# Improving Error Notification Comprehension through Visual Overlays in IDEs

Titus Barik
North Carolina State University
tbarik@ncsu.edu

***Abstract*—Error notifications, as presented by modern integrated development environments, are cryptic and confusing to developers. My dissertation research will demonstrate that modifying production compilers to expose detailed semantics about compilation errors is feasible, and that these semantics can be leveraged through diagrammatic representations using visual overlays on the source code to significantly improve compiler error notification comprehension.**

## I. Introduction

The traditional abstraction of compilers is that they are sophisticated black boxes [1]. In this abstraction, programmers write source code, from which the compiler either successfully generates an executable, or provides a notification in the event that the compiler is unable to perform the task. Modern integrated development environments (IDEs) challenge the utility of this opaque abstraction. For example, the Eclipse IDE includes an incremental compiler whose API is intentionally exposed so that the IDE can provide features like code assistance and support for refactoring.[1] Similarly, the Microsoft compiler, Roslyn, treats *compilers as a platform*, postulating that the design of more intelligent and expressive tools for use in the IDE will require compiler transparency and the exposing of deep knowledge currently internal to the compilation processes.[2]

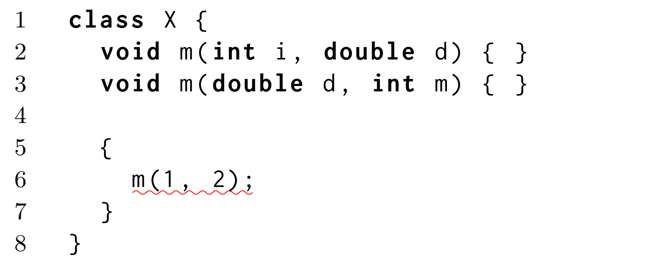
My work incorporates this philosophy of compilers as a platform to the domain of program comprehension, specifically, to the research goal of improving the comprehension of *compiler error notifications*. I hypothesize that compilers can be modified to expose detailed semantics about compilation errors, and that these semantics can be leveraged by *visual overlays* on the source code to significantly improve error notification comprehension. The contribution of my research will demonstrate: 1) that diagrams on the source code, or visual overlays, are an appropriate representation that aligns cognitively with the way developers reason about error notifications, 2) that such a visual representation can be generated algorithmically through conceptual and visual *ontologies*, and 3) that a concrete implementation of such a tool is useful, usable, and improves error notification comprehension.

## II. Why Improve Error Notification Comprehension Using Visual Overlays?

There are a significant number of studies that indicate that error notifications today are cryptic and confusing for developers; for a detailed literature review, see [2]. In part, this is attributable not only to the complexity of modern programming languages, but also because IDEs only partially leverage the range of visual affordances available. For example, consider the error notification for the source code listing in Figure 1a, a situation in which the method call `m(1, 2)` is ambiguous due to method overloading. The OpenJDK compiler output is:

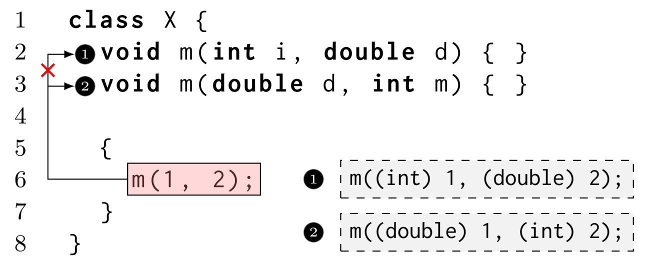
```
reference to m is ambiguous, both method
m(int,double) in X and method m(double,int) in X
match
```

In addition to the text notification, the IDE provides a red wavy underline overlay on Line 6 to indicate the point of the error. It can do so because the compiler provides the IDE with the description of the message, the line number, and column number through an *error object*.[3] Though the underlying semantics of the error are provided as text to the developer, the IDE cannot take advantage of a natural language representation.



```
1  class X {
2    void m(int i, double d) { }
3    void m(double d, int m) { }
4
5    {
6      m(1, 2);
7    }
8  }
```

(a) Traditional error notifications in IDEs



```
1  class X {
2    ❶void m(int i, double d) { }
3    ❷void m(double d, int m) { }
4
5    {
6      m(1, 2);          ❶ m((int) 1, (double) 2);
7    }
8  }                     ❷ m((double) 1, (int) 2);
```

(b) Explanatory error notifications in IDEs

Fig. 1. A comparison of (a) traditional error notifications and (b) explanatory error notifications in IDEs. Expressive representations are enabled by exposing compilers internals for use by the IDE.

[1]http://www.eclipse.org/jdt/core/

[2]http://msdn.microsoft.com/en-us/vstudio/roslyn.aspx

[3]For example, the `ErrorItem` object in the Developer Tools Environment API in Visual Studio 2013 offers essentially the following properties: column, description, error level, file name, line, and project.

By contrast, consider a more expressive notification, such as through my prototype in Figure 1b. Such a notification can be generated when the compiler exposes internal semantics to the IDE. As before, the IDE indicates the point of the problem, through ▭ (red box). Additionally, the compiler informs the IDE that the concept of CLASH, or conflict between elements in the program, has occurred. The compiler can do so because of a common ontology that enables it to express such ideas to other systems. Visually, the IDE translates the concept of CLASH and marks the associations ↗ (arrows), and the conflict between them × (red cross). Furthermore, the explanation draws visual attention through enumerations, such as ❶ and ❷ (numbered black circles). For each item in the enumeration, the compiler conveys its internal reasoning for why it considers this to be a problem, by providing the IDE an instantiation of the issue, which is rendered as ⬚ (dashed gray box).

Developers can potentially benefit from this choice of diagrammatic representation. Self-explaining has been shown to be an effective metacognitive strategy for understanding materials, and diagrams can promote the self-explanation effect [3]. Importantly, this effect is significantly increased when comparing diagrams against text-only representations due to computational offloading, re-representation, and graphical constraining [3]. High explanation fidelity can also help developers build useful mental models [4]. And having the system explicate *why* it behaves a certain way can result in a better understanding of the issue as well as stronger trust in the tool [5].

One challenge is the way in which the compiler and IDE should communicate their semantics, so that these visualizations can be generated on arbitrary source files. Asenov and Müller have applied a domain-specific language, or grammar, to support visualization of code contracts [6], with some success. Consequently, I think the development of ontologies for error notifications can offer similar benefits for the automatic visualization of error explanations, by providing a consistent an unified vocabulary, such as CLASH. My initial work describes the details of such an ontology for error notifications [7].

## III. EVALUATION PLAN

**Representation alignment to mental models.** If visual overlays act as cognitive offloading for developers, then the visual representation should align with the way in which developers would reason about notifications. I have conducted a preliminary evaluation on mental models and diagrammatic representations by asking undergraduate programmers to hand-draw annotations on printed source code. Our early results show that participants generally used the same markings, such as arrows and $\times$, to consistently describe concepts, but a formal study is still needed. My expectation is that this study will use a combination of eye tracking and cognitive models to validate the appropriateness of the representation.

**Ontologies for mapping concepts to visualizations.** Modern compilers contain an overwhelming number of error notifications, but many errors share common conceptual explanations. I will conduct an empirical investigation by examining a corpus of error notifications from production compilers. A taxonomic approach will be used to construct a conceptual ontology that classifies these error notifications into semantic concepts. The semantic concepts will be mapped to a visual ontology, so that a visualization in an IDE can accept one or more concepts from the compiler, and render them appropriately.

**A software tool for expressive error notifications.** I will conduct a user study in which developers will explain error notifications through the Visual Studio IDE, with a modified version of Roslyn that supports error notification semantics. Experimental frameworks, such as Barista, have enabled the creation of visual representations on source code within IDEs [8], and Visual Studio now offers the majority of these capabilities through its Windows Presentation Framework, making it an appropriate environment in which to develop such a tool. The study will compare developer comprehension using our novel visualization technique against the traditional error notifications offered by the baseline IDE.

## IV. CONCLUSION

The visual affordances available to tool designers within integrated development environments are underutilized. By exposing compiler internals to the IDE through a conceptual to visual mapping, visualizations can be generated algorithmically and be made more expressive. My dissertation work will apply these ideas to improving compiler error notification comprehension, but I believe that my work can be extended by researchers to develop diagrammatic representations for other software engineering tasks, such as conveying unit test coverage or explaining changes between multiple versions of the code.

## ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under Grant No. 1217700. I thank the Software Engineering group at ABB Corporate Research for their support. I also thank my advisor, Dr. Emerson Murphy-Hill, for his insights and support.

## REFERENCES

[1] C. Lattner and V. Adve, “LLVM: A compilation framework for lifelong program analysis & transformation,” in *International Symposium on Code Generation and Optimization*, 2004, pp. 75–86.

[2] V. J. Traver, “On compiler error messages: What they say and what they mean,” *Advances in Human-Computer Interaction*, vol. 2010, pp. 1–26, 2010.

[3] S. Ainsworth and A. T. Loizou, “The effects of self-explaining when learning with text or diagrams,” *Cognitive Science*, vol. 27, no. 4, pp. 669–681, Aug. 2003.

[4] T. Kulesza, S. Stumpf, M. Burnett, S. Yang, I. Kwan, and W.-K. Wong, “Too much, too little, or just right? Ways explanations impact end users’ mental models,” in *VL/HCC*, Sep. 2013, pp. 3–10.

[5] B. Y. Lim, A. K. Dey, and D. Avrahami, “Why and why not explanations improve the intelligibility of context-aware intelligent systems,” in *CHI*, Apr. 2009, pp. 2119–2128.

[6] D. Asenov and P. Müller, “Customizing the visualization and interaction for embedded domain-specific languages in a structured editor,” in *VL/HCC*, Sep. 2013, pp. 127–130.

[7] T. Barik, J. Witschey, B. Johnson, and E. Murphy-hill, “Compiler error notifications revisited: An interaction-first approach for helping developers more effectively comprehend and resolve error notifications,” in *ICSE NIER*, Jun. 2014.

[8] A. J. Ko and B. A. Myers, “Barista: An implementation framework for enabling new tools, interaction techniques, and views in code editors,” in *CHI*, Apr. 2006, pp. 387–396.