) Input device interaction

Navigation varies according to the used approach for visualization. Herman, Melanc¸, and Marshall share the opinion, that the layout algorithm is not the only part of the visualization, needed to overcome the problems, raised by the large graphs [15]. That is why UniVis provides several interaction techniques, concerned with the used input devices.

If the system is visualized by using the system plane approach (see III.A.3)a)) the navigation features are focused on the plane, where the system representation is situated on. Thus, the interactions tend to manipulate only the target plane. On the other hand, when using the approaches which visualize the system in three dimensions (see III.A), the navigation is classic three dimensional – panning, zooming and rotation. These three basic interactions relies on the distance from the viewer to the target object and therefore it is dynamically calculated for the panning and the zooming, whereas the point of the rotation is defined by the currently selected element or from the center of the cuboid, bounding the whole layout.

UniVis also integrates alternative approaches for interaction in three dimensional environment. To simplify the navigation and use minimal number of input devices' data, the visualization is controlled by using the concepts of UniCam - interaction technique, described by Zeleznik and Forsberg [48]. This type of interaction uses the input from simple input devices, such as mouse or stylus. It manipulates the visualization according to defined rules, based on the position and the direction of the user movement actions on the input device.

2) Stereo viewing

Stereo view exploration of the data presents the information in different environment in order to use the perceptual senses of the user as an advantage [6]. This type of representation aims to facilitate the user in the exploration of three dimensional objects, such as software system graph, represented in the space (see III.A).

UniVis integrates simple type of stereo viewing of the software system visualization, using active shuttering glasses [8]. The technique of visualization uses two synchronized views, showing the visualization from slightly different points of view, imitating the eyes of the viewer. Using the stereo capabilities of the output devices, these two views are merged and explored via the active shuttering glasses. The interaction remains as in the three dimensional approaches.

3) Filtering

Besides the natural visual clustering and the edge bundling, UniVis provides additional ways for interaction with the nodes and the relations in order to improve the orientation in the system visualization. The integrated approaches are implemented as the most relevant features, needed in exploring a graph – full text search of a node by its name and switching (on and off) of the relations. The full text search accelerates the finding of the nodes by using the Patricia set data structure, based on the most popular Trie structure [34]. On the other hand, the switching of the relations helps the user to choose which relations should be visualized. The control of the switching is categorized by the type of the relation and by the incidence with given node (Fig 8).

IV. UniVis representation properties

Young and Munro define the representation as "a graphical (and other media) depiction of a single component" [46]. The following paragraphs describe in detail the representation level properties inherent to UniVis.

A. Individuality

Individuality concerns the unique representation of different parts of the visualization, according to their semantics. This representation property is achieved by using the fully qualified name of the source code elements and displaying it as a label, placed next to its corresponding visual element.

To avoid the information overload, the labels are shown in dynamically calculated size, corresponding to the dimensions of the whole system visualization and available only if their scaled size to the screen appears readable to the user. It is assumed that font sizes, less than 5 pt. are not readable.

The readability of the labels also appeared to be a problem for their visualization in the third dimension. To optimize the performance, UniVis uses the billboards technique – all the text labels are placed in plane, parallel to the image plane [9].

B. Distinctive appearance

Distinctive appearance concerns the contrasting, recognizable appearance of the representations. It contradicts with the low visual complexity (see IV.D). The most natural distinctive appearance is achieved by the color of the elements' representations.

Usually the coloring of the elements is used to indicate that given elements belongs to a certain group (component of the system). Xie, Poshyvanyk, and Marcus use predefined color interval to illustrate the similarities between vectors in Latent Semantic Indexing space, representing the source code and documentation of a software system [44]. UniVis integrates similar simplified approach for generation of distinct color nuances, according to the ownership information for every represented element (see III.C.1)).

C. High information content

The information content is appraised by the amount of information, provided through the representations. UniVis uses the size and the position of the elements to show certain

information about them

The real size of every representation is dynamically calculated by the lines of code, contained in its corresponding code element. UniVis provides the possibility to choose the resize mechanism of given element according to the measurements of different source code metric, but this feature is still on its development stage and is included in the future work (see VII.A).

The positioning of the elements is calculated by the layout algorithm and shown in the placement of the methods around their parent class (see III.C.4)). The same layout algorithm is used to arrange the representations on the next level of generalization, when the components are separated as single planes.

Other information, concerning the source code of the given visual element is available on demand by certain command in the context menu.

D. Low visual complexity

The visual complexity is related to the simplicity of the representation. UniVis visualizes the software system and its elements in the context of their interrelationships. Since the source code metrics or other properties of the elements are not relevant to such context, we can state that the visual complexity of UniVis is reduced to minimum.

V. UniVis visualization properties

Young and Munro define the visualization as "a collection or configuration of individual representations (and other information) which comprise a higher level component" [46]. The following paragraphs represent a detailed view over the visualization properties of UniVis.

A. Simple navigation

The navigation is significant part of the user orientation in the visualization. In order to keep the user perception for the visualized data, the navigation should be done adequately to the used visual metaphor.

As we described in the previous sections, UniVis integrates several techniques for navigation (see III.C), adaptable to the used visual metaphor, ensuring that the user will have the comfort to explore the data in the current context. More navigation features, needed for deep understanding of the explored system are included as future work (see VII.B).

B. High information content

The information content of visualization is the equivalent of the information content for the representation – it aims to decrease the information overload. Balzer and Deussen uses a layout, based on the hierarchy of the system elements (packages, classes, methods, etc.) and hemispheres for grouping them [4]. Similarly, UniVis uses the bloom effect and custom force-directed layout to group the elements by components and to define the color to represent their subcomponent.

C. Well-structured visualization

The visual complexity of the visualization depends on the representations, included in it, their relations and the structure of the visualization itself. To present adequate amount of information and decrease the visual complexity, UniVis uses the bloom effect and its interrelationship with the distance from the object to the viewer. When the distance to the viewer is increased (i.e. the visualization is zoomed out), the bloom size is also increased and the elements are "absorbed" by their own bloom, forming amorphous color cloud, representing group of elements (Fig 8, Fig 11). When the appropriate layout is used, this group represents whole system component. In the case with decreased distance to the viewer, the size of the bloom appears smaller and the elements – more distinguishable. Young and Munro also describe these methods as "scalability of visual complexity and information content" [46].

The number of the elements in the visualization is proportional to the size of the visualized software system. The large number of visualization elements increases the volume of the space, needed to display it. Thus, some of the elements appear very small from the viewer point of view. To reduce the information overload and speed up the performance, UniVis uses level-of-detail techniques, affecting the elements, situated to a great distance from the viewer.

D. Varying levels of detail

The levels of detail, presented in the different states of the visualization should be relevant to the user needs – the system overview should be shown for newcomers, while more detailed view will be visualized on demand. UniVis applies combination of standard methods and the bloom effect to achieve such variations in the displayed information. The used force-directed layout algorithm situates the nodes in semantic cluster, while their color is generated according to their place in the system structure. Finally, the bloom effect is applied to the result picture and the radiance of the near elements forms natural bloom clusters. These clusters are the elision technique [29], used to show only the needed information, corresponding to the user defined zoom scale.

E. Resilience to change

The resilience of the visualization is its ability to remain stable after changes on the presented information. Aiming at such stability, the most important part of the presented visualization approach is the choice of appropriate layout algorithm. A layout algorithm can be described as stable if it keeps the physical position of given graph elements, when their corresponding system elements are semantically changed. We also call this mental map preservation [20]. The best combination of aesthetically appealing and useful layout [41] is subjective and depends on the target information for the visualization. The applicability of Iliev's force-directed layout (see III.C.4)) for visualization of series of graph

drawings, representing software system, is explored with ReViewer [19]. The results show that the layout is stable and preserves users' mental map. Moreover, the Iliev's layout is suitable for "functional comparisons between program versions" [6].

F. Good use of visual metaphors

The metaphors, used for the visualization should be familiar for most of the target users. Ploix defines the metaphor as transition of knowledge between different domains, which uses the "intuitive knowledge of one domain to help the understanding of the other one" [30]. Inspired by the stars representation in Chrome experiments [7], the night Earth view from NASA [26] and other popular science materials of the space, the UniVis integrates the bloom effect as a basic element of all the used metaphors. The specific element of the metaphors borrows shapes, concepts and representations from well-known information representations – geographical map (see III.A.3)a)), globe (see III.A.2)), galaxies (see III.A.1)) and their combinations. These representations are popular among wide range of users, which means that their understandability will be satisfactory.

G. Approachable user interface

The interface is considered approachable, when it achieves a balance between the hardware used and the hardware popularity among the users. Since the software development is the target area for UniVis, the input devices and the user interface is considered standard. The basic interactions are achieved by using the mouse and the keyboard (see III.F). The user interface contains standard controls like sliders, dropdowns, context menus, etc.

H. Integration with other information sources

The integration concerns the linkage between the current data visualization and other types of its visualizations. UniVis visualizes the source code of software systems, written in standard Java. The alternative visualizations of several software data aspects are represented in the different Java IDEs. UniVis experiments to integrate the visualization in Eclipse [42]. The basic idea is to show the visualization of selected software projects on demand in internal views of the IDE. The development stage is still on its experimental phase and included in the future work (see VII.D).

I. Good use of interaction

The good use of interaction consists of variety of user interaction techniques which improve the user perception and maintain the interest. UniVis applies several techniques for better interaction. The first is the discussed bloom effect. Another one is the UniCam navigation, providing easy control by the use of one mouse button. The advantage of this technique is that it can be easily applied in the mobile context, where the user would be able to control the visualization only by using single finger gestures. The interaction technique, which needs additional user devices, is the stereo viewing of the visualization (see III.F.2)).

J. Suitability for automation

According to Young and Munro "a good level of automation is required in order to make the visualizations of any practical worth" [46]. To save time and provide easy to use visualization, UniVis encapsulates all the preliminary operations - the building of the abstraction, the analysis of the code elements and their relationships, etc. By implementing such mechanism separately, UniVis automates the process of construction of the visualization and respectively gives a method for easy visualization of single system or several related systems.

VI. Results and discussions

In spite of the presence of numerous tools for software visualization and comprehension, the developers, software architects and managers are still encountering problems in the analysis and understanding of unfamiliar software system. Similar problems also arise in the processes of bug fixing, refactoring, optimization, etc. These processes become more and more complicated with the need of systems with growing scope [33]. On one hand, the volume and complexity of the source code supporting a large scale software system increases with the addition of new functionalities. On the other hand, the process of software development is characterized with very high dynamics and the time, required to get acquainted with the whole system (or parts of it) needs to be decreased to minimum.

As a system for software visualization and comprehension, UniVis demonstrates and combines several experimental approaches for representation of software system. These approaches tend to provide full control over the represented information. Some of them give a general overview of the system, whereas others - detailed view of its components and their structure. The rest of the approaches use intuitive methods for information representation by using widespread and popular metaphors. As a result, the visualization can be generally categorized as multidimensional and adaptive.

A. Multidimensional

The multidimensional (or „layered") structure of the visualization appears in different aspect according to the metaphor, used to represent the system's source code and the attributes of the visualization's elements. Generally, these aspects can be divided into color and location.

1) Color

Frequently, the semantic coloring of the elements facilitates the orientation in the system representation. The package ownership in UniVis is shown by the color of every visualization element (see III.C.1)), while the used graph layout algorithm places the elements from given package close to each other. This leads to forming of different color

layers, which can be used for identification of the structural components of the system. Moreover, a problem in the system structure can be identified visually, by detection of specifically colored areas in the general view of the system.

2) Location

The layered structure is noticeable in the location distributions of the elements in the space and the groups which they form. The space metaphors rely on the third dimension and the layout algorithm to present the software as a number of different space systems whereas the software system is represented as a galaxy (see III.A.1)). The combined metaphors presumes that the user perceives the information better in a plane and presents the details of the lower level of abstraction in planes, distributed in three dimensions. On the other hand, to resemble real world objects and increase the understandability, the geographic metaphors present the software system as a planet with cities and highways.

The illustrated location aspects of the visualization are used to identify of the strength of the dependencies between the different system's components and there interconnections in a simple way, close to the real world.

B. Adaptive

The adaptability of the visualization is achieved with the use of approaches, realizing dynamic reactions to the given system information and its state. The following paragraphs discuss the results which makes UniVis an adaptive software visualization tool.

1) Universal

The universal description of all the proposed metaphors provides a mechanism for visualization of any data, described using entity-relationship model or graph structures. The methods are integrated into UniVis and their interoperability for this type of visualization is shown. The latter means that such combination of visualization methods can be applied in all the spheres of research, using entity-relationship abstractions for the data and more specifically – for software systems, written in various programming languages.

2) Visually clustered

As a key specific of the visualization, the bloom effect (see III.C.2)) adapts it to the user interaction – when the distance between the view point and the displayed objects varies, the visible information is filtered visually by dynamically changing the bloom size. The result radiances cover the detailed information and only the most significant parts of it remains. These visualization characteristics accomplish the initial goal to the software system representation, following the principle "overview first, zoom and filter, then details on demand" [38].

3) Stable

The dynamic processes of software system development produces frequent changes in the source code. To maintain the visualization up-to-date and keep the user perception for the entire system representation, UniVis integrates a change resilient force-directed layout algorithm (see V.E). As result the visualization preserves its state according to the changes in the source code. By using this valuable visualization quality, the user can easily identify structural problems in the system only by observing the translations between the visual elements.

VII. Future work

The future work on UniVis can be categorized in four main aspects – the adjustment of the representation and visualization attributes according to semantic qualities of the code; visualization supplements for dynamic simulations, representing the changes in the visualized system; optimizations for ensuring better capacity for the visualization; integration of UniVis with other information sources and systems.

A. Semantic adjustment

The node sizes of the system graph are currently determined proportionally to simple source code metric – the lines of code (see IV.C). The future work in this aspect of the visualization includes a mechanism for node size determination, relying on other software metrics [22].

UniVis also provides a linkage between the visualization elements and their code snippets (see IV.C). The reversed linkage – from the source code to the visualization elements, is planned for future work and will be used as a base for the realization of the dynamic simulations (see VII.B)

The component planes visualization approach (see III.A.3)b)) relies on the preliminary defined system components. To the current moment, this component identification is based only to the package structure of the system. The work of Garlan and Shaw [11] inspires the identification of system components and should be beneficial for extracting the basic characteristics of common types of software architecture and respectively - the identification of the components, contained in the certain type of architecture.

B. Dynamic simulations

Several authors present visualizations, which show the system activities on their execution or through dynamic visualizations of the static data ([30], [36], [38]). The used visualization gives the possibilities for integration of similar approaches by using the metaphor of light. The bloom effect brightness and size can be used to point key changes in the system and apply approaches such as animation of the program flow between the elements' representations, application of visual effects over the nodes, changed in the consequent code revisions. The latter future work objective concerns the UniVis ability to develop a method, which provides iterative building of the abstract system model.

C. Optimization

The size of the industrial software systems increases and appears as a great impediment for the tools such as UniVis.

The visualization of thousands of elements with a good performance is inconceivable without the use of specialized optimizations. UniVis uses several techniques to speed up the performance – level-of-detail for the distant objects (see V.C), billboards for the elements' labels (see IV.A) and display lists for the edges curves. The future work on the optimizations concerns experiments with point sprites, vertex buffer objects and geometry instancing [39].

D. Integration

As a tool, targeted to the software developers, UniVis should be convenient to use as a part of the development process. The integration of the OpenGL view in the Eclipse IDE is initiated (see V.H) and the future work in this aspect concerns the mutual work of UniVis, the IDE and the used versioning system.

ACKNOWLEDGMENT

This work was partially supported by the Bulgarian Science Fund through contract ДМУ 02/18 – 2009 "Fast Orientation in Complex Information Systems".

REFERENCES

[1] V. Aginsky, M. J. Tarr, How Are Different Properties of a Scene Encoded in Visual Memory?, Journal Visual Cognition, volume 7, pages 200 - ?, 2000.

[2] K. Alfert, A. Fronk, 3-Dimensional Visualization Of Java Class Relations, in proc. of the Fifth World Conference on Integrated Design & Process Technology, Dallas, Texas, 2000.

[3] K. Alfert, A.Fronk, Manipulation of 3-dimensional Visualizations of Java Class Relations, In proc. of Integrated Design and Process Technology, IDPT-2002, USA, 2002.

[4] M. Balzer, O. Deussen, Hierarchy Based 3D Visualization of Large Software Structures, in proc. of IEEE Conf. Visualization (VIS '04), p. 598.4, 2004.

[5] H. Byelas, A. Telea, Visualization of areas of interest in software architecture diagrams, in proc. of ACM symposium on Software visualization (SoftVis '06), pp. 105 – 114, New York, USA, 2006.

[6] D. Bonyuet, M. Ma, K. Jaffrey, 3D Visualization for Software Development, in proc. of IEEE Int'l Conf. Web Services (ICWS '04), p. 708, 2004.

[7] Chrome Experiments, Stars, retrieved from http://workshop.chromeexperiments.com/stars/, Google, 2012

[8] A. Craig, W. R. Sherman, J. D. Will, Developing Virtual Reality Applications: Foundations of Effective Design, Developing Virtual Reality Applications: Foundations of Effective Design, Morgan Kaufmann Publishers Inc., USA 2009.

[9] X. Décoret, F. Durand, F. X. Sillion, J. Dorsey, Billboard clouds for extreme model simplification, in proc. of SIGGRAPH '03, pp 689 – pp 696, New York, NY, USA, 2003.

[10] H. Gall, M. Jazayeri, C. Riva, Visualizing Software Release Histories: The Use Of Color And Third Dimension, in proc. of the IEEE International Conference on Software Maintenance, ICSM '99, p. 99, Washington, DC, USA, 1999.

[11] D. Garlan, M. Shaw, An Introduction to Software Architecture, School of Computer Science, Carnegie Mellon University, Pittsburgh, January, 1994.

[12] H. Graham, H.Y. Yang, R. Berrigan, A Solar System Metaphor for 3D Visualisation of Object Oriented Software Metrics, in proc. of Australasian Symp. Information Visualisation, pp. 53-59, 2004.

[13] O. Greevy, M. Lanza, C. Wysseier, Visualizing Feature Interaction in 3-D, in proc. of third IEEE Int'l Workshop Visualizing Software for Understanding and Analysis (VISSOFT '05), p. 30, 2005.

[14] O. Greevy, M. Lanza, C. Wysseier, Visualizing Live Software Systems in 3D, in proc of ACM Symp. Software Visualization (SoftVis '06), pp. 47-56, 2006.

[15] Herman, G. Melanc¸ M.S. Marshall, Graph Visualization and Navigation in Information Visualization: A Survey, IEEE Trans. Visualization and Computer Graphics,vol. 6, no. 1, pp. 24-43, 2000.

[16] D. Holten, Jarke J. van Wijk, Force-Directed Edge Bundling for Graph Visualization, Eurographics/ IEEE-VGTC Symposium on Visualization, Vol 28, number 3, Berlin, Germany, 2009.

[17] D. Hubanova, H. Haralambiev, M. Lazarova, S. Boychev, Dynamic Visual Clustering Using Bloom Effect, Sixth International Scientific Conference Computer Science, Ohrid, Macedonia,pp. 308 – 311, 2011.

[18] I. Iliev, H. Haralambiev, M. Lazarova, S. Boychev, Dynamic Force-Directed Graph Layout for Software Visualization, International Scientific Conference on Information, Communication and Energy Systems and Technologies, Volume 3, pp. 885 – 888, Nis, Serbia, 2011.

[19] D. Ivanov, H. Haralambiev, M. Lazarova, S. Boychev, Quality Assessment of Graph Drawing Sequences Representing Software Systems Evolution, Journal Information Technologies and Control, Year VIII, No.4, pp.20–29, 2011.

[20] D. Ivanov, M. Lazarova, H. Haralambiev, D. Lilov, Visual Identification of Mental Map Anomalies in Graph Drawing Algorithms, in proc. of Automatics and Informatics, pp. 267 – pp. 270, Sofia, Bulgaria, 2012.

[21] J. Lamping, R. Rao, P. Pirolli, A Focus + context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies, Human Factors in Computing Systems, CHI '95 Conference Proceedings, ACM Press, 1995.

[22] M. Lanza and R. Marinescu, Object-Oriented Metrics in Practice, first edition, Berlin: Springer Verlag, 2006.

[23] J. Maletic, A. Marcus , M. L. Collard, A Task Oriented View of Software Visualization, In proc. of the 1st International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT '02), p.32, Washington, DC, USA, 2002.

[24] B.A. Malloy, J.F. Power, Using a Molecular Metaphor to Facilitate Comprehension of 3D Object Diagrams, in proc. of IEEE Symp. Visual Languages and Human-Centric Computing (VL/HCC '05), pp. 233-240, 2005.

[25] A. Marcus, L. Feng, J. Maletic, 3D Representations for Software Visualization, In proc. of SoftVis '03, Pages 27-ff, New York, USA, 2003.

[26] National Aeronautics and Space Administration, Earth city lights, retrieved from http://visibleearth.nasa.gov/view.php?id=55167, Visible Earth - a catalog of NASA images and animations of our home planet.

[27] Object Management Group, retrieved from http://www.omg.org/spec/KDM/1.0/, Knowledge Discovery Meta-Model, January 2008.

[28] T. Panas, R. Berrigan, J. Grundy, A 3D Metaphor for Software Production Visualization, In proc. of the Seventh International Conference on Information Visualization, IV '03, pp.314, Washington, DC, USA, 2003.

[29] G. Parker, G. Franck, C. Ware, Visualization of Large Nested Graphs in 3D: Navigation and Interaction, J. Visual Languages and Computing,vol. 9, no. 3, pp. 299-317, 1998.

[30] D. Ploix, Building Program Metaphors, retrieved from http://damien.ploix.free.fr/papiers/ppig96/ppig96.html, October 2008.

[31] O. Radfelder, M. Gogolla, On Better Understanding UML Diagrams through Interactive Three-Dimensional Visualization and Animation, in proc. of Working Conf. Advanced Visual Interfaces (AVI '00),pp. 292-295, 2000.

[32] H. Ramadhan, Z. Al-Khanjari, H. Al-Lawati, Design and Evaluation of a Visual Framework for Facilitating Re-engineering and Re-use of Relational Databases, in proc. of International Conference of Automation and Information , pp. 342 – pp. 349, Spain, 2003.

[33] K. Ramkumar, J. Indumathi, A framework for architecture recovery of web applications, in proc. of 4th WSEAS International Conference on Software Engineering, Parallel & Distributed Systems, Article No 1, Stevens Point, Wisconsin, USA, 2005.

[34] T. U. Rehman, retrieved from http://www.badgenow.com/, February, 2013.

[35] J. Rekimoto, M. Green, The Information Cube: Using Transparency in 3D Information Visualization, Proc. Third Ann. Workshop Information Technologies and Systems,pp. 125-132, 1993.

[36] J. Rilling, S. P. Mudur, On the Use of Metaballs to Visually Map Source Code Structures and Analysis Results onto 3D Space, In proc. of the Ninth Working Conference on Reverse Engineering, WCRE '02, pp.299, Washington, DC, USA, 2002.

[37] C. J. Satish, T. Raghuveera, Visualizing object oriented software using virtual worlds, in proc. of 4th WSEAS International Conference on Software Engineering, Parallel & Distributed Systems, Article no. 3, Stevens Point, Wisconsin, USA, 2005.

[38] B. Shneiderman, The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations, In proc. of IEEE Symposium on Visual Languages, pp. 336, Washington, DC, USA, 1996.
[39] Silicon Graphics, Inc., OpenGL, retrieved from http://www.opengl.org
[40] R. Smith, Color gamut transform pairs, Comput. Graph., pp.12 – pp.19, 1978.
[41] R. Teyseyre, M. R. Campo, An Overview of 3D Software Visualization, IEEE Transactions on Visualization and Computer Graphics , Volume 15 Issue 1, pp. 87 – pp.105, Piscataway, NJ, USA, 2009.
[42] The Eclipse Foundation, Eclipse, retrieved from http://www.eclipse.org/
[43] H. Washizaki, S. Takano, Y. Fukazawa, Visualization of binary component-based program structure with component functional size, in proc. of 5th WSEAS international conference on Applied computer science, pp. 911 – pp. 917, Stevens Point, Wisconsin, USA, 2006.
[44] X. Xie, D. Poshyvanyk, A. Marcus, 3D Visualization for Concept Location in Source Code, In proc. of the 28th international conference on Software engineering, ICSE '06, pp.839 – pp.842, New York, NY, USA, 2006.
[45] A. Yanakiev, H. Haralambiev, K. Kraichev, Revisiting Abstract Syntax Tree as a Basis of Source Code Knowledge Models, in proc. of Informatics in the Scientific Knowledge, pp.242 – 257, Varna, Bulgaria, 2012
[46] P. Young, M. Munro, Visualizing Software in Virtual Reality, in proc. of Sixth Int'l Workshop Program Comprehension (IWPC '98), p. 19, 1998.
[47] I. Zayour, T. C. Lethbridge, A Cognitive and User Centric Based Approach For Reverse Engineering Tool Design, in proc. of the conference of the Centre for Advanced Studies on Collaborative research, CASCON '00, pp. 16, IBM Press, 2000.
[48] R. Zeleznik, A. Forsberg, UniCam - 2D Gestural Camera Controls for 3D Environments, In proc. of the 1999 symposium on Interactive 3D graphics (I3D '99), pp. 169 – pp. 173, New York, NY, USA, 1999.

Dimitar S. Ivanov was born in Vidin, Bulgaria on 04.07.1987. He graduated "Ekzarh Antim I" high school of mathematics and science (2006) and continued his education as bachelor's degree of informatics (2010) in the Sofia University "St. Kliment Ohridski". He also finished his master's degree in "E-Management and E-business" (2012) in Sofia University "St. Kliment Ohridski". At the current moment D. Ivanov studies his PhD on "Automated Systems for Information Processing and Management" in Technical University of Sofia.

He started his career as JUNIOR SOFTWARE ENGINEER in the Applied Research and Development Center at Musala Soft (September 2009 – October 2010) and continued as SOFTWARE ENGINEER (October 2010 – February 2013). In the current moment he is SENIOR SOFTWARE ENGINEER and coordinates the research process in the Applied Research and Development Center at Musala Soft, Sofia, Bulgaria. He has several publications in conferences and journals including:

- Visual Identification of Mental Map Anomalies in Graph Drawing Algorithms (D. Ivanov, M. Lazarova, H. Haralambiev, D. Lilov, in proc. of International Conference of Automatics and Informatics, pp. 267 – pp. 270, Sofia, Bulgaria, 2012);
- Quality Assessment of Graph Drawing Sequences Representing Software Systems Evolution (D. Ivanov, H. Haralambiev, M. Lazarova, S. Boychev, Journal Information Technologies and Control, Year VIII, No.4, pp.20–29, 2011);
- ReViewer: tool for monitoring and analysis of graph images (D. Ivanov, H. Haralambiev, M. Lazarova, S. Boychev, in proc. of International Conference of Automatics and Informatics, Sofia, Bulgaria, 2011).

Assistive Prof. Dr. Milena Lazarova is a promising young scientist in the Faculty "Computer systems and control" (FCSC) at Technical University of Sofia. She is first prize winner for the project "System for discovering disasters" on educational science Expo 2008, as well as bearer of a Batch of Honor of the state Agency for Information Technologies and Communications. In the last two years she is a software projects leader with which students from FCSC at Technical University of Sofia participate in the "Imagine Cup" competition organized by Microsoft. Two years in a row (2008 and 2009) the project leaded by her are winners in competitions on national level and represent Bulgaria on the world finals in the category "Software Design". She is a leader of the internal-university contract funded by SRS, Technical University of Sofia. She has several publications in conferences and journals including:

- Visual Identification of Mental Map Anomalies in Graph Drawing Algorithms (D. Ivanov, M. Lazarova, H. Haralambiev, D. Lilov, in proc. of International Conference of Automatics and Informatics, pp. 267 – pp. 270, Sofia, Bulgaria, 2012);
- Quality Assessment of Graph Drawing Sequences Representing Software Systems Evolution (D. Ivanov, H. Haralambiev, M. Lazarova, S. Boychev, Journal Information Technologies and Control, Year VIII, No.4, pp.20–29, 2011);
- ReViewer: tool for monitoring and analysis of graph images (D. Ivanov, H. Haralambiev, M. Lazarova, S. Boychev, in proc. of International Conference of Automatics and Informatics, Sofia, Bulgaria, 2011).

Haralambi K. Haralambiev was born in Burgas, Bulgaria on 19.12.1985. He graduated "Akad. Nikola Obreshkov" high school of mathematics and science (2004) and continued his education with the bachelor's degree of computer science in Sofia University "St. Kliment Ohridski" (2008). Several years later he also finished his master's degree on "Discrete and algebraic structures" (2013).

His career started as JUNIOR SOFTWARE ENGINEER in Musala Soft (July 2005 – September 2005), continued as SOFTWARE ENGINEER (October 2005 – January 2007), SENIOR SOFTWARE ENGINEER (February 2007 – May 2008) and TEAM LEAD (June 2008 – October 2011) at Musala Soft. In the current moment he is TECHNICAL STAFF DEVELOPMENT MANAGER and coordinates the application and integration of the processes in Musala Soft, Sofia, Bulgaria. He has several publications in conferences and journals including:

- Visual Identification of Mental Map Anomalies in Graph Drawing Algorithms (D. Ivanov, M. Lazarova, H. Haralambiev, D. Lilov, in proc. of International Conference of Automatics and Informatics, pp. 267 – pp. 270, Sofia, Bulgaria, 2012);
- Dynamic Force-Directed Graph Layout for Software Visualization (I. Iliev, H. Haralambiev, M. Lazarova, S. Boychev, International Scientific Conference on Information, Communication and Energy Systems and Technologies, Volume 3, pp. 885 – 888, Nis, Serbia, 2011.);
- Applying source code analysis techniques. A case study for a large mission-critical software system (H. Haralambiev, S. Boychev, D. Lilov, in proc. of EUROCON, April 27-29, 2011).

Delyan Lilov was born in Sofia, Bulgaria on 26.11.1969. He has a master degree in informatics from Sofia University "St. Kliment Ohridski" (1994).

He is founder, CEO and a shareholder in one of the most successful Bulgarian software companies - Musala Soft. In nine years, due to his extensive experience, professionalism and vision, Musala Soft expanded and positioned as a reliable and prestigious long term partner for world industry leaders such as mag, EnBW, IBM, HP, SAP and others. As a CEO Mr. Lilov defines company strategy, leads operations and is involved in key accounts. Commitment to quality, continuous process improvement and mainstay of company teams are among his core areas of activity. Mr. Lilov holds a MA in Computer Science. He is a leader from the Musala Soft side of the Unified Science Group on SCAM (Software Code Analysis and) of SU-FMI and Musala Soft. He has several publications in conferences and journals including:

- Visual Identification of Mental Map Anomalies in Graph Drawing Algorithms (D. Ivanov, M. Lazarova, H. Haralambiev, D. Lilov, in proc. of International Conference of Automatics and Informatics, pp. 267 – pp. 270, Sofia, Bulgaria, 2012);
- Applying source code analysis techniques. A case study for a large mission-critical software system (H. Haralambiev, S. Boychev, D. Lilov, in proc. of EUROCON, April 27-29, 2011);
- Improving software architecture through circular dependency detection (B. Strandjev, H. Haralambiev, D. Lilov, in proc. of International Conference Automatics and Informatics, October, 2010).