Vol.2016-SE-194 No.1
2016/11/17

3次元空間におけるプログラムの動的可視化

大神 勝也[1,a)]　中川 尊雄[1,b)]　畑 秀明[1,c)]　松本 健一[1,d)]

概要：プログラムの可視化は，複雑なプログラムに対する理解や問題個所の抽出を容易に行うために有用な技術である．特に，3 次元空間における可視化は 2 次元空間の場合より多くの情報を同時に表現することが可能であるため，プログラム実行の際に得られる複雑な情報の可視化が期待できる．本研究では，プログラムの動的情報を 3 次元空間上でリアルタイムに可視化する手法を提案した．提案手法は Java プログラムを可視化の対象としており，統合開発環境 Eclipse のプラグインとして実装した．既存の統合開発環境が備えているブレークポイントやステップ実行といったデバッグ機能を使用しながらリアルタイムに可視化できることが本ツールの特徴である．

キーワード：ソフトウェア可視化，動的可視化，リバースエンジニアリング，コードの街，コールグラフ

Dynamic Program Visualization in 3D Space

Katsuya Ogami[1,a)]　Takao Nakagawa[1,b)]　Hideaki Hata[1,c)]　Kenichi Matsumoto[1,d)]

Abstract: Software visualization is a useful technique to understand complex software and identify problems. Compared to the 2D visualization, the visualization in 3D space is expected to present more information. In this study, we propose a technique of dynamic program visualization in real time. We implemented this technique as an Eclipse plugin, which targets Java programs. It can visualize Java programs in real time while using debug system (e.g. breakpoint, step execution) of existing Integrated Development Environments.

Keywords: Software visualization, Dynamic visualization, Reverse engineering, Code city, Call graph

1. はじめに

自身の携わるプログラムの構造やふるまいを理解することは，ソフトウェア開発者にとって重要かつ大きな労力を要する作業である．たとえば，新規にプロジェクトに参入した開発者は，開発対象プログラムについて，一から早急に理解を進める必要がある．また，開発者が既にプログラムの内容を理解しているとしても，プログラムは開発の過程で徐々に変化していくため，その状況を常に把握している必要がある．

このような理解作業を少しでも容易にするため，複雑なプログラムの構造を図やツールによって可視化することが広く行われている．最も単純な例としては UML におけるクラス図・シーケンス図がこれに相当する．また，ユーザの入力に応じて図の上に情報をオーバーラップさせたり，図を変化させるなどの，インタラクティブな可視化ツールの開発も行われている．一般的にこうした可視化ツールの多くは，プログラムを実行せずに得られる静的情報の可視化を対象としている．

一方で，プログラムの実行時に得られる情報も，開発者にとって有用である．例として，特定の処理が呼び出されるタイミングや，データの具体的な入出力などがあり，静的な情報からは取得困難なものも多く含まれる．しかし，動的情報は操作や入力の時系列に依存する複雑な情報であり，単純な可視化を与えることが難しい．開発者は従来，

[1] 奈良先端科学技術大学院大学情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma, Nara 630-0192, Japan
[a)] ogami.katsuya.ny7@is.naist.jp
[b)] takao-n@is.naist.jp
[c)] hata@is.naist.jp
[d)] matumoto@is.naist.jp

目的の情報を得るために，難解な実行ログを読んだりプログラムを手動でステップ実行するなど，根気と時間を要する作業を求められてきた．

そこで我々は，複雑な動的情報に対する効果的な可視化の実現を目指し，プログラムの実行と並行して可視化図がまさに動的に変化するような新たな可視化手法を提案する．提案手法は，静的情報の可視化で多く見られる 2 次元図ではなく，3D グラフィックのリアルタイムレンダリングを用いることで表現可能な情報量を増やし，Java プログラムの実行と同期してプログラムの呼び出し関係等の情報を可視化するものである．また本手法は，プログラムを “コードの街” として可視化することによって，プログラムの構造，クラスやメソッドの特徴を一目で把握することを可能としている．コードの街は CodeCity [1] や EvoSpaces [2] で確立され，昨今ではプログラム可視化の研究において活発に用いられている．

本論文では，提案手法の詳細，ならびに提案手法を実装したツール RocatDebugger[*1] の設計や，今後の課題について述べる．

本論文の構成は次の通りである．第 2 章では，従来の動的可視化の問題，また新しい動的可視化に望まれる要件について述べる．第 3 章では，本手法を実装したツール RocatDebugger の概要について述べる．第 4 章では，RocatDebugger の設計について述べる．第 5 章では，プログラムの動的可視化に関する研究，プログラムの 3 次元空間可視化に関する研究について紹介する．第 6 章では，まとめと今後の課題について述べる．

2. プログラムの動的可視化

最も基本的，かつ現在でも広く使用されている動的情報の獲得方法として，メソッドの冒頭にプリント関数を仕込むなどして文字列をコンソールに表示させるものがある．この方法を用いると，開発者はプログラムを操作しながら，特定のメソッドが呼び出されたタイミングを確認することができる．しかしながら，この手法が有効であるのは，開発者がプログラム構造についてある程度の前提知識を持つような場合に限られる．なぜなら，呼び出され得るメソッドについて予想できなければ，そもそもプリント関数を埋め込むべき位置もわからないからである．また，仮にすべてのメソッドに文字列を出力させたとしても，メソッド名などから機能を類推することが難しく，有用とはいえない．これはプリント関数を利用した方法に限らず，統合開発環境のデバッグ機能を使用してブレークポイントを仕込む場合も同様に前提知識を要すると考えられる．

すなわち，あらかじめプログラム中の特定の場所に目星をつけて動的情報を検査するためには，プログラムの構造やふるまいに関するおおまかな知識を事前に獲得する必要がある．一般に，プログラムの大まかな構造を知るために，ドキュメントを参照したり，パッケージ，クラス，メソッドの名前などを参考にできるが，こうした情報が常に適切に提供されているとは限らない．そこで，本研究においては，プログラムの構造とプログラムの動作を同時にわかりやすく可視化することで，開発者にプログラムの構造やふるまいについての事前情報を提供することを目指す．

プログラムの全体像を表示し，プログラムの処理の間にアクティブになったクラスやメソッドを全体像の中から提示できれば，開発者はコード調査の開始地点を定められるようになる．この足掛かりを得てはじめて開発者は冒頭で述べたような事前知識を得て，より詳細な解析にうつることができると考えられる．そのため，可視化ツールはデバッグ機能と密接に利用できることが好ましい．

また，LOC (コード行数) や CC (循環的複雑度) といったメトリクスは，クラスやメソッドの特性を知るために有用である．例えば，LOC はクラスやメソッドの規模に深く関わるメトリクスであり，可視化によって LOC を視覚的に比較すれば，開発者は最も規模の大きいクラスを一目で発見することができる．こういった情報もまた開発者がプログラムを理解するための助けとなるので，動的可視化においてはこれらの情報も含めて表現したい．しかし，2 次元空間可視化においてあまりに多くの情報を表現させることは困難である．例として UML に色を加えることにより情報を付加するといった方法 [3] も存在するが，度を越えた情報の付加が複雑化を起こし可視化の質を低下させることは想像に難くない．一方で，3 次元空間可視化はより多くの情報の付加にも耐えうる方法である．3 次元空間において様々なメトリクスと共に動的可視化する手法を確立できれば，開発者はより多くの情報を容易に理解できるようになるだろう．

3. RocatDebugger

我々は，第 2 章で述べたような要件を満たす新たな可視化手法を提案し，RocatDebugger として実装した．本ツールは統合開発環境 Eclipse[*2] のプラグインであり，Eclipse のデバッグ機能を使用しながら動的可視化することが可能である．また，ブレークポイントを使用することによってプログラムが中断された場合には，その状態でのスタックトレースが可視化される．

本手法では，メソッド，クラス，パッケージが 3 次元空間内で立体的なオブジェクトとして表現される．プログラムの実行時には，これらのオブジェクトが動作を起こすことにより，ユーザはプログラムの実行の状態を視覚的に知ることができる．以降，3 次元空間において可視化される

[*1] 奈良先端科学技術大学院大学情報科学研究科ソフトウェア工学研究室で開発されている可視化ツールの名前 ROCAT に由来する．

[*2] https://eclipse.org

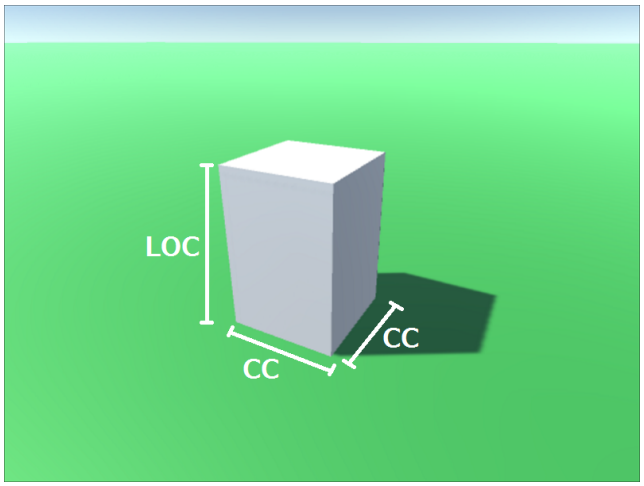

図 1　可視化されたメソッド

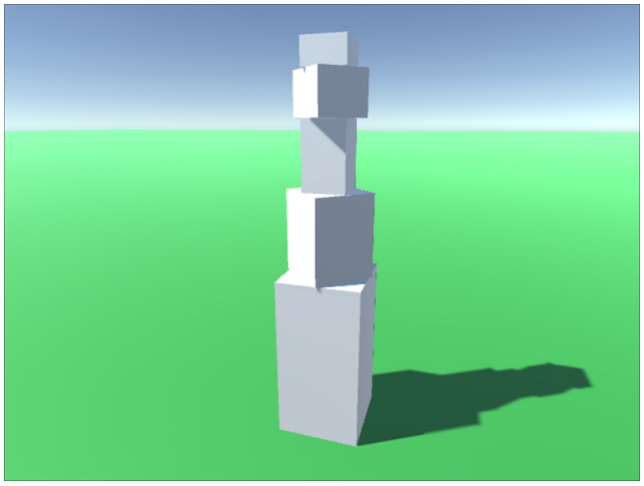

図 2　可視化されたクラス

内容についてそれぞれ詳しく述べていく．

3.1　メソッドの可視化

本手法の可視化空間において，プログラムの構造を表現するための最小単位はメソッドである．コード内の 1 つのメソッドは図 1 で示されるような 1 つの立方体へと置き換えられる．立方体の大きさは 2 つのメトリクスを表しており，高さが LOC，幅が CC に対応している．

3.2　クラスの可視化

コード内の 1 つのクラスは，クラスが内包するメソッドの積み重ねによって，図 2 で示されるような建物として表される．建物の高さはメソッドの LOC の合計に等しくなるので，他の建物と比較することによりクラスの規模を一目で把握することが可能となる．なお，メソッドを積み重ねるたびに 45°の回転がなされているのはメソッドを区別するためであるが，将来的には色やテクスチャを貼り付けることで区別しやすくする予定である．

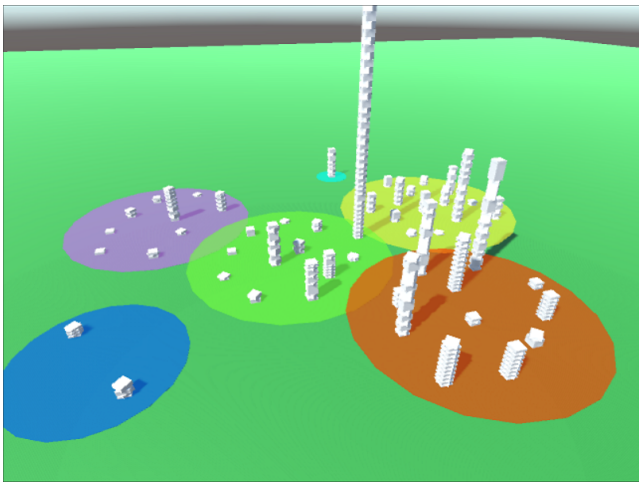

図 3　可視化されたプログラムの全体構造

3.3　パッケージ可視化

同一のパッケージに属するクラスは 1 か所に集められ，円形の板 (パッケージ) に乗せられる．どのパッケージにも属していないクラスが存在する場合は，それらのクラスがデフォルトパッケージという 1 つのパッケージに属しているものとして扱う．図 3 ではある 1 つのプログラムを可視化した際の全景を示しており，複数のパッケージとそれらに属するクラスが表示されている．この図を見た時，ユーザは中央に映る巨大なクラスに興味を抱くか，あるいはコードの臭い [4] を感じ取るかもしれない．

3.4　プログラムの動的可視化

プログラムの実行中に呼び出されたメソッドは，図 4 に示されるように，呼び出したメソッドから呼び出されたメソッドへと流れる粒子によってリアルタイムに可視化される．また，呼び出されたメソッドは変色し，時間の経過によって徐々に元の白色へと戻っていく．ユーザはプログラム全体の中からアクティブなクラスとメソッドを容易に発見することが可能である．

また，Eclipse のデバッグ機能であるブレークポイントやステップ実行を用いることによりプログラムが中断された時には，その状態でのスタックトレースが粒子の流れによって表現される．通常，スタックトレースを確認するためにはコンソールなどに文字列を表示させて 1 つずつメソッド名を追っていく必要があるが，本手法を用いればプログラムの構造を確認しながら一目でスタックトレースを知ることが可能である．

3.5　配置

クラスを無秩序に配置した場合，メソッドの呼び出しを表す粒子の流れが互いに交差したり，あるいは距離が遠すぎたりといった不都合が生じる．こういった問題は可視化の質を低下させる恐れがあるため，クラスの配置は適切に

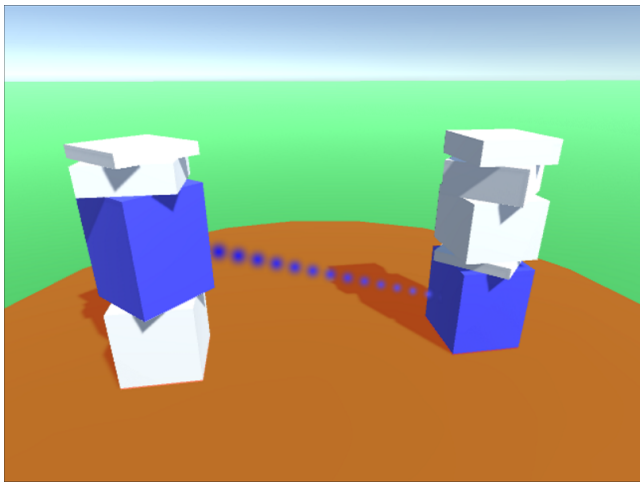

図 4　呼び出されたメソッドの可視化

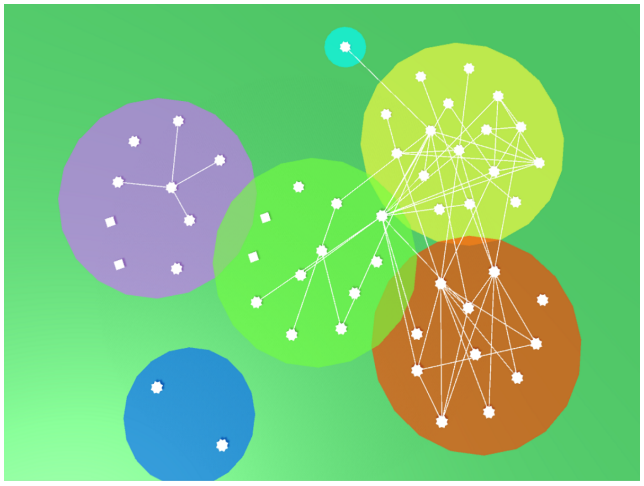

図 5　メソッドの呼び出し関係を利用したクラスの配置

行うべきである．

本手法ではクラスの配置のために力学モデル [5] を使用する．力学モデルは，リンクを持つ複数のノードを適切に配置するためのグラフ描画アルゴリズムであり，ノード間で働く引力と斥力が均衡する位置に各ノードは配置される．本手法でクラスの配置に力学モデルを適用するために，クラスをノード，クラス同士の関係性の強さをリンクと見なす．図 5 は図 3 のプログラムを真上から見下ろした図であり，ここでは配置方法の説明のためにメソッドの呼び出し関係のあるクラス同士に白線を表示している．メソッドの呼び出し関係が存在するクラス同士や同一のパッケージに属しているクラス同士ほど関係性が強く設定され，これらのクラスはクラス間で働く引力によってなるべく近くに配置される．他にも，互いにクラスが至近距離に近づき過ぎないように一定の距離を保つなど，いくつかの処理を加えている．

4. 設計

図 6 に本ツールのアーキテクチャを示す．本ツールは，Eclipse プラグイン，監視モジュール，可視化モジュールの 3 つのモジュールの実装によって構成される．モジュール間でのデータの送受信はソケット通信によって行われる．

Eclipse プラグインは，ユーザが本ツールを使用するためのユーザインタフェースである．ユーザは Java プロジェクトと実行構成を Eclipse 上で選択した後，プラグインによってプログラムの実行が開始される．この際プラグインは，対象のプログラムを監視するモジュールを JVM (Java Virtual Machine) に付与する．

監視モジュールは，対象のプログラムの実行中に呼び出されたメソッドを収集し，可視化モジュールに送信する．プログラム実行中に発生するイベントを監視するために，JVMTI[*3] を使用して実装した．JVMTI は，エージェントと呼ばれるプログラムを実装するためのインタフェースであり，プロファイリングやモニタリングの開発ために用いられる．実装したエージェントを Java プログラムの実行の際に -agentpath オプションを用いて付与することによって，容易に JVM の監視を行うことが可能である．

可視化モジュールは，Java ソースコードから得た情報と監視モジュールから受信した情報を画面に表示する．レンダリングエンジンとして Unity[*4] を使用しているため，高品質なリアルタイムレンダリングを可能としている．

5. 関連研究

5.1 プログラムの動的可視化

現在，2 次元空間においてプログラムの動的可視化を行うための様々なツールが存在している．ツールの多くはプロファイリングを目的としていることが多いが，プログラムの動作を可視化するという点において本論文の手法との関連性は強いため，各ツールについて紹介していく．

FlameGraph [6] は，CPU やメモリのプロファイリングを目的とするツールである．このツールは，一定時間の間隔でスタックトレースをサンプリングすることにより，スタックフレームごとの CPU 占有率を可視化する．横軸でサンプルの数，縦軸でスタックトレースが表される 2 次元空間における可視化手法であり，あたかも炎のような外見となりツールの名前もそれに由来する．リアルタイム性はなく，時間的な順序に関する情報やプログラムの構造を示す情報も含んでいないが，複雑なスタックフレームを 2 次元空間において理解しやすい形で表現することに優れており，またプログラミング言語の壁を越えて幅広い環境で使用できる強力なツールである．

Kieker [7] は，実行中のプログラムを監視して得られた情報をリアルタイムに表示するツールである．棒グラフや線グラフなど表現は単純であるが，時間あたりの各メソッ

[*3] http://docs.oracle.com/javase/jp/7/technotes/guides/jvmti/index.html
[*4] http://japan.unity3d.com

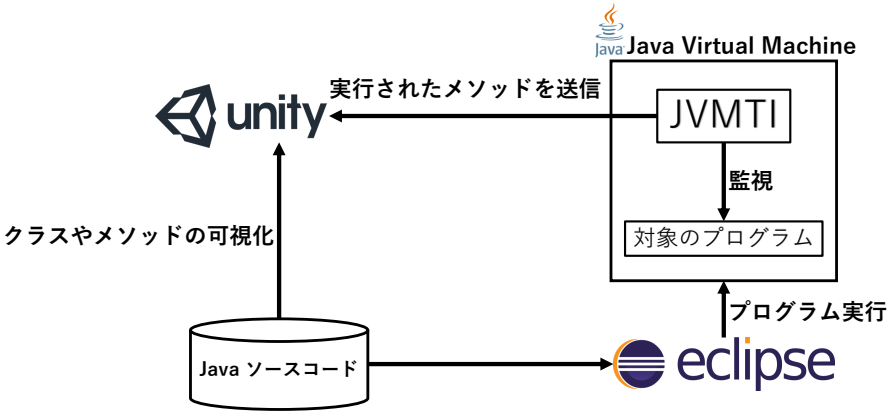

図 6　RocatDebugger アーキテクチャ

ドの呼び出し回数や JVM のヒープ使用率，ガベージコレクションの回数といった多様な情報を確認することができる．また，Web ブラウザから閲覧可能であり利便性は高い．

Jive, Jove [8] は，ある時点で実行されているクラスとパッケージやそれらの実行時間を 2 次元空間において可視化するツールである．本手法と同様，開発者のプログラム理解の補助を目的としており，“プログラムが何を実行しているか” をリアルタイムに可視化できることが特徴である．2 次元空間可視化であるため多くの情報を同時に表示することには適しておらず，目的に応じて可視化のウィンドウを切り替える必要がある．また，プログラムの構造を表現することはしていない．

5.2　コードの街

本手法のようにクラスなどを建物に置き換えてプログラム構造を表現する手法は，その外見からコードの街と比喩される．プログラムを街として表現する方法は CodeCity や EvoSpaces で確立され，以降はプログラムの可視化手法として活発に使用されるようになった．CodeMetropolis [9] は描画のために Minecraft[*5] を使用しており，より街らしく可視化することに特化している．こういった動きには，プログラムを身近な物に例えることによって直感的に理解できる物へと変換するという目論見が含まれていると考えられる．これらのツールより以前には，文献 [10] のようにパッケージとクラス，メソッドを半球を用いて表現する手法が存在している．

コードの街を動的可視化のために用いたツールとして，ExplorViz [11] がある．ExplorViz によって，メソッドの実行回数や実行時間がコードの街の中でリアルタイムに可視化され，Web ブラウザから閲覧することができる．ツールの特徴として，単一あるいは複数のプログラムからなるシステムを対象に可視化できること，過去の時間を入力することにより自由に履歴を遡ることができることが挙げられる．また，Kieker や ExplorViz のようにサーバに収集された情報を Web ブラウザから閲覧するツールは，リモートアクセスによって複数のユーザ間で同一の可視化データを共有できるという利点がある．ただし，情報は一定時間の間隔で記録されるのでリアルタイム性はやや失わる．また，LOC や CC といったメトリクスは可視化に含まれていないので，メソッドやクラスの特徴を可視化から捉えることはできない．

6.　おわりに

本論文ではプログラムを動的可視化する手法と，本手法を実装したツール RocatDebugger について紹介した．本ツールは未だプロトタイプであり，いくつかの機能は実装が完全ではなくユーザインタフェースも洗練されていない．特に，可視化空間においてクラス名やメソッド名をポップアップさせること，より適切にクラスを配置するために力学モデルを調整することは重要である．また，より多くの情報を加えて可視化空間で表現することも計画しており，メソッドの実行時間を色などを用いて可視化すればプロファイリングへの応用も期待できる．

本ツールの実装が終わり次第，我々は本ツールを用いた手法評価を開始する．ユーザが本ツールを使用することにより，プログラム理解の速度や理解度にどの程度の向上が得られるか，アンケートや問題を提示することで計測する予定である．

謝辞　本研究は JSPS 科研費 16H05857 の助成を受けた．

[*5] https://minecraft.net

参考文献

[1] Wettel, R. and Lanza, M.: Visualizing software systems as cities, *2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, IEEE, pp. 92–99 (2007).
[2] Alam, S. and Dugerdil, P.: Evospaces visualization tool: Exploring software architecture in 3d, *14th Working Conference on Reverse Engineering (WCRE 2007)*, IEEE, pp. 269–270 (2007).
[3] Coad, P., Luca, J. d. and Lefebvre, E.: *Java modeling color with UML: Enterprise components and process with Cdrom*, Prentice Hall PTR (1999).
[4] Fowler, M.: Refactoring: Improving the Design of Existing Code, *11th European Conference. Jyväskylä, Finland* (1997).
[5] Fruchterman, T. M. and Reingold, E. M.: Graph drawing by force-directed placement, *Software: Practice and experience*, Vol. 21, No. 11, pp. 1129–1164 (1991).
[6] Gregg, B.: The Flame Graph, *Queue*, Vol. 14, No. 2, pp. 10:91–10:110 (online), DOI: 10.1145/2927299.2927301 (2016).
[7] Van Hoorn, A., Waller, J. and Hasselbring, W.: Kieker: A framework for application performance monitoring and dynamic software analysis, *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, ACM, pp. 247–248 (2012).
[8] Reiss, S. P. and Tarvo, A.: What is my program doing? Program dynamics in programmer' s terms, *International Conference on Runtime Verification*, Springer, pp. 245–259 (2011).
[9] Balogh, G., Szabolics, A. and Beszédes, A.: CodeMetropolis: Eclipse over the city of source code, *Source Code Analysis and Manipulation (SCAM), 2015 IEEE 15th International Working Conference on*, IEEE, pp. 271–276 (2015).
[10] Balzer, M. and Deussen, O.: Hierarchy based 3D visualization of large software structures, *Proceedings of the conference on Visualization'04*, IEEE Computer Society, pp. 598–4 (2004).
[11] Fittkau, F., Roth, S. and Hasselbring, W.: ExplorViz: Visual runtime behavior analysis of enterprise application landscapes, AIS (2015).