The City Metaphor in Software Visualization

Clinton L. Jeffery
Department of Computer Science
University of Idaho
875 Perimeter Drive MS 1010
USA 83844, Moscow, Idaho
jeffery@cs.uidaho.edu

ABSTRACT

A city metaphor has become a popular method of visualizing properties of program code. This paper provides an overview of research projects that employ this metaphor for a wide range of software engineering tasks. Until now, projects employing the city metaphor have primarily focused on visualizing static and semi-static properties of software repositories, such as understanding how a program's source code structure is changing over time, and who is changing what. This paper compares these existing code cities and suggests likely avenues of future research.

Keywords
software visualization; code cities

1 INTRODUCTION

In the 1980's, works of science fiction such as TRON and Neuromancer popularized the notion that code and data could be visualized in 3D and understood by means of animation and anthropomorphization. A metaphor was needed in order for the human viewer to understand the visualization of abstract mathematical constructs that operate on scales beyond our normal cognitive range. Since humans are trained from an early age to understand how to navigate through man-made structures in the real world, it was natural to envision software as a city inhabited by many dynamic entities whose behavior could be understood by means of direct observation.

The first generation of virtual reality hardware in the late 1990's inspired researchers to attempt to implement the metaphor of the software city with headsets and SGI workstations. Almost simultaneously, rapid advances in the game industry and the internet made such shared online virtual 3D spaces popular and useful, even without specialized hardware.

However, the rate of development of software city visualizations has been modest and has followed the boom-bust cycle of virtual reality technologies, rather than the rapid rate at which computer game technology has advanced. The rate of advances in software cities has been limited by the difficulty of programming them, in addition to the varying rate at which funding for VR research has been available.

This paper surveys the published research on software city visualizations. The goal is to understand the range of results achieved, their interconnections, and what gaps or missing pieces are needed in order for this genre of software tools to become more widely developed and used.

By necessity, the paper draws a line at *city* visualizations of software. Thus, it does not consider some fine work exploring other metaphors for software, including metaphors that employ a galactic or solar system space metaphor, an atomic metaphor, a cellular metaphor, or other geographic metaphors such as islands [Kuhn08] [Misi18] [Schr18]. Any metaphor that helps users is valuable, and some of this related research has produced gorgeous images. Nevertheless, although some software might arguably be developed by means that fit one of these other metaphors, most software that gets studied is written as a semi-planned artificial human construct, for which the city metaphor is a better fit than other metaphors where human planning is not so obviously involved.

2 RESEARCH QUESTIONS

The paper first establishes via literature survey what coarse-grained, large-scale static and semi-static software properties have been depicted using software cities. It then asks the following research questions, which focus on the question of dynamic analysis of program execution behavior:

- What fine-grained, dynamic program execution behaviors have been depicted using software cities?
- Is it possible to integrate both large-scale static and fine-grained dynamic information into the same software city visualization?

3 EXISTING IMPLEMENTATIONS

Software World and Component City

Knight and Munro used the Maverik VR toolkit [Hubbold99] to develop a tool called Software World [Knig00] that is an early single-user example of the city metaphor for software visualization. Their layout maps Java classes to *districts*, within which methods are buildings whose height is one storey per 10 LOC. Parameters are depicted by exterior doors. Lighter and darker building colors indicate public and private. The mapping of one building onto one function seems suited to visualization of finer-grained details at the expense of scalability to larger systems.

The work of Knight and Munro was later subsumed by a tool called Component City, which depicts components as buildings, a larger granularity shared by most other research projects that employ the city metaphor [Charters02]. Component City was not based on the Software World implementation, but instead uses XML and XSLT to generate VRML that can be viewed in browsers with a suitable plug-in. The published papers on Software World and Component City do not include an evaluation, but their wide influence can be seen in their number of citations and the number of projects that utilize the same or similar metaphor.

Figure 1 shows two views from Software World. The upper image shows a street level view of many, varying sizes of methods. The lower image shows an overview of a district based on a reasonably sized class, with two methods that contain large amounts of code.

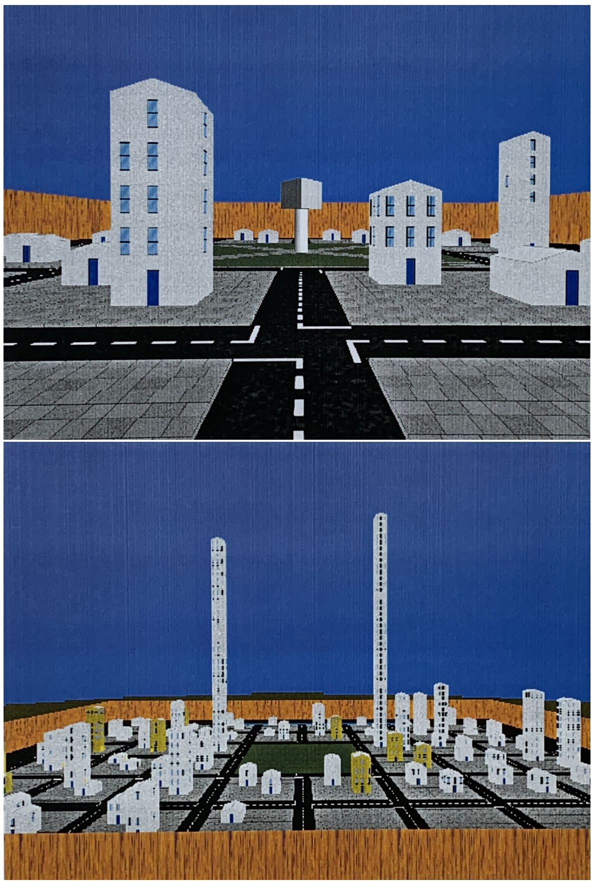
Figure 1: Software World (images courtesy of Claire Knight, from [Knigh00b])

Code City

Wettel et al developed a system called Code City in which a building represents an entire class, with a height proportional to the number of methods, rather than to lines of code [Wett07]. Length and width of the buildings were proportional to the number of member variables. The layout of buildings was performed using a treemap, atop a land whose elevation indicated nesting level of packages.

The increased scalability of representing an entire class as a building allows Code City to visualize large systems. consisting of many thousands of classes. The tool is used in the study of code evolution over time in software repositories. Code City was evaluated by demonstrating its scalability to large real-world software systems such as ArgoUML, Azureus, and VisualWorks, allowing views of as many as 8,000 classes. The work has been heavily cited or directly used in much software city visualization work that came after it. Figure 2 shows views of two large software systems from Code City.

Vizz3D

Panas et al developed an unnamed software city implementation for visualizing C++ programs using their visualization package named Vizz3D [Pana07]. Like Software World, their visualization depicts member functions as buildings; a blue platform serves as a pad for all the functions within a class. The number of lines of code in a member function is indicated by the texture of its exterior walls.

In contrast with most software city implementations, the Vizz3D based tool visually depicts more dependencies between functions and classes, such as "water towers" showing dependencies between header files and the classes defined within them. The system incorporates some dynamic analysis information, including the timing information produced by gprof, to augment a vast amount of static information that is depicted in the visualization.

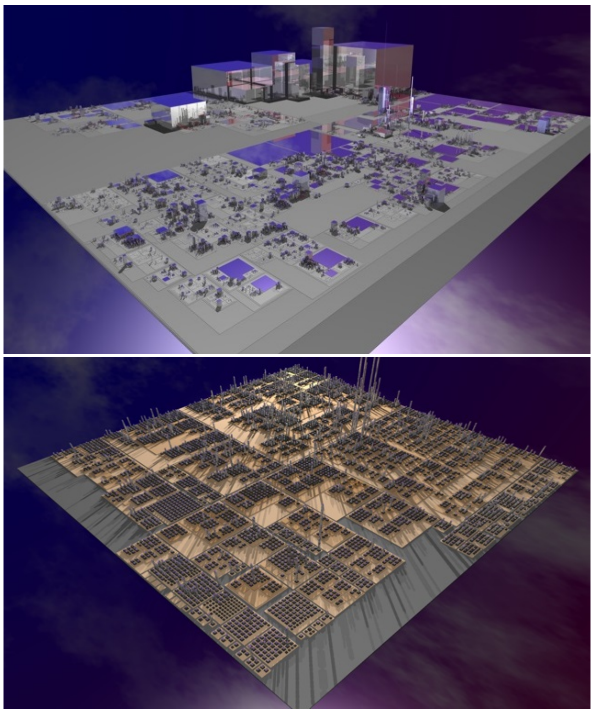

Figure 2: Code City (images courtesy of Michele Lanza)

Figure 3 shows a city generated using Vizz3d. The system is evaluated mainly by means of examples that demonstrate the range of stakeholders and roles (such as managers, testers, and engineers) that can utilize the single-view architectural visualization in performing their required tasks.

UML-City

Lange and Chaudron devised a UML-City visualization by combining two methods of reusing UML diagrams to produce visualizations in a system called MetricView Evolution [Lang07] [Lang07b]. A first method, called MetaView, focused on showing relationships between entities across reduced-size depictions of the entire set of UML diagrams available for a given software project. A second method, called MetricView, superimposed various software metrics data atop UML diagrams, as a vertical height, color, or shape.

The combination of MetaView and MetricView on large software systems produces an enriched, high-level view of the UML diagrams that resembles a city. The usefulness of these views was validated by a detailed user study of 100 M.S. students that demonstrates both improved speed and correctness of understanding UML diagrams when using the 3D visualizations than when trying to understand the software using a plethora of conventional, disconnected 2D UML diagrams.

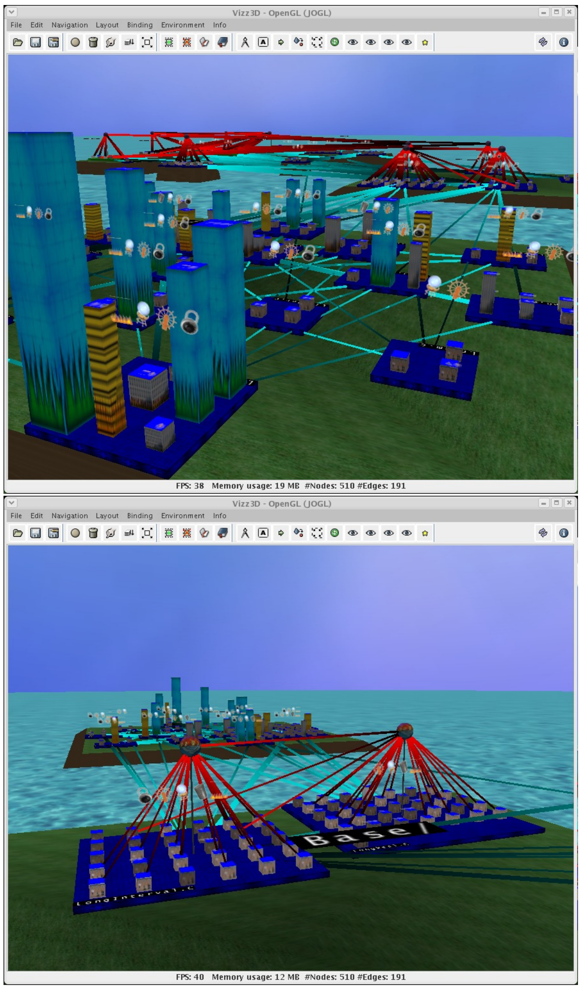
Vizz3D - OpenGL (JOGL)
File Edit Navigation Layout Binding Environment Info
FPS: 38 Memory usage: 19 MB #Nodes: 510 #Edges: 191
Vizz3D - OpenGL (JOGL)
File Edit Navigation Layout Binding Environment Info
Base/
FPS: 40 Memory usage: 12 MB #Nodes: 510 #Edges: 191

Figure 3: Vizz3d (images courtesy Thomas Epperly)

EvoStreets

Steinbruckner and Lewerentz created a system called EvoStreets [Stei10]. They adapt ideas from cartography, particularly the primary, secondary, and tertiary data models that respectively distinguish raw data, a detailed internal model, and a view-specific model that is render-friendly.

EvoStreets lays out classes, depicted as cylinders or cuboids, around streets, corresponding to directories or packages. The origin date of the various software components is indicated by their elevation given on a topographic map. Arguably, laying out structures around streets is more consistent with actual human cities than are space-filling techniques such as a treemap algorithm, although treemaps scale well.

EvoStreets has been used to study spatial orientation on VR head-mounted displays [Rude18]. Figure 4 shows an EvoStreets layout.

VizzAspectJ-3D

Bentrad and Meslati extended the city metaphor used in CodeCity to support the visualization of aspects in

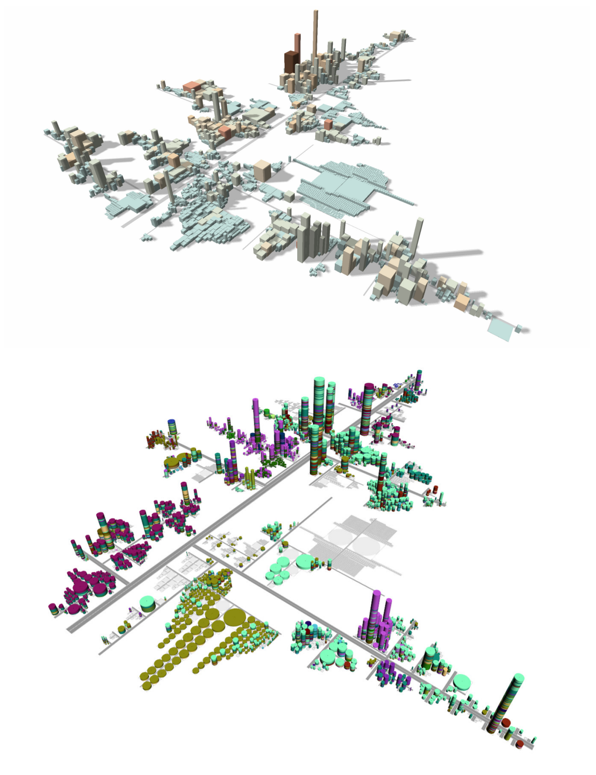

Figure 4: Evostreets (images ©Claus Lewerentz and Markus Uhlig, BTU Cottbus, 2016, by permission)

AspectJ programs [Bent11]. They developed an Eclipse plug-in in which both classes and aspects are buildings. The aspect buildings use the number of pointcuts and advice to determine building height. Different color-codings are used in complexity and dependencies views of the classes and aspects.

SkyscrapAR

Souza et al developed an augmented reality system called SkyscrapAR that visualizes Java systems with a metaphor similar to Code City except with building height denoting *code churn* [Souz12]. Churn refers to the idea that the number of lines of changes to a class may indicate its probability of containing software bugs. The visualization is projected onto a physical marker card that allows direct manipulation of the position, orientation, and scaling of the city, usually on a tabletop.

SynchroVis/ExplorViz

Waller et al invented a piece of software called SynchroVis in which classes are buildings [Wall13]. Each *instance* of a class is visualized as a storey on the building, allowing per-instance depiction of runtime dependencies and behavior.

SynchroVis depicts threads using colored arrows, allowing the visualization of concurrent behavior, communication and synchronization issues such as deadlock. Icons depict reasons why code may be suspended in a given instance, such as a method call to another object, or a wait on a semaphore. Figure 5 shows SynchroVis in action.

The software engineering group that developed SynchroVis also adapted their ExplorViz software to produce an Oculus Rift implementation with which to interact with a city metaphor [Fitt15]. This implementation allows translation, rotation, and scaling of the model using hand gestures input using a Kinect device.

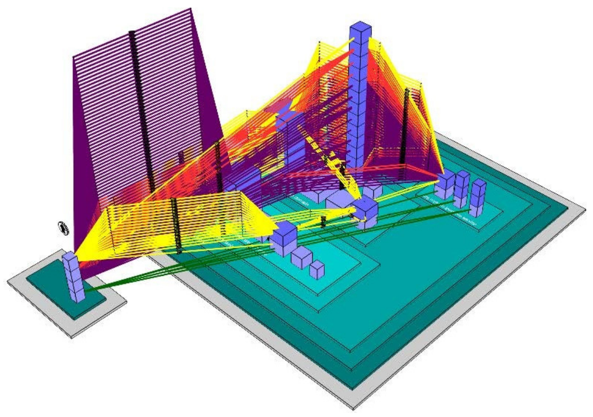

Figure 5: SynchroVis (image courtesy of Wilhelm Hasselbring)

CityVR

Merino et al created a VR application for the HTC Vive using the CodeCity system to generate the 3D visualization [Meri17]. The number of lines of code in a class is indicated via brightness. Source code for any class can be pulled up in a translucent 2D heads-up-display by pointing and clicking on the building. A similar system was used used on the HoloLens to explore whether immersive augmented reality technology will help to overcome usability issues with 3D visualizations in performing software engineering tasks [Meri18].

VR City

Vincur et al wrote a VR application for HTC Vive in which building layout is more organic because the amount of surface area contact between buildings is proportional to the amount of coupling between them [Vinc17]. For highly-coupled classes, the visualization may appear somewhat jigsaw-like. Although classes are represented by entire buidings with storeys representing methods as in Code City, the lengths and widths of storeys depict method metrics rather than class metrics. This simultaneously visualizes more information while resulting in building shapes that are more interesting and less uniform.

The VR City visualization allows the use of color-coding to highlight revision log information such as author commits, or dynamic information such as execution call traces. Figure 6 shows VR City's elegant use of color coding and translucence.

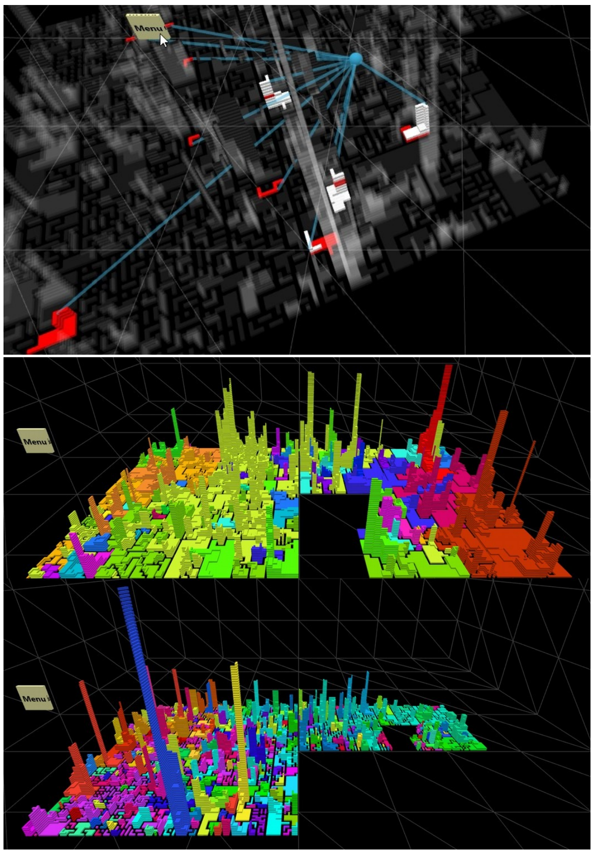

Figure 6: VR City (images courtesy of Ivan Polasek)

Code Park

Khaloo et al developed a system called Code Park that utilizes the city metaphor more for navigation of source code rather than for study of software architecture [Khal17]. Building size is proportional to code size but software engineering metrics are not the point of the tool.

The walls of buildings in Code Park contain source code with IDE-like syntax coloring and code navigation features. Users can toggle between birds-eye-view and first-person view on the ground. Movement between locations such as moving from a variable use to its definition is performed by animating up to birds-eye-view and then down to the new location, improving the user's orientation within the code base, compared with the more hyperlink-like teleportation that would be common in IDEs. Figure 7 shows Code Park's building layout (left) and a depiction on source code inside a building's walls.

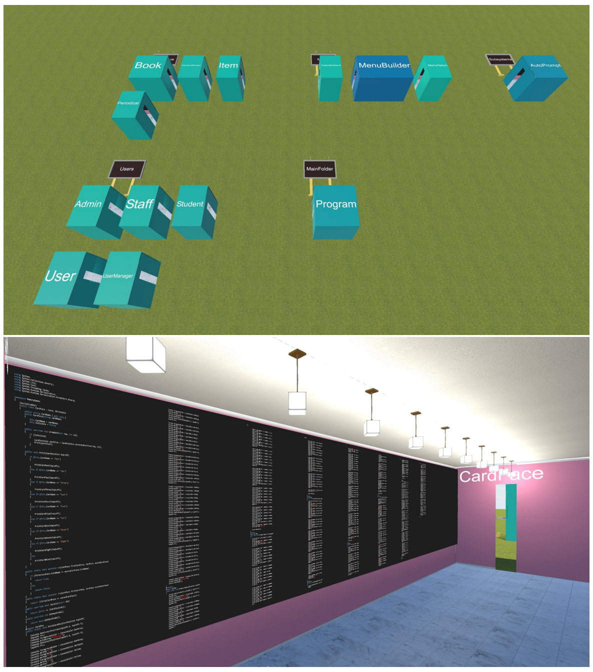

Figure 7: Code Park (images courtesy of Joseph LaViola)

High-Rise

Ogami et al developed a city-metaphor tool to deliver profile timing information in near real-time [Ogam17]. The height of buildings rises and falls over time, proportional to the amount of time consumed by the class in a fixed time frame such as the last L milliseconds. This design blurs the line between a city metaphor and a three dimensional bar chart.

By watching the profiles of different classes fluctuate over time, developers can notice issues that are not portrayed in conventional profiler output. It is clear that this visualization depicts transitions between major phases in an application very well.

LD-City

de Graaf and Khalili developed a city metaphor specifically for the representation of linked data, called LD-City [Graaf17]. Within that context, their visualization is used to detect anti-patterns such as *god classes* or *feature envy*. Their prototype implementation is built atop Three.js and related web technologies.

Comparison

Table 1. presents a comparison of existing software city visualizations. Column *Language* indicates the language(s) supported by the tool. Column *VR* specifies the type of VR hardware required, if any. Column *Building* conveys what a building denotes, such as a class or a function/method. Column *Src* indicates whether the tool depicts or integrates source code.

Column *Static* shows the static program properties depicted. Column *Dynamic* shows the dynamic program activity depicted. Column *Instr* gives the source/type of instrumentation used for dynamic program execution behavior, if any. An entry of *n/a* means an item is unknown or not applicable.

4 RESEARCH OPPORTUNITIES

Looking at Table 1, several obvious opportunities can be identified for near-term future research on visualizations using the city metaphor. Opportunities to better visualize information from static analysis are discussed first, followed by opportunities that involve incorporation of dynamic analysis information.

In this section, the research opportunities are sometimes couched in terms of suggested potential software-city-metaphor solutions, but really, better solutions may be empirically derived or simply invented by whomever incorporates that aspect into their software city visualization first. Many opportunities suggested here involve adding useful information to the city visualization. The research questions are often: what information is needed in a given context, and how to provide that information in a manner that is intuitive and does not result in cognitive overload.

Additional Static Information

None of the published works on the city metaphor deal with certain aspects of available static analysis information. For example, many of the published research works omit relationships between classes or functions beyond approximate juxtaposition of neighbors from the same file, package, or directory.

Tools that do visualize dependencies such as inheritance or caller-callee relationships risk overwhelming the viewer with too many interconnections, occluding too much of the cityscape. There is an opportunity to depict this information without overload, such as using animation, or an intermittent or context-sensitive depiction. It is possible to identify circumstances under which different dependencies merit visible rendition for a limited period of time, without overwhelming the viewer's other items under observation.

Integrating More Dynamic Information

Various existing tools feed dynamic analysis information into a city visualization post-mortem from log files, or at run-time. Either way, dynamic information has been far less utilized in city-metaphor visualizations. This is probably due more to the difficulty of extracting fine grained program execution behavior than to the challenges posed to the visualization researcher of how to depict dynamic information in the city.

In any case, it is long overdue for software city visualizations to incorporate more kinds of dynamic information from program executions. Such information is not suggested to reduce or replace the static information currently being visualized, but rather, to complement and leverage the static information visualization in order to visualize dynamic behavior that is intrinsically more difficult to understand.

Function or method calls and returns, timing information and thread operations have been depicted by existing city visualization tools. In future software cities should include depictions of behavior such as data structures and access patterns, stack and heap allocation, and input/output behavior.

Populating the City

In real life, cities exist in order to organize and facilitate the coordinated efforts of large numbers of humans. If the city is found to be useful as a metaphor at all in software visualization, its most likely purpose is to provide structure and context that assists the viewer in interpreting the dynamic aspects of program behavior, avoiding that cognitive overload that is otherwise ubiquitous when viewing software in its abstract complexity.

The population of objects within a software city might be graphic primitives such as spheres or cuboids. Adopting the terminology of videogames, a dynamic rendered object with which the user may interact may be termed an *entity*. It is a research question whether human viewers will find it more effective to visualize software objects via anthropomorphic entities, such as humans, robots, or pets that populate the city, or whether they will find abstract shapes just as effective.

The most obvious category of entity that would naturally construe a citizen of the city would be an application domain object: a class (or record/struct) instance where the corresponding type is from the application domain (as opposed to being an implementation artifact such as a glue data structure type). Criterion for citizenship might also include lifespan, since many or most objects in some programs do not exist long enough to deserve much investment from the visualization subsystem.

Although invoking a method on a citizen means that a thread of execution will be active within that citizen's class, we are often primarily interested in where that citizen object is being accessed from. Rather than drawing an arrow from caller to callee, citizens should stroll toward the code from where they are accessed. Some citizens will always be hanging out within the code (building) of some containing or highly-coupled object, while other citizens may continuously be going to and fro between many different sites from which they are referenced in the code. Such citizens, which may indicate design or performance bugs, will be very evident if they are portrayed walking around on the streets of the city.

Tool	Language	VR	Building	Src	Static	Dynamic	Instr
SoftwareWorld	Java	Maverik	function	n/a	LOC #methods public/private # parameters param. types	n/a	n/a
ComponentCity	XML	VRML	compon.	n/a	func. attributes	n/a	n/a
CodeCity	SmallTalk Java C++	n/a	class	n/a	# methods # attributes package struct.	n/a	n/a
Vizz3D	C/C++	n/a	function	n/a	LOC complexity call graphs contains inheritance str. conn. comp.	gprof	none (`-pg`)
SkyscrapAR	Java	AR	class	n/a	LOC churn	n/a	n/a
UML-City	UML	n/a	class various	n/a	various metrics author	n/a	n/a
VizzAspectJ	Java AspectJ	n/a	class aspect	n/a	# methods # pointcuts # advices	n/a	n/a
EvoStreets	Java	n/a	class	n/a	module age coupling # dependencies module size last mod. date author	n/a	n/a
SynchroVis ExplorVis	Java	Rift	class	n/a	inheritance implementation association	instances calls thread op	Kieker traces
CityVR	Java/C++	Vive	class	yes	LOC # methods # attributes	n/a	n/a
VR City	Java	Vive	class	yes	LOC # methods author coupling	trace loc.	inTrace traces
Code Park	C#	n/a	class	yes	size method names	n/a	n/a
High-Rise	Java	n/a	function	no	n/a	time	ASM injection
LD-City	LD-R	n/a	(dynamic)	n/a	#instances # properties	n/a	n/a

Table 1: Feature Sets of City-Metaphor Visualizations

Time Scales

Understanding the execution behavior of software systems includes a need to understand dynamic behavior at widely ranging time scales, perhaps from sub-microsecond granularity to things that challenge human patience. A visualization tool can implement yet-another slider to control the speed of execution, and leave the user in control, but users have better things to think about. Alice in Wonderland and Star Trek are suggestive, but perhaps Dr. Who's *Weeping Angels* provide a better metaphor and potential user interface mechanism: the speed of execution behavior should automatically stop while an entity is being observed, either via explicit pointer selection or by entering a user's center of vision.

Dealing with Dynamic Data

Some of the more dynamic oriented software cities animate the buildings themselves, with building height depicting the number of instances, or the amount of time spent in a class. Separate from the scalability problem introduced by such approaches, they have a metaphor problem. Although software cities are by no means required to be "realistic" or follow rules that would apply to human cities, the value of the metaphor depends on an analogy. Humans can recognize buildings as solid artificial structures that have a purpose and that can be navigated around.

While it is easy in software to grow and shrink a set of cuboids in an animated 3D bar chart, the notion of buildings rapidly growing and shrinking violates the city metaphor and makes the entities softer and more transient. Cities, streets, and topography are a more natural fit to depict static or semi-static data, providing a context in which to interpret the vast amounts of dynamic information that humans need to understand.

Whereas the challenge of visualizing and making comprehensible very large bodies of code is a semi-static one, a software city can portray large amounts of dynamic data: on the stack, on the heap, or entering and exiting via I/O ports. In most cases, the primary challenge is the depiction of large amounts of information, much of which is very short-lived.

Short-lived information may be categorized by its importance. If it is important enough that the user may need to interact with it, it may require a temporal stop or slowdown so that the user has time to take it in. If user interaction is not required, such information may still be useful, and might by depicted by means of a more subtle visual effect rather than an entity.

Multi-user Shared Viewing

From the earliest example of Software World, researchers have identified a networked multi-user environment as an objective for software city visualizations, but this feature adds enough implementation challenges that it is usually omitted. A substantial opportunity exists for a cloud-based software city whose open infrastucture is accessed and shared by multiple research groups.

Multi-Lingual City Visualization

Several existing projects support two or more mainstream languages such as Java and C++ by means of some standard common tool for obtaining static information, such as an Eclipse IDE plugin. While such multi-lingual support is compelling, no standard exists for high-level language-neutral dynamic analysis information. A standard is needed that provides higher-level dynamic information than that available via virtual machines that support managed code in many languages, such as the JVM or .Net.

In addition, many large software systems are written in a combination of languages. For example, a software city might depict a program written in a higher level language (such as Python or Java) with an understanding that its subterranean or aquatic regions depict the behavior of intermediate or lower-level elements, such as ones invoked through a native interface to libraries written in C or C++.

5 CONCLUSION

The city metaphor has been adopted by a growing number of research groups for the visualization of large, complex software systems. Existing research has been successful in scaling to real-world software with a large number of classes, for which many static properties of the program code can be immediately observed.

Compared with the static information for which they were originally developed, city visualizations have not been applied as much to the task of making dynamic program execution behavior visible and understandable. In that domain software cities are potentially even more useful than the applications where it has thusfar been applied. Tremendous potential exists for the research groups that apply this metaphor to a broader range of dynamic program execution behavior. Such projects will require access to general purpose, high-performance execution frameworks capable of reporting a wide range of execution events, such as JVMTI [JVMTI] or Alamo [Jeffery98].

Several of the recent research advances that use the city metaphor focus on VR, despite head-mounted displays still being inadequate for displaying large quantities of text. This has resulted somewhat in a bifurcation between high-text software cities that run on desktops, and low-text software cities that run on VR systems.

6 REFERENCES

[Bent11] Bentrad, S. and Meslati, D. 2D and 3D Visualization of AspectJ Programs. In 10th International Symposium on Programming and Systems, pp. 183-190, IEEE 2011.

[Charters02] Charters, S.M., Knight, C., Thomas, N., and Munro, M. Visualization for informed decision making; from code to components. In SEKE 02: Intl. Conference on Software Engineering and Knowledge Engineering, pp. 765-772, ACM Press 2002.

[Fitt15] Fittkau, F., Krause, A., and Hasselbring, W. Exploring Software Cities in Virtual Reality, in 3rd Working Conference on Software Visualization, IEEE, pp. 130-134, 2015.

[Graaf17] de Graaf, K.A., and Khalili, A. Visualizing Linked Data as Habitable Cities. VOILA 2017, Third Intl. Workshop on Visualization and Interaction for Ontologies and Linked Data, Vienna, CEUR-WS vol. 1947, pp. 131-138, 2017.

[Hubbold99] Hubbold, R., Cook, J., Keates, M., Gibson, S., Howard, T., Murta, A., West, A., and Pettifer, S. GNU/MAVERIK: A micro-kernel for large-scale virtual environments. VRST'99, pp. 66-73, ACM, 1999.

[Jeffery98] Jeffery, C., Zhou, W., Templer, K., and Brazell, M. A Lightweight Architecture for Program Execution Monitoring. Proc. of PASTE'98. ACM SIGPLAN Notices 33(7).

[JVMTI] Java Virtual Machine Tool Interface (JVM TI). https://docs.oracle.com/javase/8/docs/technotes/guides/jvmti.

[Khal17] Khaloo, P., Maghoumi, M., Taranta, E., Bettner, D., and Laviola, J. Code Park: A New 3D Code Visualization Tool, in Working Conference on Software Visualization, IEEE, pp. 43-53, 2017.

[Knig00] Knight, C., and Munro, M. Virtual but Visible Software. 2000 IEEE Conference on Information Visualization. London, IEEE, pp 198-205.

[Knigh00b] Knight, C. Virtual software in reality. Ph.D. Thesis, University of Durham, 2000.

[Kuhn08] Kuhn, A., Loretan, P., Nierstrasz, O. Consistent Layout for Thematic Software Maps, Proceedings of the 15th Working Conference on Reverse Engineering, WCRE '08. pp. 209-218, 2008.

[Lang07] Lange, C. and Chaudron, M. Interactive Views to Improve the Comprehension of UML Models - An Experimental Validation, Proceedings of the 15th IEEE International Conference on Program Comprehension, pp. 221-230, 2007.

[Lang07b] Lange, C., Wijns, M., and Chaudron, M. A Visualization Framework for Task-Oriented Modeling Using UML, 40th Annual Hawaii International Conference on System Sciences, p. 289a, 2007.

[Meri17] Merino, L., Ghafari, M., Anslow, C. and Nierstrasz, O. CityVR: Gameful Software Visualization. in International Conference on Software Maintenance and Evolution (ICSME), Shanghai, IEEE, pp. 633-637, 2017.

[Meri18] Merino, Bergel, and Nierstrasz. Overcoming Issues of 3D Software Visualization through Immersive Augmented Reality, in Working Conference on Software Visualization, pp. 54-64, IEEE, 2018.

[Misi18] Misiak. et al. IslandViz: A Tool for Visualizing Modular Software Systems in Virtual Reality. In Working Conference on Software Visualization, IEEE, pp. 112-116, 2018.

[Ogam17] Ogami, K., Kula, R.G., Hata, H., Ishio, T. and Matsumoto, K. Using High Rising Cities to Visualize Performance in Real-Time, in Working Conference on Software Visualization, Shanghai, IEEE, pp. 33-42, 2017.

[Pana07] Panas, T., Epperly, T., Quinlan, D., Saebjornsen, A. and Vuduc, R. Communicating Software Architecture using a Unified Single-View Visualization, in Proc. ICECCS 2007, Auckland, IEEE, pp. 217-228.

[Rude18] Rüdel, M.O., Ganser, J., and Koschke, R. A Controlled Experiment on Spatial Orientation in VR-Based Software Cities, in Working Conference on Software Visualization, pp. 21-31, IEEE, 2018.

[Schr18] Schreiber and Misiak. Visualizing Software Architectures in Virtual Reality with an Island Metaphor. VAMR 2018, International Conference on Virtual, Augmented, and Mixed Reality, pp. 168-182, 2018.

[Souz12] Souza, R., Silva, B., Mendes, T. and Mendonça, M. SkyscrapAR: An Augmented Reality Visualization for Software Evolution, in II Workshop Brasileiro de Visualizaçao de Software, Natal, RN, Brasilian Computing Society, 2012, pp. 17-24.

[Stei10] Steinbrückner, F. and Lewerentz, C. Representing Development History in Software Cities, in Proceedings of the 5th international symposium on Software visualization, SOFTVIZ 2010, ACM, New York, pp. 193-202.

[Vinc17] Vincur, J., Navrat, P., and Polasek, I. VR City: Software Analysis in a Virtual Reality Environment, in International Conference on Software Quality, Reliability and Security Companion (QRS-C), Prague, IEEE, pp. 509-516, 2017.

[Wall13] Waller, J., Wulf, C., Fittkau, F., Doring, P., and Hasselbring, W. SynchroVis: 3D Visualization of Monitoring Traces in the City Metaphor for Analyzing Concurrency. in First IEEE Working Conference on Software Visualization, Eindhoven, Netherlands, IEEE, pp. 1-4, 2013.

[Wett07] Wettel, R. and 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.