Visualizing Software Ecosystems as Living Cities

Clinton Jeffery

University of Idaho, Moscow ID 83844, USA
jeffery@uidaho.edu
http://www2.cs.uidaho.edu/~jeffery/personal.html

Abstract. Several groups visualize software systems using "city" metaphors, mapping software onto features such as buildings and roads. This abstract introduces a "living city" metaphor, where programs are visualized as a city populated by users, data structures, threads of execution, and bugs. A living city is a 3D, multi-user virtual world in which the visible artifacts are software and data.

Keywords: software visualization, virtual environments.

1 Introduction

Software visualization includes static views such as UML Class Diagrams, as well as animated program execution behavior.

1.1 Visualizing Software as Cities

Wettel and Lanza [1] developed CodeCity to visualize software as a city. Classes are buildings, whose height indicates the number of methods. Width and length depict the number of attributes. CodeCity depicts large software systems, such as this 8,000 class program.

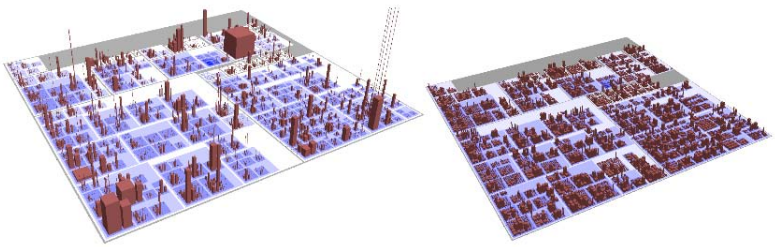

Fig. 1. CodeCity buildings are classes. Height gives the number of methods.

In CodeCity, topography depicts package structure. Layout groups classes in the same package together, and then uses a modified treemap algorithm [2]. CodeCity's static views show how software systems evolve over time. Their metaphor provides the backdrop for the visualization of dynamic program behavior proposed in this paper.

C. Stephanidis (Ed.): Posters, Part I, HCII 2013, CCIS 373, pp. 640–644, 2013.

1.2 Related Work

Knight and Munro [5] developed a tool in which a program is a virtual world, a directory is a country, each file is a city, each class is a district, and each method is a building. They do not map the building interiors, but exterior characteristics show size, parameters, and so on. Kuhn et al [3] developed topological software maps, with island chains depicting relationships of software components.

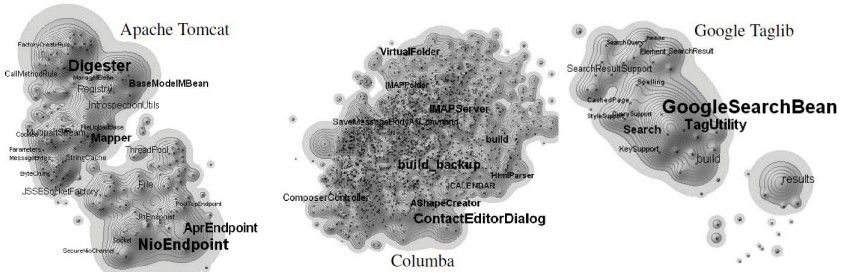

Fig. 2. Topological software maps constructed from component relationships

In CrocoCosmos, [4] a primary model depicts the software system; secondary and tertiary models depict details. Subsystems are streets, with contained classes as buildings. Age is depicted by centrality and elevation.

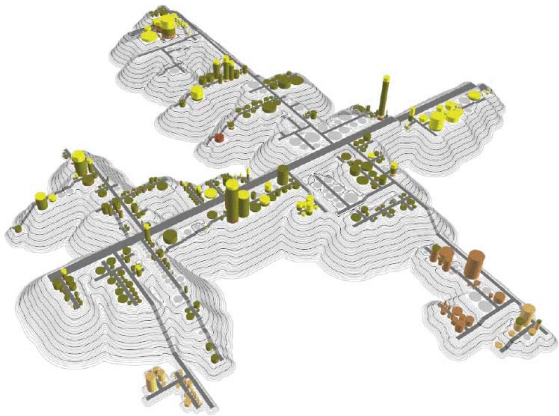

Fig. 3. CrocoCosmos

2 The Living City Metaphor

The "city" metaphor enables sharing, interacting, and debugging collective software development efforts. Extensions to the static software city metaphor are required. The primary extensions introduce dynamic entities.

2.1 Visualizing a Software Ecosystem

Besides peer assistance and review, working in a software city on a set of related programs injects interest and fun, and reduces the cost of collaboration.

2.2 Static Extensions to the City Metaphor

Additional metaphors extend the static backdrop developed by Wettel and Lanza.

directories and packages are roads sizes equivalent to street, arterial, and freeway are needed.

classes' buildings' dimensions height gives the number of methods (storeys). The width is the number of variables. The length is log(| longest method in C|).

Building exterior appearance A texture shows class age and a blended color suggests last commit time. Some old programs are maintained, while others are in ruins.

Class internals Inside a class building is an informative layout for users to walk around in. Starting from ground floor constructors and a directory, one accesses methods via elevator.

Method body details building interiors contain dynamic information: number of activation records live, threads executing there, etc.

Time model Unlike wall-clock time, CPU time may be frozen, or go backwards, a la the Dagger of Time.

Processes and threads Tasks appear as "weeping angels": frozen when seen; moving when unseen.

Functions Functions are singletons containing one public method. A function library resembles a village.

Representing heap Applications are humanoids. Libraries are robots.

Garbage Unused memory lies on the ground.

Atoms data appears as books (string), hammers (int) and saws (real).

External entities Networks are airports, databases are sea ports, local files are mines. Handles are runtime entities such as aircraft or ships.

Associations Class references add connectivity, beyond the street system.

Inheritance Subclasses have a physical resemblance, such as copied buildings with extra floors.

Aggregation Part-of relationships are physical adjacency or containment.

The call stack A beam-of-light model points backwards from callee to caller.

Bugs and warnings A bug report is a spawn-point that emits monsters that attack executions. Killing the bug in-game is temporary; until the bug is fixed, the spawn-point remains.

Building layout After placing the largest buildings first, and placing others around them, and placing associated or coupled classes together, Loretan [6] suggests associating classes by their lexical vocabulary.

Ghosts The city contains remembrances of fixed bugs and deleted code

2.3 Dynamic Extensions

A software city is changed by the software developers that write the code.

Users End users are depicted by computer controlled avatars (NPC's). Users work on use cases, suspend, log out, or crash due to bug attacks.

Developers Developers are NPC's when not present.

Processes and threads The software code forms buildings, but executions are NPC's that move around in a building. A thread standing still is blocked or thrashing and needs help.

3 Case Study: Unicon

This section explores living cities by proposing one for the Unicon programming language, called Unicon City. The case study considers the code that is bundled with the language distribution. The Unicon source code includes 195K lines of C code in the language implementation.

About 350K lines of Unicon code in the distribution was written by 60+ authors. Unicon City has a horizontal axis street (`unicon`), sized so that a person can walk across. The length of the axis street is proportional to the number of lines of code. A street 1cm/LOC long would be 5.7km. The width of this primary arterial is scaled ln(LOC) meters (13.2m).

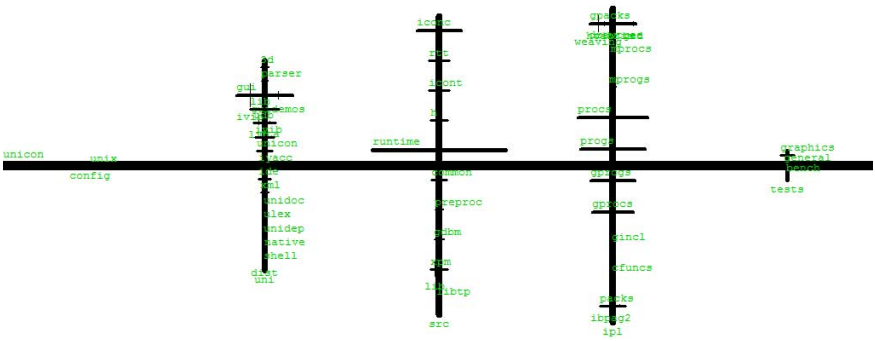

Fig. 4. First street layout for Unicon City

This first street layout for Unicon City is based on the Unicon directory hierarchy. The "CodeCity" treemap algorithm has more crossroads, laying out disconnected parts of the tree adjacently. The "CrocoCosmos" street-based layout is closer to Unicon City. An automatic city layout algorithm might generate a software city atop a map of a real city to increase the aesthetics of the result.

4 Implementation

These tools under construction are needed to complete the living city:

collaborative virtual environment CVE (cve.org) is one.
world generation from a software codebase generate street layouts and buildings from code.
incremental algorithms for code updates update data models from repository commits
high performance dynamic data Unicon reports ˜120 types of events such as control flow, calls, returns, data structures, and garbage collection. Dynamic program behavior events will be used to animate the software city.
NPC AI NPC's need "intelligent" behavior.
early adopter user base Some of the Unicon language community might use this project, but others will prefer or require more privacy than it affords.

5 Conclusions and Future Work

The Living Cities metaphor is a vision of the future of software development: an IDE, visualization tool, and MMO. Although this vision requires substantial time and effort to achieve, the enabling technologies are all in place.

References

1. Wettel, R., Lanza, M.: Visualizing Software Systems as Cities. In: Proceedings of VISSOFT 2007 (4th IEEE International Workshop on Visualizing Software For Understanding and Analysis), pp. 92–99. IEEE Computer Society Press (2007)
2. Shneiderman, B.: Tree visualization with Treemaps: a 2-D Space-filling Approach. ACM Transactions on Graphics 11(1), 92–99 (1992)
3. Kuhn, A., Loretan, P., Nierstrasz, O.: Consistent Layout for Thematic Software Maps. In: Proceedings of the 15th Working Conference on Reverse Engineering, WCRE 2008, pp. 209–218 (October 2008)
4. Steinbruckner, F., Lewerentz, C.: Representing Development History in Software Cities. In: Proceedings of the 5th International Symposium on Software Visualization, SOFTVIZ 2010, pp. 193–202. ACM, New York (2010), `http://csbob.swan.ac.uk/visWeek10/softvis/docs/p193.pdf`
5. Knight, C., Munro, M.: Comprehension with[in] Virtual Environment Visualizations. In: Proceedings of the Seventh International Workshop on Program Comprehension, Pittsburgh, PA, May 5-7, pp. 4–11.
6. Loretan, P.: Software Cartography. M.S. Thesis, University of Bern (2011)