Combining Information Retrieval and Relevance Feedback for Concept Location

Sonia Haiduc

- WSU CS Graduate Seminar -
  Jan 19, 2010
Software changes

• **Software maintenance**: 50-90% of the global costs of software

• Why does software change?
  – Requirements change

• What types of changes?
  – Implement new functionality
  – Change existing functionality
  – Fix bugs, etc.

How do we change software?
Process of software change

- Change request
- Concept location
- Impact analysis
- Implementation
- Change Propagation
- Testing
Concept location

• Starts with a change request (bug report, new feature request)

• Ends with the location where the change is to start (e.g., a class or a method)

• Approaches:
  – Static: searching, navigating source code based on dependencies between entities (inheritance, method calls, etc.), etc.
  – Dynamic: analyze execution traces
  – Combined
Concept location example

Bug #2373904, ATunes v1.10.0

- **Title:** Fullscreen doesn't work for Windows
- **Text from bug tracker:** Fullscreen looks like screenshot.
Searching

- One of the main activities performed by developers while changing software
- Used during concept location
- IDEs support searching
- Traditional source code search: keyword and regular expression matching
- Modern source code search: Information Retrieval (IR)
IR-based concept location

Query

IR Search

Enter your query:

Ok  Cancel

Ranked list of results

<table>
<thead>
<tr>
<th>#</th>
<th>Method</th>
<th>Class</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>getFace</td>
<td>org.eclipse.ui.jface</td>
<td>0.99</td>
</tr>
<tr>
<td>2</td>
<td>nextEntry</td>
<td>org.eclipse.jdt.indexBlock</td>
<td>0.95</td>
</tr>
<tr>
<td>3</td>
<td>getSeparator</td>
<td>org.eclipse.jdt.core.util</td>
<td>0.95</td>
</tr>
<tr>
<td>4</td>
<td>validate</td>
<td>org.eclipse.jface.IDialog</td>
<td>0.87</td>
</tr>
<tr>
<td>5</td>
<td>setTextDig</td>
<td>org.eclipse.ui.text</td>
<td>0.86</td>
</tr>
</tbody>
</table>

IR Engine

Index
The searching paradox

• User has an information need because she does not know something
• Search systems are designed to satisfy these needs, but the user needs to know what she is looking for (query)
• If the user knows what she is looking for, there may not be a need to search in the first place
Challenge in search-based CL: the query

- Text in the query needs to match the text in the source code
- Difficult to formulate good queries
  - Unfamiliar source code
  - Unknown target
- Some developers write better queries than others
Eclipse bug #13926

Bug description:

JFace Text Editor Leaves a Black Rectangle on Content Assist text insertion. Inserting a selected completion proposal from the context information popup causes a black rectangle to appear on top of the display.
Queries

• **Q1**: jface text editor black rectangle insert text

• **Q2**: jface text editor rectangle insert context information

• **Q3**: jface text editor content assist

• **Q4**: jface insert selected completion proposal context information
Results

• Q1: jface text editor black rectangle insert text
  – Position of buggy method: 7496
• Q2: jface text editor rectangle insert context information
  – Position of buggy method: 258
• Q3: jface text editor content assist
  – Position of buggy method: 119
• Q4: jface insert selected completion proposal context information
  – Position of buggy method: 723

Q5: Whole change request: position 54

Ranked list of results

<table>
<thead>
<tr>
<th>#</th>
<th>Method</th>
<th>Class</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>getFace</td>
<td>org.eclipse.ui.JFace</td>
<td>0.99</td>
</tr>
<tr>
<td>2</td>
<td>nextEntry</td>
<td>org.eclipse.jdt.IndexBlock</td>
<td>0.96</td>
</tr>
<tr>
<td>3</td>
<td>getSeparator</td>
<td>org.eclipse.jdt.core.Util</td>
<td>0.95</td>
</tr>
<tr>
<td>4</td>
<td>validate</td>
<td>org.eclipse.jface.IDialog</td>
<td>0.87</td>
</tr>
<tr>
<td>5</td>
<td>setTextDlg</td>
<td>org.eclipse.ui.Text</td>
<td>0.86</td>
</tr>
</tbody>
</table>
IR-based CL in unfamiliar software

Developers:

• Rarely begin with a good query: hard to choose the right words
• Even after reformulation, vague idea of what to look for -> queries not always better
• Can recognize whether the results retrieved are relevant or not to the problem
Relevance Feedback (RF)

• **Idea:** you may not know how to articulate what you are looking for, but you will recognize it when you see it
• Helps users iteratively refine their query until they find what they are looking for
• Every consecutive query is meant to bring the search results closer to the target (“more like this”)
• Queries are reformulated automatically based on:
  – The initial results to the query
  – Information about whether or not these results are relevant to the goal, often provided by users (feedback)
Types of relevance feedback

• Explicit
  – Users specify which search results are relevant and which are not

• Implicit
  – User navigates
  – Clicked-on documents: relevant
  – Example: Google search (user logged in)

• Pseudo
  – The top $n$ search results are automatically considered relevant
IR + RF

• RF found to be one of the most powerful methods for improving IR performance (40-60% increase in precision)

• Often you just need feedback about a few documents to really make your results better

• First and most common approach: **Rocchio**
  
  – **Recognize** what is **relevant** for your goal and what is **irrelevant**
  – Add terms from relevant documents to query
  – Remove terms from irrelevant documents from query
  – Iterative process – rounds of feedback
  – Four parameters, representing the number of documents marked in a round of feedback, and the importance of: the initial query, relevant terms, irrelevant terms
java +coffee +flavor +buy –programming –island -sdk

java +island +christmas +buy +vacation -coffee –programming -sdk

java +programming +example +arraylist –island –coffee –sdk
http://www.pandora.com
IR + RF for concept location

- Developers can focus on source code rather than queries
- Developers writing poorer queries can get results as good as the ones writing good queries
- Process:
  - Corpus creation
  - Indexing
  - Query formulation
  - Ranking documents
  - Results examination
  - Feedback
Corpus creation

• First stage in using an IR technique
• Source code considered as a text corpus: identifiers and comments
• The source code is parsed using a developer-defined granularity level (i.e., methods or classes)
• Each method (or class) is extracted from the source code and has a corresponding document in the corpus
• Eliminating common terms from corpus
• Natural language processing (NLP) techniques can be applied to the corpus (e.g., stemming)
Indexing

• The corpus is indexed using the IR technique
• A mathematical representation of the corpus is created - Vector Space Model (VSM)
• Each document (i.e., each method or class) has a corresponding vector in the VSM
Query formulation

• A developer selects a set of words from the change request which constitutes the initial query
• If filtering or NLP was used in the corpus creation, the query will get the same treatment
• Simpler approach: consider the entire change request as the initial query
• Query is also represented in VSM as a vector
Ranking documents

• Similarities between the query and every document from the source code are computed
• The documents in the corpus are ranked according to the similarity to the query
• The similarity measure depends on the IR method used
  – **Cosine similarity**: angle between the vector representations of the query and the document
Results examination

• Iterative process
• Steps:
  1. The developer examines the top N documents in the ranked list of results
  2. For every document examined, the developer makes a decision on whether the document will be changed
  3. If it will be changed, the search succeeded and concept location ends.
  4. Else, the developer marks the document as relevant, irrelevant. If a decision cannot be made, examine the next document in the list
  5. After the N documents are marked a new query is automatically formulated
Change Request = Initial Query

JFace Text Editor Leaves a Black Rectangle on Content Assist text insertion. Inserting a selected completion proposal from the context information popup causes a black rectangle to appear on top of the display.

1. `createContextInfoPopup()` in `org.eclipse.jface.text.contentassist.ContextInformationPopup` ✔️
2. `configure()` in `org.eclipse.jdt.internal.debug.ui.JDIContentAssistPreference` ✗
3. `showContextProposals()` in `org.eclipse.jface.text.contentassist.ContextInformationPopup` ✔️

+ words in ✔️ documents
- words in ✗ documents

New Query
IRRF tool

• IR Engine: *Lucene*
  – Based on the Vector Space Model (VSM)
  – Input: *methods*, *query*
  – Output: a ranked list of methods ordered by their textual similarity to the query

• Relevance feedback: *Rocchio algorithm*
  – Models a way of incorporating relevance feedback information into the VSM
  – Relevance tags: “relevant”, “irrelevant”, “don’t know”
First study (ICSM ‘09)

• **Goal:** study if the use of RF improves IR

• One developer providing RF
  – 7 years of programming experience (5 years in Java)
  – Not familiar with the systems used

• Concept location for 16 bug reports from 3 systems
  – Extracted the text of the bug reports and the list of changed methods from bug tracking systems
  – The modified methods are called *target methods*
  – Task: locate one of the target methods
Methodology

• Consider bug textual description as initial query
• Perform concept location:
  – Using only IR
  – Using IRRF ($N = 1, 3, 5$ documents in a round of feedback)
• **Metric**: #methods analyzed until finding a target method
• IRRF: iterative – stop when:
  – Target method in top N results - **Success**
  – More than 50 methods analyzed - **Failure**
  – Position of target methods in the ranked list of results increases for 2 consecutive rounds (query moving away from target) - **Failure**
# Systems

<table>
<thead>
<tr>
<th>System</th>
<th>Vers.</th>
<th>LOC</th>
<th>Methods</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eclipse</td>
<td>2.0</td>
<td>2,500,000</td>
<td>74,996</td>
<td>7,500</td>
</tr>
<tr>
<td>jEdit</td>
<td>4.2</td>
<td>300,000</td>
<td>5,366</td>
<td>750</td>
</tr>
<tr>
<td>Adempiere</td>
<td>3.1.0</td>
<td>330,000</td>
<td>28,622</td>
<td>1,900</td>
</tr>
</tbody>
</table>
## Results

<table>
<thead>
<tr>
<th>System</th>
<th>RF improves IR</th>
<th>RF does not improve IR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eclipse</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>jEdit</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Adempiere</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>13</strong></td>
<td><strong>5</strong></td>
</tr>
</tbody>
</table>
Conclusions of the first study

- RF is a promising technique for easing query formulation during concept location

- RF improves IR-based concept location in most cases

- In some cases RF does not improve IR (user mislead by the poor description of the bug)
Some limitations of the study

• RF depends on a series of parameters
  – we chose the values based on literature and preliminary tests
  – we used the same values for all systems

• Only one developer provided feedback

• CL ended automatically when a target method was retrieved in the top n results
Second study - goals

• Address the limitations of the first study
• Parameters:
   – Design a methodology to automatically find the most suited parameters for a particular software system
   – Calibrate the parameters separately for each system
• Number of developers:
   – Collect data from a larger number of developers
• Realistic concept location process (CL):
   – Does not end automatically when target method is in top N results
   – Developers can change their mind about the relevance of a method
Calibration of parameters

• Implemented a tool that automatically finds the RF parameters
  – Number of documents in a round of feedback (N)
  – Weight of terms in the initial query (a)
  – Weight of terms from relevant documents (b)
  – Weight of terms from irrelevant documents (c)

• Idea: mimic CL on a software system using a set of bug reports and their target methods

• 4 change requests per system, 4 software systems
Calibration of parameters (contd)

• For each change request - 500 configurations:
  – N = 1, 2, 3, 4, 5
  – a = 0.25, 0.5, 0.75, 1
  – b = 0, 0.25, 0.5, 0.75, 1
  – c = 0, 0.25, 0.5, 0.75, 1

• For each configuration {N, a, b, c}:
  – Used ASTAR algorithm to simulate an ideal user: find the optimum set of relevance tags for the methods (relevant or irrelevant) so that the target methods are retrieved the fastest
  – Recorded the number of methods analyzed to find a target method

• Chose the parameter combination that lead to the best results for all 4 change requests for a system

• Findings: each system has a different configuration of parameters
Study setup

- Participants: 18 students in a Data Mining class @ West Virginia University
- Java quiz
- Systems and change requests:
  - 4 Java systems, different domains and sizes
  - 4 bug fix requests for each system (different than the ones used for calibration)
- IRRF Tool: integrates the parameters determined in the calibration phase
Change in methodology

• Relevance marks: “relevant”, “irrelevant”, “don’t know”, “target”

• Concept location stops when:
  – The user marks a method as “target”
  – The user analyzed more than 50 methods

• Users can change their mind about the relevance of a method at any time query reformulated accordingly
Status

• Data collection finalized

• Data analysis in progress
Conclusions

• RF is a good addition to IR for concept location
  – Easier query formulation
  – Better results
• RF parameters need to be adjusted automatically to each software system
• More studies needed to understand better the situations when RF does not improve IR
Future work

• Relevance feedback in IDEs, to help developers search and navigate source code
• RF for other software engineering tasks
• Other RF algorithms
• Other types of relevance feedback: implicit, pseudo
• Study how developers decide if a document is relevant or not to their task