Visualizing Data in Software Cities

Susanna Ardigò, Csaba Nagy, Roberto Minelli, Michele Lanza

REVEAL @ Software Institute — USI, Lugano, Switzerland

Abstract—The city metaphor for visualizing software systems in 3D has been widely explored and it has led to many diverse implementations and approaches. However, when looking at software systems in general, and when using specifically a city approach, it is evident that something is missing: The data. Indeed, software systems are intrinsically *driven by data*, which is usually managed using databases or often also simply stored in files coming in a variety of formats, such as CSV, XML, and JSON. While such data files are part of a project's file system and can thus be easily retrieved, the situation is different for databases: A database is usually not contained in the file system, and its presence can only be inferred from the source code which contains the database accesses.

We present an extension of the CodeCity implementation, M3TRICITY2, with two new contributions: First, we consider data files and use simple metrics to integrate them in the visualization seamlessly. Second, we present a novel way to add a database to the visualization by making use of the one remaining space left unused: the sky and the underground. We present our contributions and illustrate them on various software systems.

Index Terms—Software visualization, Database access visualization, Software maintenance, CodeCity

I. INTRODUCTION

The city metaphor has been widely explored to visualize the structure and the evolution of software systems [1]–[9]. When looking at a software city, while we see its structure and implementation, something is missing: The data. But Software systems run on data. It intrinsically drives them, giving them a "reason" to exist. If data is an integral part of software systems, then it should be part of the city visualization.

Classic software visualization approaches consider systems as files located in nested folders, above all because this is the way their source code is stored and versioned. Data on the other hand can originate from various sources. The most straightforward approach is to manage data using data files, which often come in various formats, such as XML and JSON. Such files are usually versioned together with the source code. A more systematic and scalable approach is for a system to make use of databases. Relational databases (*e.g.,* PostgreSQL, MySQL, Oracle) have been widely used for decades, while recently there has been an uptick in the use of NoSQL databases (*e.g.,* Neo4j, MongoDB). Databases are generally not contained in a system's repository, and as a result they have so far remained largely outside the focus of visualization approaches. Some researchers have developed approaches to visualize databases together with software systems [10]–[13], but no state-of-the-art method considers the system and its various data sources as a whole, presented in one view where they evolve together and interact with each other.

We present M3TRICITY2, an extension of M3TRICITY [14], which seamlessly integrates various data sources of a system into the visualization. First, we represent data files in the city and map simple metrics on their meshes (*i.e.,* their shapes rendered in the visualization). Second, we add the database to the visualization using the free space of the sky above or the underground below the city. We infer the schema of an application's database using SQLInspect [15], a static analyzer to inspect database usage in Java applications. We calculate metrics of the database entities, such as the number of columns of tables or the number of classes accessing them. Finally, we present the system with its data sources using the history-resistant layout of M3TRICITY, *i.e.,* the new entities remain at their reserved place throughout the evolution. The resulting view seamlessly integrates data sources to a software city and enables a more comprehensive understanding of a system in terms of its source code and its data.

M3TRICITY2 is a web application available at `https://metricity.si.usi.ch/v2`.

II. RELATED WORK

Since the seminal works of Reiss [1] and Young & Munro [2], many studied 3D approaches to visualize software systems. The software as cities metaphor has been widely explored and led to diverse implementations, such as the Software World approach by Knight *et al.* [3], the visualization of communicating architectures by Panas *et al.* [4], [16], Verso by Langelier *et al.* [5], CodeCity by Wettel *et al.* [6], [17], EVO-STREETS by Steinbrückner & Lewerentz [7], CodeMetropolis by Balogh & Beszedes [8], and VR City by Vincur *et al.* [9].

Only a few approaches considered presenting the databases together with the source code, and interestingly, most use the city metaphor. Meurice and Cleve presented DAHLIA to visualize database schema evolution [10], which uses the city metaphor where buildings in the city represent database tables. DAHLIA 2.0 presented the city of the source code and the database side-by-side [11]. Zirkelbach and Hasselbring presented RACCOON [12], a visualization approach of database behavior, which uses the 3D city metaphor to show the structure of a database based on the concepts of entity-relationship diagrams. Marinescu presented for enterprise systems a meta-model containing object-oriented entities, relational entities and object-relational interactions [13], stating that "*to perform reverse engineering on enterprise software systems we need a specific meta-model which contains [...] entities from the object-oriented part and [...] entities regarding the relational part [...] and the interactions between the two paradigms.*"
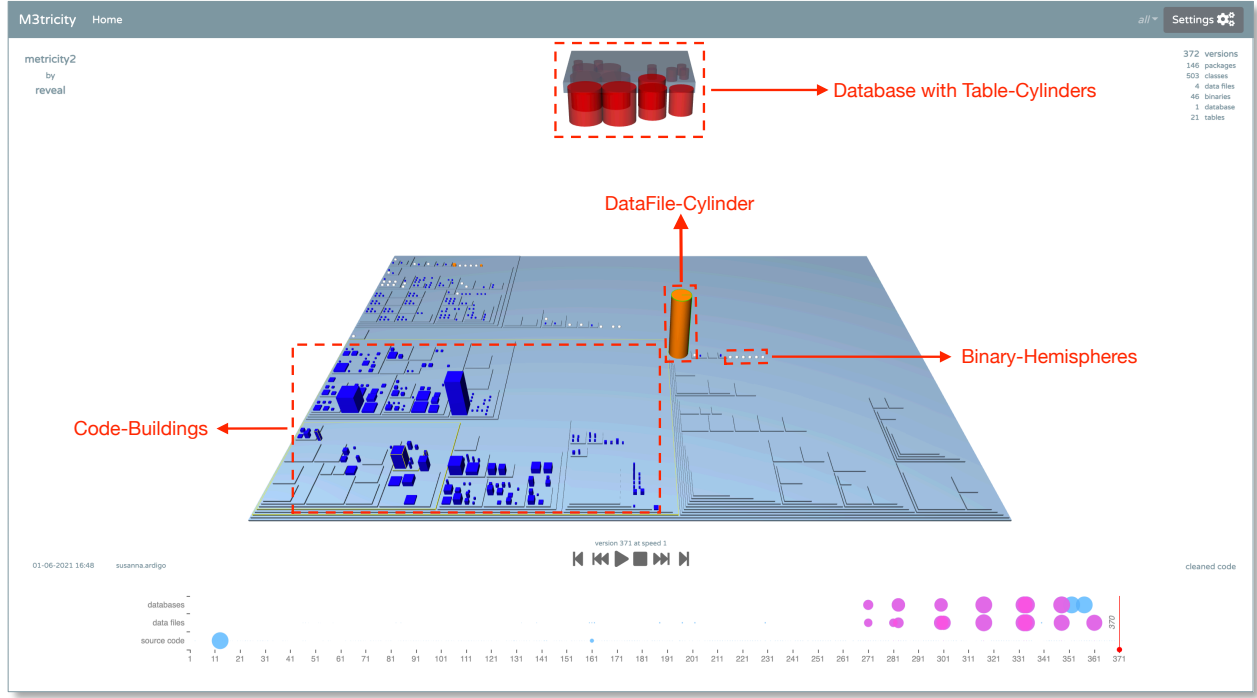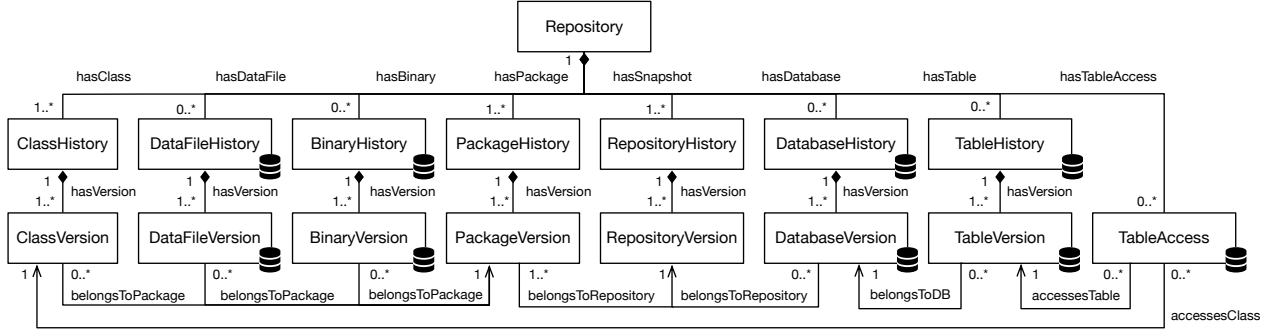
Fig. 1. The Main Page of M3TRICITY2

Fig. 2. Evolution Model of M3TRICITY2

## III. VISUALIZING DATA IN SOFTWARE CITIES

The goal of M3TRICITY2 is to represent a software system and its data sources (*i.e.,* data files and databases), while differentiating the two parts in the visualization. M3TRICITY2 builds on M3TRICITY [14], to which we refer for details on the history-resistant layout.

Figure 1 shows a screenshot of M3TRICITY2 visualized in M3TRICITY2. The software city visualization is in the center with the database as a cloud above the city. Information panels present the name of the repository (top left), the metrics of the system (top right), the actual commit (bottom left), and its commit message (bottom right). The timeline at the bottom depicts the evolution of the project where one can spot significant changes in the metrics. The evolution can be controlled with the buttons below the city.

We illustrate the model of M3TRICITY2 and discuss how to analyze and visualize the data in software cities.

A. Evolution Model with Data Entities

M3TRICITY2 extends the evolution model of M3TRICITY with entities representing data sources. The underlying model is depicted in Figure 2. The purpose of M3TRICITY was to keep the visualization resistant to changes throughout the evolution. Accordingly, the model keeps track of the *histories* of each *version* of the entities.

In M3TRICITY a *Repository* is composed of *Packages* and *Classes*. M3TRICITY2 adds *Databases* and *Tables* to the model. Additionally, *TableAccess* association classes represent when *Classes* access *Tables* to query or modify data in the database. We also add an entity for *DataFiles*, *i.e.,* files in the repository to store data (*e.g.,* XML or JSON files). Finally, we add a *Binary* entity to distinguish other non-source files in the repository (*e.g.,* images). In Figure 2, a database icon 🛢 highlights the new parts of the model.

B. Analysis of Data in Software Cities

The top part of Figure 3 summarizes the process in place when the user asks for the analysis of a system.
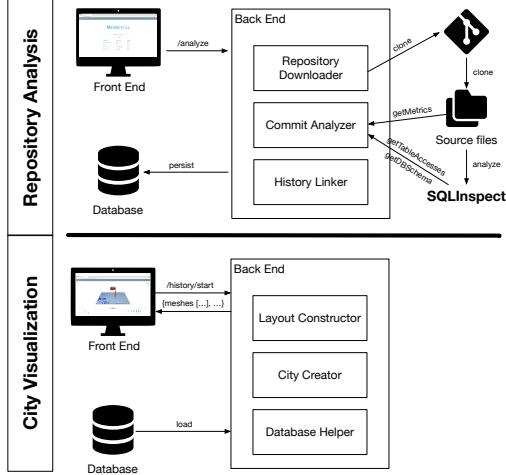

Fig. 3. Repository Analysis and City Visualization processes

The analysis starts after the user has provided the URL of a Git repository. The key components taking part in the analysis are the *Repository Downloader*, *Commit Analyzer*, and *History Linker*. The Repository Downloader clones the project and the Commit Analyzer starts analyzing each commit.

M3TRICITY collected metrics of the source files. M3TRICITY2 introduces new Commit Analyzer steps to identify the data files and extract their metrics. It runs SQLINSPECT, a static analyzer that inspects database usage in Java applications [15]. SQLINSPECT extracts the SQL queries embedded in the classes and returns an inferred schema of the application with a list of table accesses. It supports relational databases (*i.e.,* MySQL, SQLite, Apache Impala) used in JDBC applications, Android apps, and applications using Spring or Hibernate frameworks. SQLINSPECT was also extended to infer the database schema of applications using MongoDB. The Commit Analyzer processes the output files of SQLINSPECT and calculates related metrics to update the actual model of the system.

The metrics collected for data files are the following: *Number of Entities* represents the total number of entities (*e.g.,* XML elements or JSON objects) in the file; *Number of Entity Types*; *Maximum Number of Properties per Entity* describes the number of properties of the largest entity type; *Maximum Nesting Level* describes the maximum depth of the sub-entities. We collect the *Number of Columns* and the *Number of Table Accesses* for database tables.

The History Linker traverses the entities and creates History entities to represent multiple versions of the same entity over the evolution. For example, when a DataFile (*e.g.,* an XML) is moved or renamed, a single DataFileHistory will represent the same file at different locations in multiple snapshots. In the end, the constructed model is persisted in a database.

C. Visualization of Data in Software Cities

The bottom of Figure 3 depicts the visualization process. The Front End sends the request to the Back End, which returns the meshes with their locations for the current snapshot. The main components of M3TRICITY2 in this process are the *Layout Processor* and the *City Creator* in the Back End, and *Canvas Creator* and *Mesh Renderer* in the Front End.

M3TRICITY2 defines new meshes for the new entities (see Figure 4). An orange *datafile-cylinder* represents a data file. A binary file has a gray *binary-hemisphere* mesh, and a database table is drawn as a red *table-cylinder*.
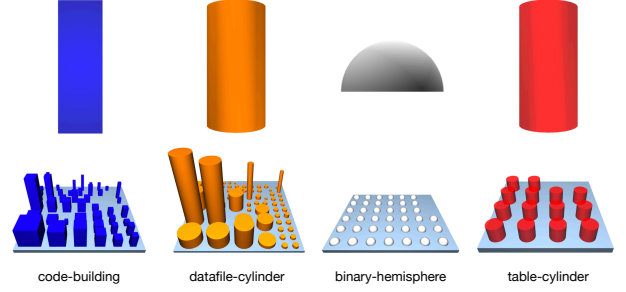

Fig. 4. The New Meshes of M3TRICITY2

The new meshes are placed in the city according to the algorithm described in Pfahler *et al.* [14]. The layout leverages the evolution model to create a history-resistant layout. A recursive bin-packing algorithm places the individual elements in the city, considering the entire history of the system. It reserves areas for all elements and ensures that the same entity remains at the same place throughout the evolution.

An interesting problem is the placement of the database tables. Conceptually they are different than regular source files: They are not part of the source code, per se. Database tables are not version-controlled together with source files, either. They are inferred, thus, they should not be likened to source code. We separate the database tables by making use of the one remaining space left unused: the Sky and the Underground.
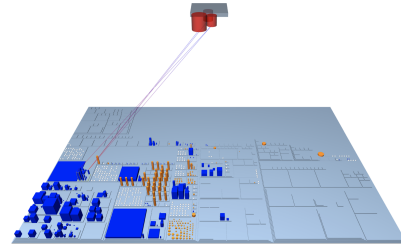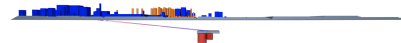

Fig. 5. City with Clouds
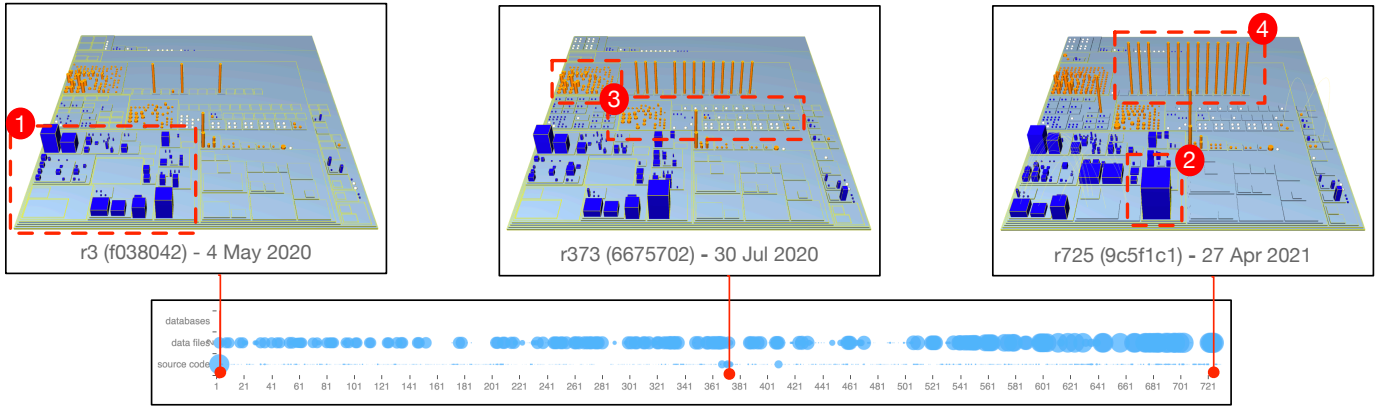

Fig. 6. City with Underground

Fig. 7. Visualizing the SwissCovid Android App with M3TRICITY2

In the *City with Clouds* visualization the file system is positioned at the ground level, and the database is positioned above the city so that the two are facing each other (see Figure 5). In the *City with Underground*, the city is lifted, giving space to the databases and tables represented below (see Figure 6). In both layouts, edges connect the accessed tables with the classes. To emphasize the inferred nature of the tables, the database is rendered with light transparency, giving it a ghostly appearance above/below the city.

IV. CASE STUDIES

A. SwissCovid Android App

Figure 7 presents the evolution of the official SwissCovid Android App.[1] Its Java back end does not use a local database but has many resource files. It has around 8k and 4k lines of code written in Java and Kotlin.

The repository was initialized with a commit of license and readme files on April 15, 2020.[2] In the third revision, on May 4, 2020, they committed the `ch.admin.bag.dp3t` package.[3] The structure remains stable after this commit, but the project is developed with regular changes, as seen in the commits of July 30[4] and April 27, 2021.[5]

Its growth is easily observable during the evolution. The timeline in Figure 7 shows regular contributions to data files. Indeed, the XML files grow from an initial 10k to 25k lines.

Interesting districts can be spotted in the new layout with the data files. The Java classes are primarily located on the bottom-left side of the city ①. One robust class ② stands out: `SecureStorage.java` – the encrypted implementation of the `android.content.SharedPreferences`.[6] This is the primary storage implementation with a critical role in the contact tracing app of Switzerland.

Above the district of the source files, we can see the neighborhoods of resource files ③. There are several tiny PNG and SVG files and folders with smaller layout XMLs. An interesting district is a folder with `strings.xml` files of various languages ④. The initial version supported only three official Swiss languages (Italian, French and German). As the app evolves, the XMLs grow, and the number of supported languages increases to twelve.

B. GnuCash Android App

GnuCash Android[7] is a companion app of the open-source GnuCash accounting software. It has an SQLite database with a central role in the project to store the transactions. Its codebase is in Java with around 27k LOC. In addition, it has XML files with about 22k lines.

Figure 8 depicts the evolution of GnuCash Android. The project started with an "*Initial commit*" by "codinguser" on May 13, 2012.[8] The commit already contained source files ①, but no database yet, which was added two commits later, in the commit "*Created database store for accounts and transactions*" on May 24, 2012.[9] The visualization confirms the appearance of two tables: `Transactions` and `Accounts`.

On April 11, 2015, a refactoring took place in the project and they also refactored the database schema (removed deprecated tables and code).[10] This revision has four tables in the database, and they are accessed only from a few classes ②. The biggest table, `Accounts`, has 14 columns and is modified 31 times throughout the evolution. It was initially added to the project with only four columns. The `DatabaseHelper` class handles the communication with the database ③. The other classes accessing the database are adapter classes: `AccountsDbAdapter` and `TransactionsDbAdapter`. M3TRICITY2 shows 11 tables throughout the evolution of the project, with some tables introduced only temporarily.

---

[1]See **https://github.com/SwissCovid/swisscovid-app-android**

[2]See **https://github.com/SwissCovid/swisscovid-app-android/commit/98363d9**

[3]See **https://github.com/SwissCovid/swisscovid-app-android/commit/f038042**

[4]See **https://github.com/SwissCovid/swisscovid-app-android/commit/6675702**

[5]See **https://github.com/SwissCovid/swisscovid-app-android/commit/9c5f1c1**

[6]See **https://developer.android.com/reference/android/content/SharedPreferences**

[7]See **https://github.com/codinguser/gnucash-android**

[8]See **https://github.com/codinguser/gnucash-android/commit/94a9c01**

[9]See **https://github.com/codinguser/gnucash-android/commit/34ce20d**

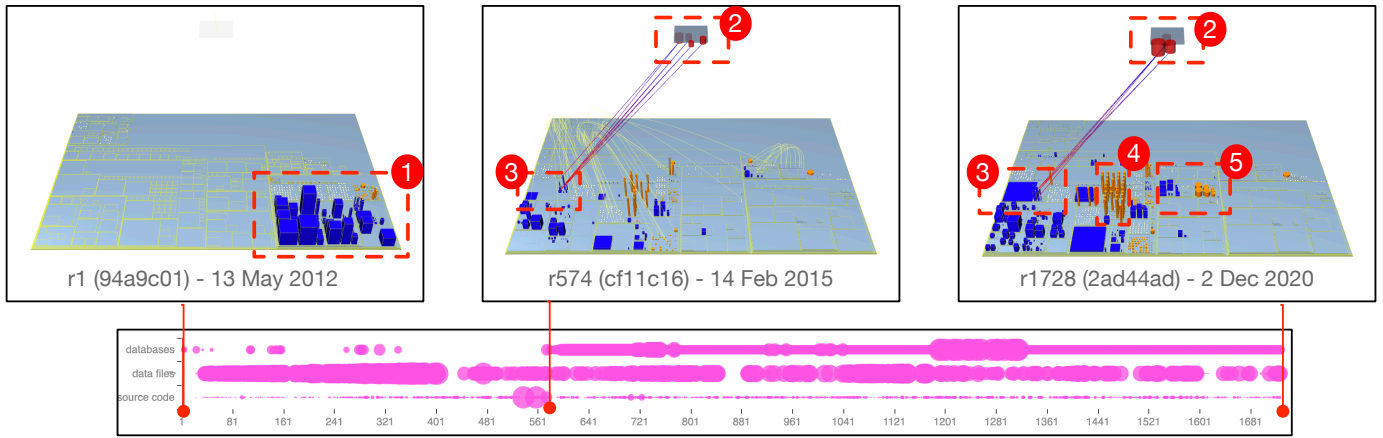[10]See **https://github.com/codinguser/gnucash-android/commit/0fa9038**

Fig. 8. Visualizing GnuCash Android App with M3TRICITY2

The last commit of the project is from December 2, 2020.[11] Districts of resources similar to the SwissCovid App can be spotted easily. The larger XML files are again the `strings.xml` files of the 35 languages of GnuCash Android ④. It is also interesting to observe some smaller and wider XML files in one district ⑤. These are `Import*.xml` files in the `test` package: They store test data.

V. CONCLUSIONS & FUTURE WORK

M3TRICITY2 extends the city metaphor with a novel way to add "data" to the visualization, making it an integral part of the visualization. We implemented two views to separate the database as an inferred element: *City with Clouds* and *City with Underground*, and connected the data tables to the classes accessing them. We added data files and used descriptive shapes and colors to differentiate them from the buildings of source classes. We demonstrated the approach on two systems, one with small data files and one with a relational database.

There are limitations to the approach: It requires the database schema, which is rarely accessible with the source code. In case it is missing, schema inference can help. Moreover, schema-less NoSQL databases that allow unstructured data might not be appropriately represented with our model, which can affect the already non-trivial problem of linking different versions of entities (*e.g.,* detecting the renaming of a document in a document store). Finally, as we demonstrated the approach on a system with only one database, multiple databases pose additional challenges. While conceptually the model can handle them, questions arise about their positioning. Seeking answers to these questions is part of our future work.

ACKNOWLEDGMENTS

[11]See **https://github.com/codinguser/gnucash-android/commit/2ad44ad**

REFERENCES

[1] S. P. Reiss, "An engine for the 3D visualization of program information," *J. Visual Languages & Computing*, vol. 6, no. 3, pp. 299–323, 1995.

[2] P. Young and M. Munro, "Visualising software in virtual reality," *Proc. 6th Int. Workshop on Program Comprehension*, pp. 19–26, 1998.

[3] C. Knight and M. Munro, "Virtual but visible software," in *Proc. 17th Int. Conf. Information Visualization*. IEEE, 2000, pp. 198–205.

[4] T. Panas, T. Epperly, D. Quinlan, A. Saebjornsen, and R. Vuduc, "Communicating software architecture using a unified single-view visualization," in *Proc. 12th Int. Conf. Engineering Complex Computer Systems*. IEEE, 2007, pp. 217–228.

[5] G. Langelier, H. Sahraoui, and P. Poulin, "Visualization-based analysis of quality for large-scale software systems," in *Proc. 20th Int. Conf. Automated Software Engineering*. ACM, 2005, pp. 214–223.

[6] R. Wettel and M. Lanza, "Visualizing software systems as cities," in *Proc. 4th Int. Workshop on Visualizing Software for Understanding and Analysis*. IEEE, 2007, pp. 92–99.

[7] F. Steinbrückner and C. Lewerentz, "Representing development history in software cities," in *Proc. 5th Int. Symposium on Software Visualization*. ACM, 2010, pp. 193–202.

[8] G. Balogh and A. Beszedes, "CodeMetropolis - code visualisation in MineCraft," in *Proc. 13th Int. Working Conf. Source Code Analysis and Manipulation*. IEEE, 2013, pp. 136–141.

[9] J. Vincur, P. Navrat, and I. Polasek, "VR City: Software analysis in virtual reality environment," in *Proc. Int. Conf. Software Quality, Reliability and Security Companion*. IEEE, July 2017, pp. 509–516.

[10] L. Meurice and A. Cleve, "DAHLIA: a visual analyzer of database schema evolution," in *Proc. 2014 Software Evolution Week - IEEE Conf. Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, 2014, pp. 464–468.

[11] ——, "DAHLIA 2.0: A visual analyzer of database usage in dynamic and heterogeneous systems," in *Proc. 2016 Working Conf. Software Visualization (VISSOFT)*. IEEE, 2016, pp. 76–80.

[12] C. Zirkelbach and W. Hasselbring, "Live visualization of database behavior for large software landscapes: The RACCOON approach," Department of Computer Science, Kiel University, Tech. Rep., 2019.

[13] C. Marinescu, "Applications of automated model's extraction in enterprise systems," in *Proc. 14th Int. Conf. Software Technologies (ICSOFT 2019)*. SCITEPRESS, 2019, pp. 254–261.

[14] F. Pfahler, R. Minelli, C. Nagy, and M. Lanza, "Visualizing evolving software cities," in *Proc. 2020 Working Conf. Software Visualization (VISSOFT)*. IEEE, 2020, pp. 22–26.

[15] C. Nagy and A. Cleve, "SQLInspect: A static analyzer to inspect database usage in java applications," in *Proc. IEEE/ACM 40th Int. Conf. Software Engineering: Companion (ICSE 2018)*, 2018, pp. 93–96.

[16] T. Panas, R. Berrigan, and J. Grundy, "A 3D metaphor for software production visualization," in *Proc. 7th Int. Conf. Information Visualization*. IEEE, 2003, pp. 314 – 319.

[17] R. Wettel and M. Lanza, "CodeCity: 3D visualization of large-scale software," in *Comp. 30th Int. Conf. Software Engineering (ICSE 2008)*. ACM, 2008, p. 921–922.