ABSTRACT

The city metaphor for visualizing software systems in 3D has been widely explored and has led to many diverse implementations and approaches. Common among all approaches is a focus on the software artifacts, while the aspects pertaining to the data and information (stored both in databases and files) used by a system are seldom taken into account.

We present M3TRICITY, an interactive web application whose goal is to visualize object-oriented software systems, their evolution, and the way they access and use data. We illustrate how it can be used for program comprehension and evolution analysis of data-intensive software systems.

Demo video URL: https://youtu.be/uBMvZFIIWtK

CCS CONCEPTS
- Software and its engineering:

KEYWORDS
- Software and data visualization, Program comprehension

ACM Reference Format:

1 INTRODUCTION

Program comprehension is a fundamental activity for software maintenance and evolution. Developers spend considerably more time reading and understanding existing code rather than writing new code [12]. Software visualization is a popular technique to perform program comprehension [19]. Many techniques have been proposed, ranging from simple 2D displays, such as polymetric views [8] and UML diagrams to more complex 3D techniques, even extending into the realm of virtual (VR) and augmented reality (AR) [3, 10, 13].

We present M3TRICITY a web application that visualizes software systems in 3D, focusing on the evolution of systems and how they use and access data [1, 17]. M3TRICITY leverages the metaphor of a city [4, 6, 15, 16, 20, 21] in the vein of CODECITY [24] and runs on any modern web browser.

2 M3TRICITY IN A NUTSHELL

In a nutshell, M3TRICITY is a web-based evolution of CODECITY, which popularized the city-based visualization of software systems through the city metaphor [23] (See Figure 1).

Figure 1: CodeCity and the City Metaphor

2.1 The User Interface of M3triCity

Figure 3 shows the main user interface of M3TRICITY. At the center we see the software city A, with folders and files represented as buildings and nested districts. Above the city the sky is used to visualize the databases (inferred) and their tables B, where connecting lines represent the accesses performed from the source code.
Various information about the system being visualized is also displayed. As M3TRICITY is geared towards evolution comprehension, additional panels provide information about the currently visualized commit. To facilitate moving through time M3TRICITY provides a control panel to easily moving forward/backward between different commits as well as fast forward/backwarding and pausing. A timeline at the bottom provides a global overview of the system evolution with additional details pertaining to the commits (i.e., along the timeline), as well as instantaneous access to a specific place in the commit history. The whole city can be rotated, the user can also change its point of view and zoom in and out. Structural changes (i.e., moving of entities) are depicted using yellow curved arcs (see top left annotation with black dots). When the user clicks on an artifact, it is highlighted both in the main visualization as well as along the timeline, denoting all commits in which the entity was involved. More customizations are also available in the settings panel.

### 2.2 Modeling Evolving Data-Intensive Systems

Figure 4 depicts the meta-model of M3TRICITY. Evolving software artifacts are modeled using “histories” in the vein of Girba’s evolution meta-model [5]. For each artifact history, we model each version including binary files and data files (e.g., JSON, XML). Databases are inferred through SQLINSPECT [14]. For each entity, M3TRICITY computes various metrics, summarized in Table 1.

<table>
<thead>
<tr>
<th>Entity</th>
<th>Metric Name</th>
<th>Entity</th>
<th>Metric Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td># Instance Variables</td>
<td>Data File</td>
<td># Entities</td>
</tr>
<tr>
<td></td>
<td># For Loops</td>
<td></td>
<td># Entity Types</td>
</tr>
<tr>
<td></td>
<td># Methods</td>
<td></td>
<td>Max # Properties per Entity</td>
</tr>
<tr>
<td></td>
<td># Lines of Code</td>
<td></td>
<td>Max Nesting Level</td>
</tr>
<tr>
<td>Table</td>
<td># Columns</td>
<td>Binary</td>
<td>Size</td>
</tr>
<tr>
<td></td>
<td># Table Accesses</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: The Metrics Supported by M3triCity
2.3 Architecture
Figure 5 shows the architecture of M3TRICITY.

![Architecture Diagram]

The frontend of M3TRICITY is implemented in TypeScript. It uses Vue.js for the user interface. The 3D visualization is created using Babylon.js. The backend is a Spring Boot application implemented using Java and Gradle.

To start the analysis, the user indicates the URL of a repository and, as an optional parameter, the database type. The execution starts in the module Repository Downloader which contacts git to clone the repository. The Commit Analyzer module iterates through all files of each snapshot of the given repository in chronological order, classifies them, analyzes them and then extracts the metrics. The project uses SQLInspect to reverse engineer the schema of the database and the interactions with the source code. The histories of the entities are created by linking the versions.

2.4 Usage
The user starts the visualization of a city by selecting a processed repository. The repository-related information is loaded from the database with the Database Helper component. The computation of the city layout is handled by the Layout Constructor component which iterates through all the entities of the city. M3TRICITY considers evolution as a first-class citizen, implementing a history-resistant layout, which allocates a dedicated position to each artifact throughout the lifetime. The City Creator module creates the meshes information that are then sent to the frontend which renders them as 3D meshes.

3 SOFTWARE CITY TALES
We illustrate how M3TRICITY can be used to comprehend the evolution of a system, by using the GNUCASH-ANDROID Android companion app of the GNUCASH accounting program as an example. GNUCASH-ANDROID allows recording transactions on-the-go to import the data into GNUCASH later. The main branch of the project is composed of 1,730 commits by 46 contributors. Figure 6 shows six M3TRICITY snapshots in the overall evolution.

![Evolution Diagram]

May 13, 2012: GNUCASH-ANDROID is born (Figure 6a). On May 13, 2012, Ngewi Fet creates the project repository with 83 Java classes, 85 data files, and 243 binaries. The source code is mainly located in the com.android.package package, nicely divided into sub-packages. The res folder contains three directories of images and several directories of data files.

Nov 4, 2012: The database is created (Figure 6b). The developers added a database which is being used by a part of the system that has been added in the meantime. Test classes are growing as well. Data files have been added, mostly related to text constants that need to be displayed.

Jan 31, 2013: GNUCASH-ANDROID undergoes a major restructuring (Figure 6c). Almost 450 files are deleted, and 220 are moved with the renaming of the folder GnucashMobile to app. The database-related classes are still present but the accesses to the tables are removed.

Jan 31, 2013: GNUCASH-ANDROID undergoes a major restructuring (Figure 6c). Almost 450 files are deleted, and 220 are moved with the renaming of the folder GnucashMobile to app. The database-related classes are still present but the accesses to the tables are removed.

Jan 31, 2013: GNUCASH-ANDROID undergoes a major restructuring (Figure 6c). Almost 450 files are deleted, and 220 are moved with the renaming of the folder GnucashMobile to app. The database-related classes are still present but the accesses to the tables are removed.

Jan 31, 2013: GNUCASH-ANDROID undergoes a major restructuring (Figure 6c). Almost 450 files are deleted, and 220 are moved with the renaming of the folder GnucashMobile to app. The database-related classes are still present but the accesses to the tables are removed.

Jan 31, 2013: GNUCASH-ANDROID undergoes a major restructuring (Figure 6c). Almost 450 files are deleted, and 220 are moved with the renaming of the folder GnucashMobile to app. The database-related classes are still present but the accesses to the tables are removed.

Jan 31, 2013: GNUCASH-ANDROID undergoes a major restructuring (Figure 6c). Almost 450 files are deleted, and 220 are moved with the renaming of the folder GnucashMobile to app. The database-related classes are still present but the accesses to the tables are removed.

Jan 31, 2013: GNUCASH-ANDROID undergoes a major restructuring (Figure 6c). Almost 450 files are deleted, and 220 are moved with the renaming of the folder GnucashMobile to app. The database-related classes are still present but the accesses to the tables are removed.

Jan 31, 2013: GNUCASH-ANDROID undergoes a major restructuring (Figure 6c). Almost 450 files are deleted, and 220 are moved with the renaming of the folder GnucashMobile to app. The database-related classes are still present but the accesses to the tables are removed.

Jan 31, 2013: GNUCASH-ANDROID undergoes a major restructuring (Figure 6c). Almost 450 files are deleted, and 220 are moved with the renaming of the folder GnucashMobile to app. The database-related classes are still present but the accesses to the tables are removed.

Jan 31, 2013: GNUCASH-ANDROID undergoes a major restructuring (Figure 6c). Almost 450 files are deleted, and 220 are moved with the renaming of the folder GnucashMobile to app. The database-related classes are still present but the accesses to the tables are removed.

Jan 31, 2013: GNUCASH-ANDROID undergoes a major restructuring (Figure 6c). Almost 450 files are deleted, and 220 are moved with the renaming of the folder GnucashMobile to app. The database-related classes are still present but the accesses to the tables are removed.
Sep 18 2015: Controlled evolution (Figure 6d). Fast forwarding 2 years, the system keeps evolving: the test suite is expanded, the developers started working on a user interface module. The database co-evolves with the system, with the addition and quick deletion of tables.

Dec 28 2015: Extending the tests (Figure 6e). The system evolves mostly with new tests.

Dec 2 2020: Fast forward (Figure 6f). Five years later the system has grown considerably, with some classes reaching considerable size in terms of variables and methods. Data file directories have been added, complementing systematic database accesses on a well-organized DB featuring the three major tables transactions, splits, and scheduled_actions.

4 RELATED WORK
Since the seminal works of Reiss [18] and Young & Munro [25], many approaches to visualize software systems in 3D have been explored. The cities metaphor has been widely used and led to diverse implementations, such as the SOFTWARE WORLD by Knight et al. [6], the visualization of communicating architectures by Panas et al. [15], VERSO by Langelier et al. [7], CODECITY by Wettel et al. [23, 24], EVO-STREETS by Steinbrückner & Lewerenz [20], CODEMETROPOLIS by Balogh & Beszedes [2], and VR CITY by Vincur et al. [22].

Only a few approaches considered presenting data(bases) together with the source code, mostly using the city metaphor. Meurice and Cleve presented DAHLIA to visualize database schema evolution [11], which uses the city metaphor where buildings in the city represent database tables. Zirkelbach and Hasselbring presented RACCOON [26], which uses the 3D city metaphor to show the structure of a database based on entity-relationship diagrams. Marinescu presented a metamodel containing object-oriented entities, relational entities and object-relational interactions [9]. M3TRICITY does not separate source code and data(bases), like existing approaches, but it shows them with their interactions in the same city.

5 CONCLUSION
M3TRICITY extends the original city metaphor by considering an important aspect that has been ignored up to now: the data. Our tool visualizes object-oriented software systems, their evolution, and the way they access data and information. M3TRICITY expands the original city metaphor by adding a number of features and concepts: using different glyphs to distinguish between the various file types, taking software evolution into account to layout the city, inferring and visualizing the databases used by a system, and providing higher accessibility by being publicly available as a web application.

To demonstrate the usefulness of our approach, we illustrate how M3TRICITY can be used to comprehend the evolution of a data-intensive system: GNU/CASH-ANDROID.

ACKNOWLEDGMENTS
We acknowledge the financial support of the Swiss National Science Foundation and the Fonds de la Recherche Scientifique for the project “INSTINCT” (SNF Project No. 190113).

REFERENCES