Hi, I’m Emily Hill, a PhD student at the University of Delaware. The work I’ll discuss today was done in collaboration with my advisors Lori Pollock and Vijay Shanker.
Exploring a program to solve a software maintenance task can be extremely challenging. Consider for example, exploring the code for a software maintenance task, from a starting method.

<Click> The developer knows of one relevant method to start exploring from. Where to look next? One way is to explore structural information such as the call graph.

In a typical scenario, a developer is given a maintenance task -- either a bug report or feature request. Oftentimes they have existing knowledge of the code, or a tip from an expert on the system and know of at least one method that’s relevant to the maintenance task. The developer needs to locate the rest of the code that’s relevant to the maintenance task. One common way to do this is by exploring the structure of the program, such as the call graph. But the call graph is enormous. In fact, only a small portion of the graph may actually be relevant to the maintenance task, which we term the relevant neighborhood. Our research focuses on ways to automatically identify this relevant neighborhood. This work is similar in spirit to concern or feature location.

Caveat: static program structure models may not capture all meaningful relationships between program elements.
Let’s examine ways to automatically identify the relevant neighborhood with an example scenario, which I will use as a running example throughout the remainder of the talk.

<Click> The general DoAction method handles all user-triggered events.
One way is to identify the relevant neighborhood is to use structural information, such as traversing the call graph from a starting point. As you can see from the example, structural information successfully identifies the two relevant methods, but returns too many results to be practically relevant. Further, notice that a quick scan of the callees, such as the Eclipse Call Hierarchy view, might cause the developer to miss the relevant method `DoPasteFromClipboard`, because it doesn’t contain either of the words 'add' or 'auction'. In addition, what if you want recursively explore the call graph? It grows too large too quickly to be useful.
Alternative: Exploring with only lexical information

Looking for: ‘add auction’ trigger in 1902 methods (159 files, 23KLOC)
- Use lexical information from comments & identifiers
- Search with query ‘add*auction’
  - 91 query matches in 50 methods
  - Only 2/50 methods are relevant

+ Locates globally relevant items
- But too many irrelevant

One alternative is to use lexical information in the comments and identifiers of the program. The best query I could come up with is the pattern ‘add wildcard auction’. This query successfully trims the number of methods the developer needs to explore from 1900 to 50, but again, only 2 of these methods are relevant.
Dora tries to combine the best of both worlds by using structural information to guide the exploration from a starting point, and use lexical information to prune irrelevant methods, leaving us with the relevant neighborhood.

Transition: So, why is it important to guide the developer’s exploration process?
Software Maintenance: Dora to the rescue

- Developers spend more time finding and understanding code than actually fixing bugs [Kersten & Murphy 2005, Ko et al. 2005]
- Critical need for automated tools to help developers explore and understand today’s large & complex software

→ **Key Contribution**: Automated tools can use program structure and identifier names to save the developer time and effort

Why are accurate exploration techniques important? Exploration techniques are important to facilitate software maintenance.

The person fixing the bug may not be the person who originally implemented the code.
To that end, we have developed a prototype tool.
The Dora Approach

Prune irrelevant structural edges from seed

1. Obtain set of methods one call edge away from seed
2. Determine each method’s relevance to query
   - Calculate lexical-based relevance score
3. Prune low-scored methods from neighborhood, using threshold
4. Recursively explore
The Dora Approach

Prune irrelevant structural edges from seed

1. Obtain set of methods one call edge away from seed
2. Determine each method’s relevance to query
   - Calculate lexical-based relevance score
3. Prune low-scored methods from neighborhood, using threshold
4. Recursively explore

I’ll go into more detail on how we calculate the relevance score, which is the main contribution of this work.
To calculate our lexical relevance score we have modified a traditional scoring mechanism from information retrieval, called tf-idf. We have modified the approach slightly to perform better on programs. The first component of tf-idf is the term frequency. Going back to our running example, consider the relevant method DoAdd, which has 6 occurrences of the query terms, ‘add’ and ‘auction’. In contrast, the irrelevant method DeleteComment has only two occurrences of the query terms. Thus, term frequency helps us differentiate between relevant and irrelevant methods.

Notice that DeleteComment has only two occurrences of the term ‘auction’. But we’re working with an “auction sniping” program. It is conceivable that the term ‘auction’ appears all over the program, in relevant as well as irrelevant methods.
Calculating Relevance Score: Inverse Document Frequency

- What about terms that appear all over the program?
- Use inverse document frequency (idf)
  - **Intuition:** Highly weight terms that appear in few documents/methods
    - Terms appearing all over program not good discriminators
    - Don’t separate relevant from irrelevant methods
  - Number of methods / number of methods containing the term

Query: 'add auction'

<table>
<thead>
<tr>
<th>Method</th>
<th>idf</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>1902/1311 = 1.45</td>
</tr>
<tr>
<td>auction</td>
<td>1902/415 = 4.58</td>
</tr>
<tr>
<td>add</td>
<td>1902/258 = 7.37</td>
</tr>
<tr>
<td>password</td>
<td>1902/29 = 65.59</td>
</tr>
</tbody>
</table>

Information retrieval (IR) commonly deals with this by using inverse document frequency in conjunction with term frequency. As you can see from the example, terms appearing in many methods have a lower idf than terms appearing in few methods.
Calculating Relevance Score: TF-IDF

- Score based on method query term frequency (tf)
- Multiplied by natural log of inverse document frequency (idf)

Query: 'add auction'

6 query term occurrences

Add
tf-idf = 4 \cdot \ln(7.37) + 2 \cdot \ln(4.58) = 11.03

Auction

Only 2 occurrences

Auction

tf-idf = 2 \cdot \ln(4.58) = 3.04

These two components are multiplied together to form a tf-idf score. From this concrete example, you can see that tf-idf helps us to better differentiate from relevant and irrelevant methods than term frequency alone.
The second component of our relevance score is the location of the terms – how the terms are used in the program.
Dora’s Relevance Score

- **Factors**
  - \( \sum \text{tf-idf for each query term in the method name} \)
  - \( \sum \text{tf-idf for each query term in the method body} \)
  - the number of statements in the method

- **How to determine weights?**
  - Applied logistic regression
  - Trained on methods from 9 concerns in previous concern location tool evaluation [Shepherd et al. 2007]
    (A concern is a conceptual unit of the software, such as a feature, requirement, design idiom, or implementation mechanism [Robillard & Murphy 2007].)

- **For details, see paper**
Example: Dora explores ‘add auction’ trigger

Scores from DoAction() seed:

- Identified as relevant with 0.5 threshold
  - DoAdd() (0.93)
  - DoPasteFromClipboard() (0.60)
- With only one false positive
  - DoSave() (0.52)
Experimental Evaluation: Research Questions

- Does an integrated lexical- and structural-based approach outperform a purely structural approach?
- Is a sophisticated lexical scoring technique required, or are naïve lexical scoring techniques sufficient to identify the relevant neighborhood?

Although the case study was a nice affirmation of our approach, we didn’t rely on it for our evaluation. We performed an experimental evaluation investigating the following two research questions.
Experimental Evaluation: Design

- **Gold Set:** 8 concerns from 4 Java programs, manually mapped by 3 independent developers [Robillard et al. 2007]
- **Compare** 4 exploration techniques: 1 structural, 3 lexical + structural
  - **Structural:** Suade [Robillard 2005]
    - Automatically generates exploration suggestions from seed set
    - Elements that have few connections outside the seed set are more relevant
    - Uses caller/callee & field def-use information to make recommendations
  - **Lexical + Structural:** Dora (sophisticated)
  - **Lexical + Structural:** boolean AND (naïve)
  - **Lexical + Structural:** boolean OR (naïve)
Experimental Evaluation: Design

- **Gold Set:** 8 concerns from 4 Java programs, manually mapped by 3 independent developers [Robillard et al. 2007]
- **Compare** 4 exploration techniques: 1 structural, 3 lexical + structural
- **Measures:** Precision (P), Recall (R), & F Measure (F)
  - \( P = \frac{TP}{TP+FP} \) (Are the results returned actually relevant?)
  - \( R = \frac{TP}{TP+FN} \) (How close are the returned results to the gold set?)
  - \( F = \frac{2PR}{P+R} \) (High when P & R are similarly high)

The measures we use are standard in the information retrieval community.

To give you an example, when searching Google, precision is high when no results (or one correct result) are returned, recall is high when every web page on the internet is returned.
## Experimental Evaluation: Design

- **Gold Set**: 8 concerns from 4 Java programs, manually mapped by 3 independent developers [Robillard et al. 2007]
- **Compare**: 4 exploration techniques: 1 structural, 3 lexical + structural
- **Measures**: Precision (P), Recall (R), & F Measure (F)
- **Methodology**
  - For each exploration technique $t$
    - For each method $m$ in the gold set
      - Score each caller & callee of $m$ with $t$
      - Calculate P, R, & F for $m$ with $t$
  - 160 seed methods, 1885 call edges (with overlap)
Results: All Concerns

- Dora outperforms Suade with statistical significance ($\alpha = 0.05$)
- Dora, OR, and Suade perform significantly better than AND
- Dora and Suade not significantly different from OR ($\alpha = 0.05$)
  - OR > Suade, $p = 0.43$
  - Dora > OR, $p = 0.033$
  - Dora > Suade, $p = 0.0037$
- Dora achieves 100% P & R for 25% of the data—more than any other technique

A boxplot of the overall results for all 8 concerns, showing the F Measure distribution for each technique.
- Shaded box represents the middle 50% of the data, from the 25th to 75th percentiles; the horizontal bar is the median, and the plus is the mean.
- We conservatively divided our experimentwise alpha by 6, so our per comparison (t-test) alpha is 0.008

- Dora returns the exact gold set for a quarter of the date, more than any other technique.
Results: By Concern

- Overall trend also seen for most concerns
- Exceptions: 9 & 12
  - AND had much higher precision
  - Relevant methods contained both query terms
Experimental Evaluation: Result Summary

- Does an integrated lexical- and structural-based approach (Dora) outperform a purely structural approach (Suade)?
  - Dora outperforms Suade with statistical significance ($\alpha = 0.05$)

- Is a sophisticated lexical scoring technique required, or are naive lexical scoring techniques sufficient to identify the relevant neighborhood?
  - Although not statistically significant, Dora outperforms OR
  - Dora, Suade, & OR outperform AND ($\alpha = 0.05$)
  - Integrated lexical- and structural-based approaches can outperform purely structural, but not all lexical scoring mechanisms are sufficient to do so
• Didn’t compare against slicing in the experiment because at the time we couldn’t find a Java slicer that worked on our benchmarks.

• Only included the most relevant work

• IR = information retrieval

• Currently don’t use comments because they could be out of date, canned (auto generated), copy-pasted with no changes, and in general are less likely to reflect developer’s intentions -- this is true of ids as well, but less likely than comments.
Future Work

- Automatically find starting seeds
- Use more sophisticated lexical information
  - Synonyms, topic words (currency, price related to bidding)
  - Abbreviation expansion
- Evaluate on slicing
Conclusion

→ Integrated lexical- and structural-based approaches outperform purely structural ones

www.cis.udel.edu/~hill/dora

This work was supported by an NSF Graduate Research Fellowship and Award CCF-0702401.
Appendix

Additional Materials
Dora’s Relevance Score

- **Developing the relevance score**
  - Used logistic regression:
    - predicts values between 0 and 1
  - Logistic regression outputs ‘x’ of the score

- **Training the model**
  - Used methods from 9 concerns in previous concern location tool evaluation [Shepherd 2007]

- **The model:**
  $$x = -0.5 + 2.5 \times \text{bin} + \text{name} + 0.5 \times \text{statement}$$

- **Where…**
  - bin = binary (1 if java file exists, 0 otherwise)
  - name = \(\sum\) tf-idf for each query term in the method name
  - statement = \(\sum\) tf-idf for each query term in a method statement

  the number of statements in the method

\[ score = \frac{e^x}{1 + e^x} \]
Results: Threshold

Figure 3: Precision-Recall Graph. Suede and Dora were evaluated at various thresholds ranging from 0 to 1. (AND and OR require no threshold). Each point represents precision and recall averaged over a given threshold, with decreasing threshold values from left to right.