# The Use of Text Retrieval and Natural Language Processing in Software Engineering

Venera Arnaoudova
Andrian Marcus

UT DALLAS

Sonia Haiduc

FLORIDA STATE UNIVERSITY 1851

Giuliano Antoniol

ÉCOLE POLYTECHNIQUE MONTRÉAL

# TB outline

- Introduction
- Background on IR and NLP
- SE tasks using IR and NLP
  - Task definition
  - Input
  - Output
  - Preprocessing
  - Technique
  - Evaluation
  - Tools used
- Conclusion and future directions

# Textual Information in Software

- Captures concepts of the problem domain, developer intentions, developer communication, etc.
- Found in many software artifacts:
  - Requirements
  - Design documents
  - Source code (identifiers, comments)
  - Commit notes
  - Documentation
  - User manuals
  - Q/A websites
  - Developer communication: emails, chat, tweets
  - Etc.

# Text Retrieval

- *Information Retrieval (IR)*: the process of actively seeking out information relevant to a topic of interest (van Rijsbergen)

- **Text Retrieval (TR):** a branch of IR where the information is stored in text format
  - Typically it refers to the automatic retrieval of documents
  - *Document* - generic term for an information holder (book, chapter, article, webpage, class body, method, requirement page, etc.)

# Natural Language Processing

- Refers to the use and ability of systems to process sentences in a natural language such as English (rather than in a specialized artificial computer language such as C++)

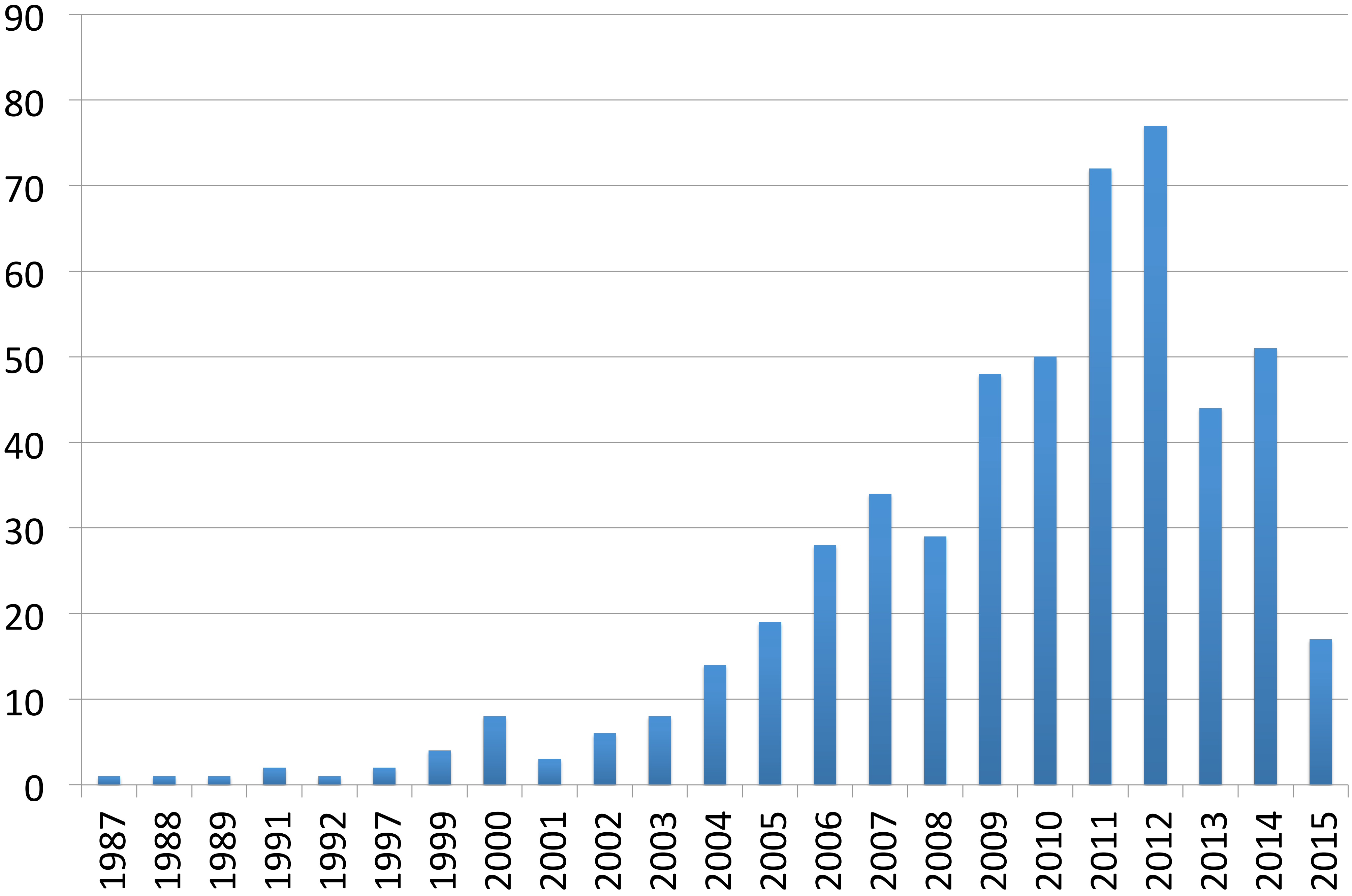
- Combines techniques from of computer science, artificial intelligence, and computational linguistics, probability and statistics

# TR and NLP in Software Engineering

- Applied to over 30 different SE tasks

- Traceability Link Recovery
- Feature/concept/concern/bug location
- Code reuse
- Bug triage
- Program comprehension
- Architecture/design recovery
- Quality assessment and measurement
- Software evolution analysis
- Defect prediction and debugging
- Automatic documentation
- Testing
- Requirements analysis
- Restructuring/refactoring
- Software categorization
- Licensing analysis
- Impact analysis
- Clone detection
- Effort prediction/estimation
- Domain analysis
- Web services discovery
- Use case analysis
- Team management, etc.

# Publications per year



# What is Text Retrieval?

- Basis for internet search engines
- Search space is a collection of documents ("bags of words")
- Search engine creates a cache consisting of indexes of each document – different techniques create different indexes
- No predefined grammar and vocabulary
- Many TR models are not intuitive for humans -> will not understand well the results of TR approaches

# Terminology

- Document = unit of information – bag of words
- Corpus = collection of documents
- Term vs. word – basic unit of text - not all terms are words
- Query
- Index
- Rank
- Relevance

# Document Granularity

- What is a *document* in source code?
  - Depends on the problem and programming language
  - Class, method, function, interface, procedure, etc.

- What is a *document* in other artifacts?
  - Depends on the artifact and problem
  - Individual requirements, bug descriptions, test cases, e-mails, design diagrams, etc.

# Most Popular Models Used in SE

- Vector Space Model (VSM)
- Latent Semantic Indexing (LSI)
- Okapi BM25 and BM25F
- Latent Dirichlet Allocation (LDA)
- Probabilistic LSI (pLSI)

# Term Weights and Document Similarities in VSM

- Term weight = Local weight * Global weight

Local weights:

- binary
- tf
- log (tf)

Global weights:

- binary
- idf
- entropy

- Most common weight: tf-idf

- Doc Similarities: Cosine, Dice, Jaccard

# A Typical TR Application

1. Build corpus
2. Index corpus – choose the IR model
3. Formulate a query (Q)
   - Manual or automatic
4. Compute similarities between Q and the documents in the corpus (i.e., relevance)
5. Rank the documents based on the similarities
6. Return the top N as the result list
7. Inspect the results
8. GO TO 3. if needed or STOP

# Using TR in SE – Option 1

- Formulate the SE problem as a text retrieval problem

- Convert the software artifacts into a text corpus

- Choose the TR model best suited to the problem

# SE as TR

- Concept/concern/feature location in software
- Traceability link recovery between software artifacts
- Impact analysis
- Software reuse
- Bug triage
- Requirements analysis
- Etc.

# Using TR in SE – Option 2

1. Analysis of the textual information in software
2. Convert the software artifacts into a text corpus
3. Choose the TR model best suited to the problem
4. Compute similarities between documents
5. Perform analysis based on these measures

# SE as Text Analysis

- Software categorization
- Refactoring and restructuring
- Reverse engineering
- Bug triage
- Clone detection
- Requirements analysis
- Defect prediction
- Change impact analysis
- Etc.

# Natural Language Processing (NLP)

- Text is not only a bag of words..

{‘a’, ’chasing’, ‘cat’, ’ fish’, ’is’, ’the’}

“a **cat** is chasing the **fish**” ≠ “a **fish** is chasing the **cat**”

# NLP Techniques

- Language Models (LM)
- Syntactic analysis
- Semantic analysis
- Sentiment analysis
- Emotion analysis

# Language Models (LM)

- Assign probabilities for sequences of words

  Corpus: “I am smiling”, “You are happy”, “We are happy”
  “I am” happy? -> $P(\text{happy} \mid \text{I am})$?

| | |
|---|---|
| uni-gram: ~ $P(\text{happy})$ | …. happy |
| bi-gram: ~ $P(\text{happy} \mid \text{am})$ | …. am happy |
| tri-gram: ~ $P(\text{happy} \mid \text{I am})$ | …. I am happy |

…
n-gram

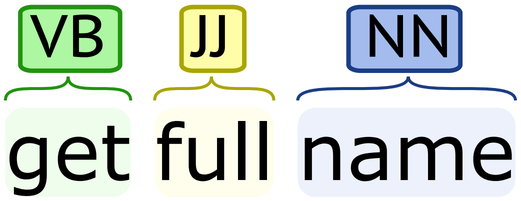
# Syntactic Analysis

- Tagging words with their respective Part-Of-Speech (POS)

**VB**: verb

**JJ**: adjective

**NN**: noun



VB JJ NN

get full name

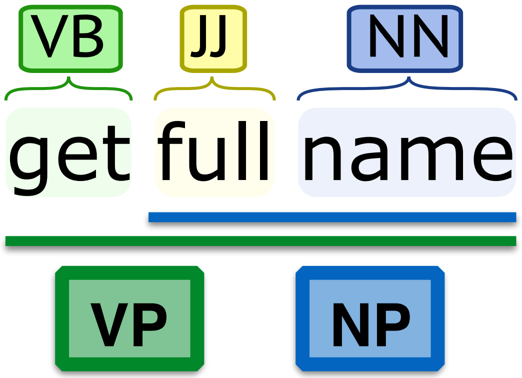
# Syntactic Analysis

- Tagging words with their respective Part-Of-Speech (POS)

- Chunking

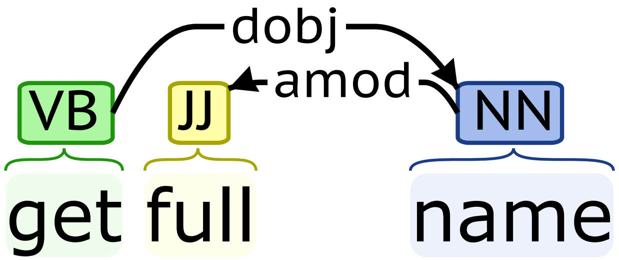
**VB**: verb
**JJ**: adjective
**NN**: noun

**NP**: noun phrase
**VP**: verb phrase



# Syntactic Analysis

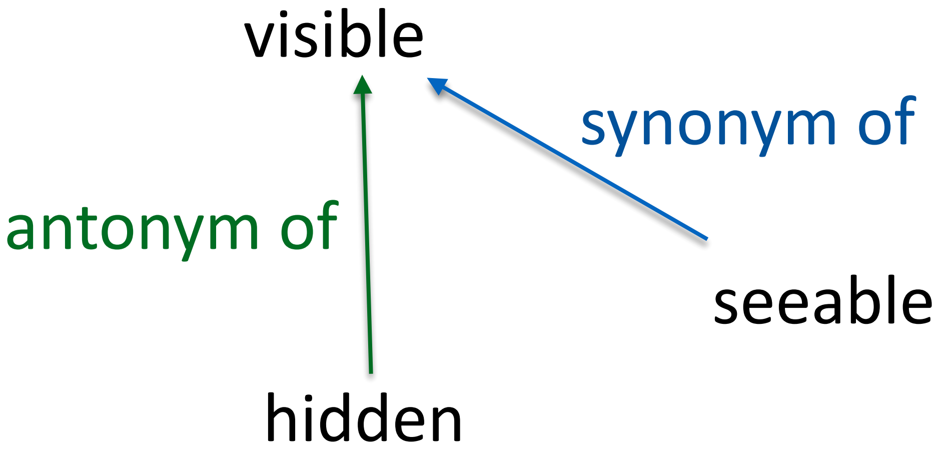
- Identifying grammatical relations between words



**amod**: adjectival modifier
**dobj**: direct object

# Semantic Analysis

- Identifying semantic relations between words



visible

antonym of

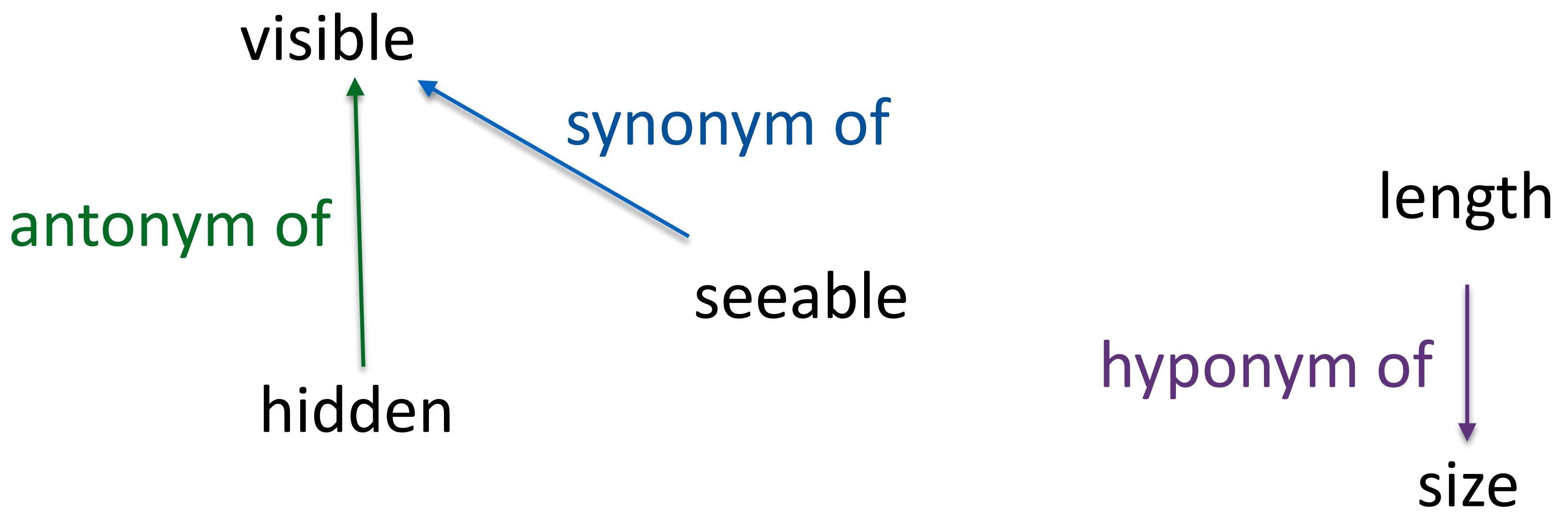
synonym of

seeable

hidden

# Semantic Analysis

- Identifying semantic relations between words



visible

synonym of

antonym of

seeable

hidden

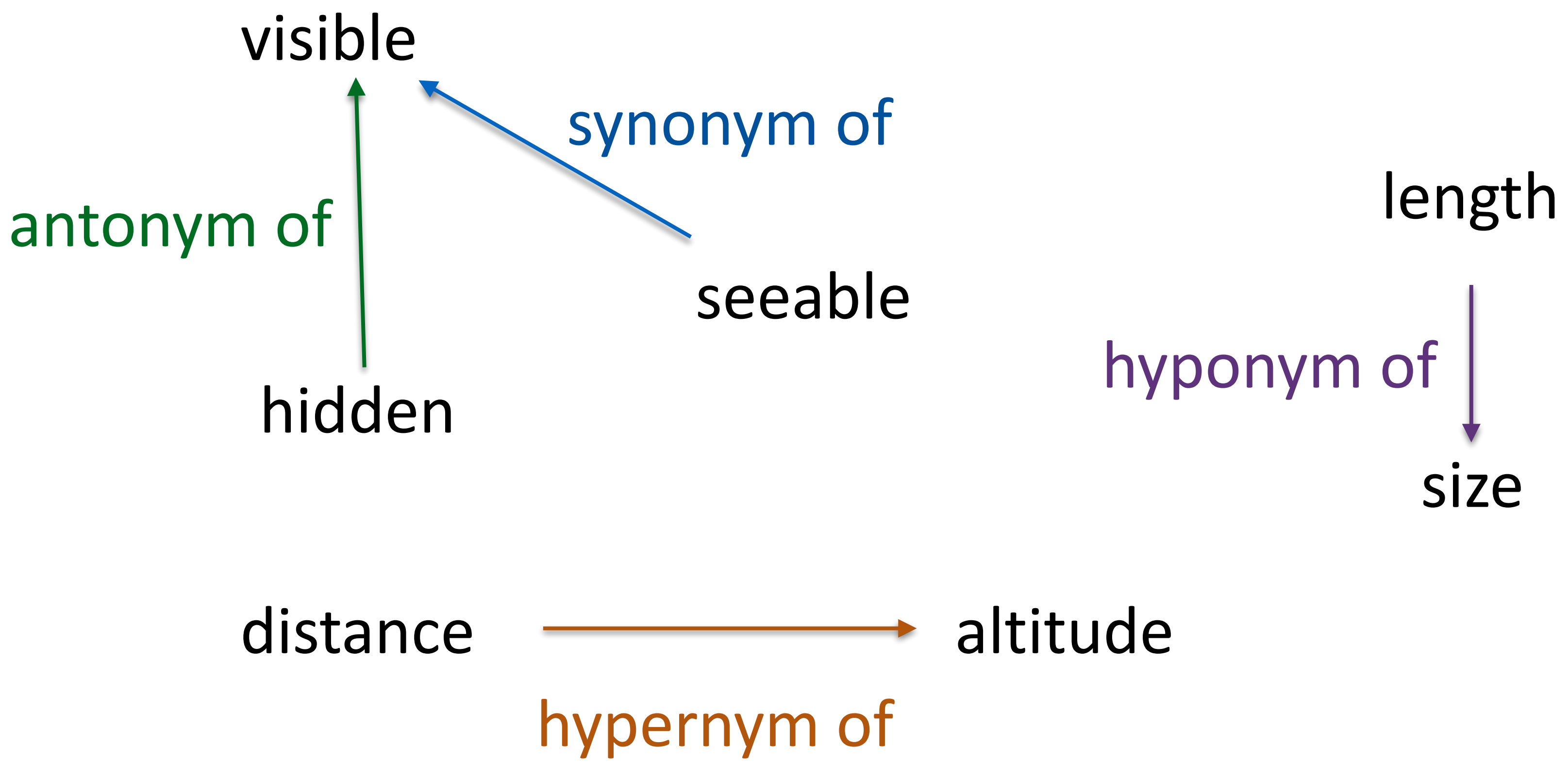
length

hyponym of

size

# Semantic Analysis

- Identifying semantic relations between words



visible

antonym of

hidden

synonym of

seeable

length

hyponym of

size

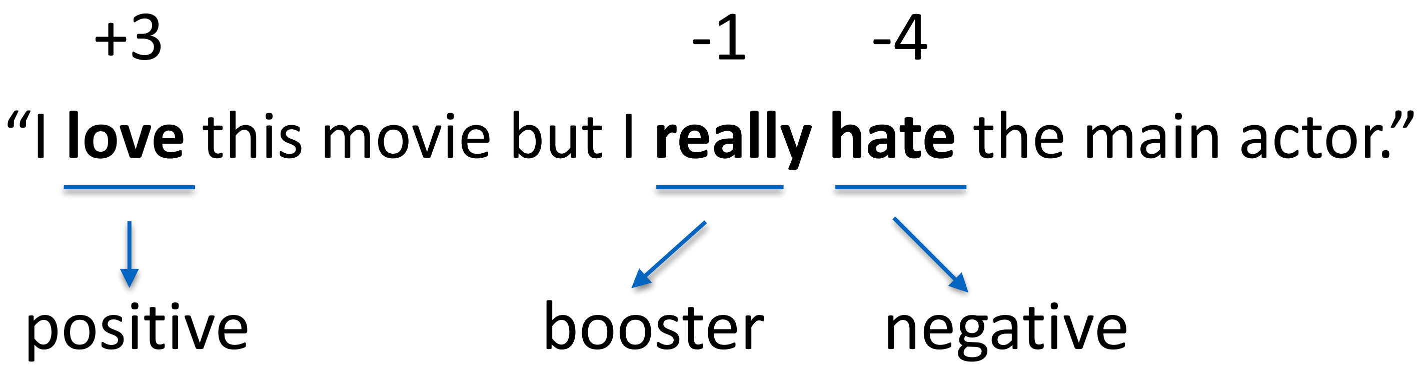
distance

hypernym of

altitude

# Sentiment Analysis

- Classify the polarity of a text



+3 -1 -4

"I **love** this movie but I **really** **hate** the main actor."

positive booster negative

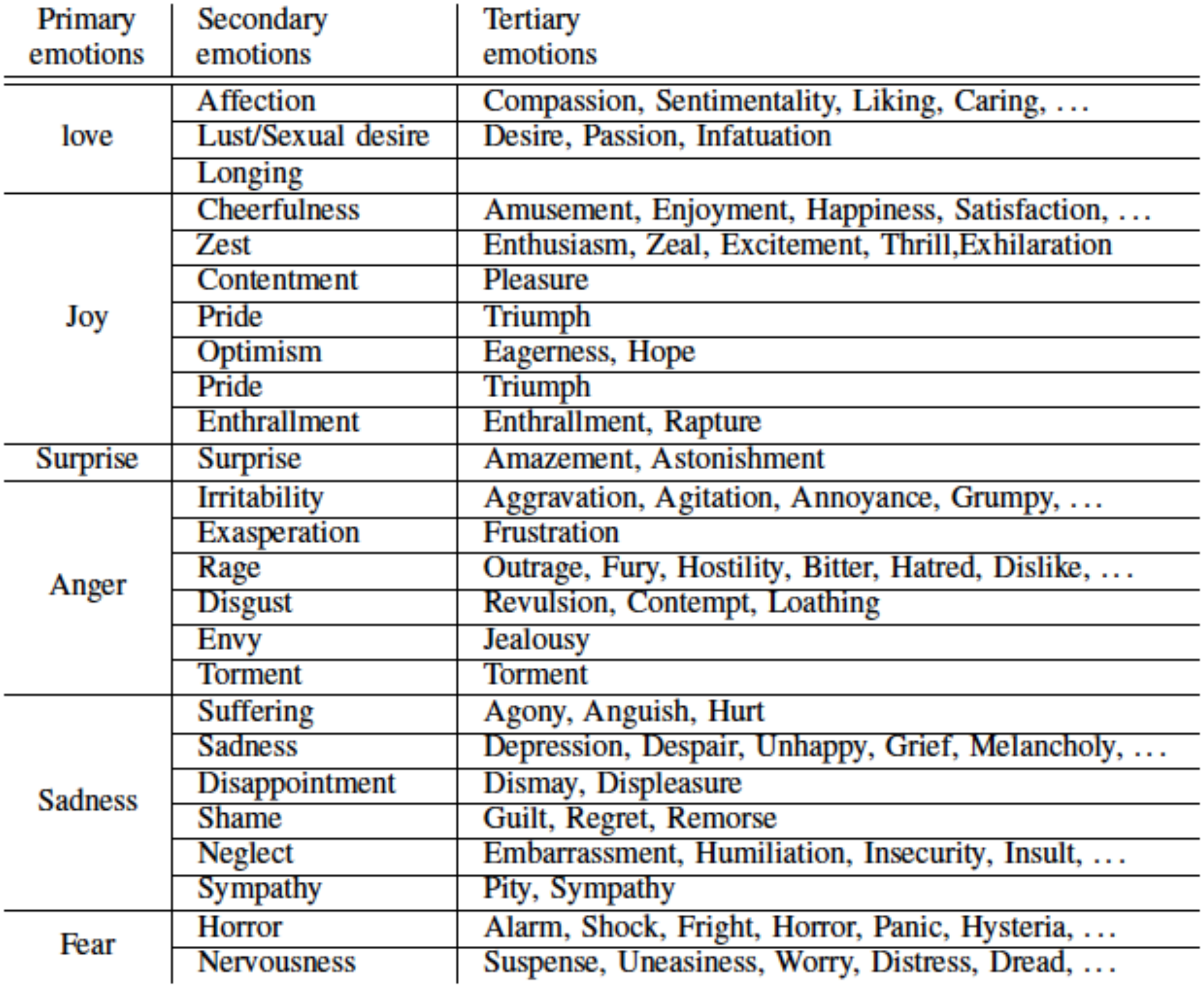
Positive sentiment strength: 3

Negative sentiment strength: -5

# Emotion Analysis

- Joy: “That’s great work guys!”
- Anger: “ I will come over to your work and slap you!”
- Sadness: “Sorry for the late response.”
- …

# Parrott's Framework



| Primary emotions | Secondary emotions | Tertiary emotions |
|---|---|---|
| love | Affection | Compassion, Sentimentality, Liking, Caring, . . . |
| | Lust/Sexual desire | Desire, Passion, Infatuation |
| | Longing | |
| Joy | Cheerfulness | Amusement, Enjoyment, Happiness, Satisfaction, . . . |
| | Zest | Enthusiasm, Zeal, Excitement, Thrill,Exhilaration |
| | Contentment | Pleasure |
| | Pride | Triumph |
| | Optimism | Eagerness, Hope |
| | Pride | Triumph |
| | Enthrallment | Enthrallment, Rapture |
| Surprise | Surprise | Amazement, Astonishment |
| Anger | Irritability | Aggravation, Agitation, Annoyance, Grumpy, . . . |
| | Exasperation | Frustration |
| | Rage | Outrage, Fury, Hostility, Bitter, Hatred, Dislike, . . . |
| | Disgust | Revulsion, Contempt, Loathing |
| | Envy | Jealousy |
| | Torment | Torment |
| Sadness | Suffering | Agony, Anguish, Hurt |
| | Sadness | Depression, Despair, Unhappy, Grief, Melancholy, . . . |
| | Disappointment | Dismay, Displeasure |
| | Shame | Guilt, Regret, Remorse |
| | Neglect | Embarrassment, Humiliation, Insecurity, Insult, . . . |
| | Sympathy | Pity, Sympathy |
| Fear | Horror | Alarm, Shock, Fright, Horror, Panic, Hysteria, . . . |
| | Nervousness | Suspense, Uneasiness, Worry, Distress, Dread, . . . |

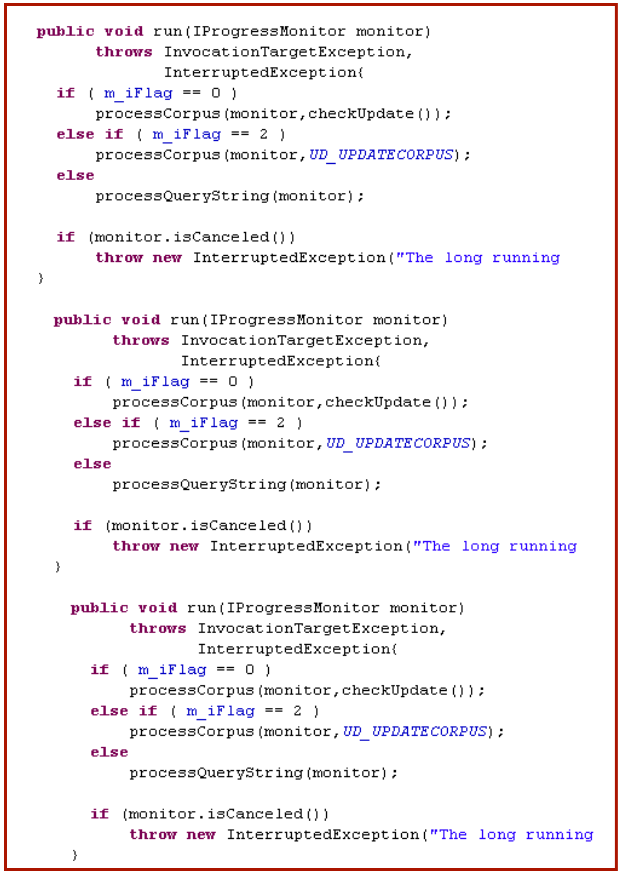
# Creating a Corpus of a Software System

- Parsing software artifacts and extracting documents
  - ***corpus*** – collection of documents (e.g., methods)
- Text normalization (white space and non-textual tokens removal, tokenization)
- Splitting: split_identifiers and SplitIdentifiers
- Stop words removal
  - common words in English, standard function library names, programming language keywords
- Stemming

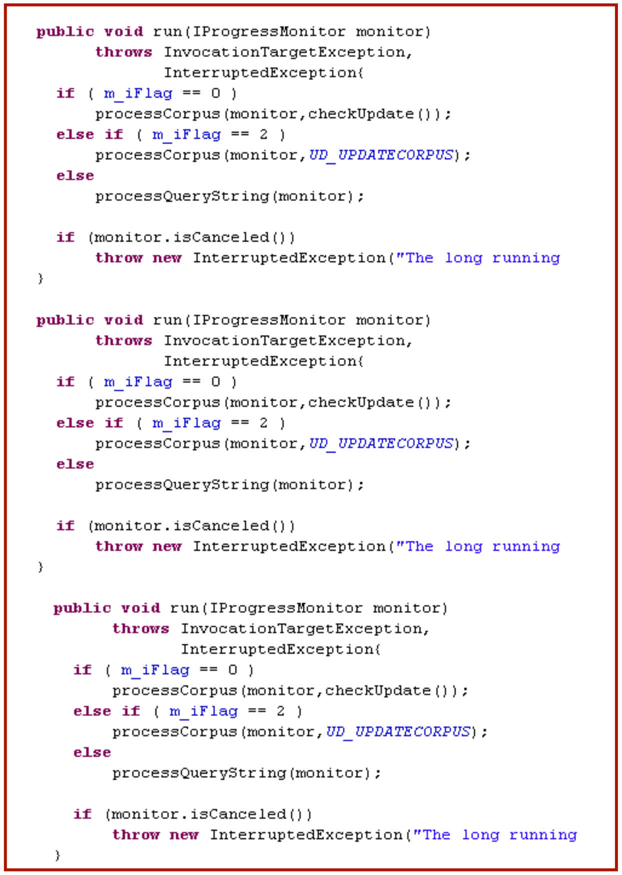
-> **Software Lexicon**

# Parsing Source Code and Extracting Documents

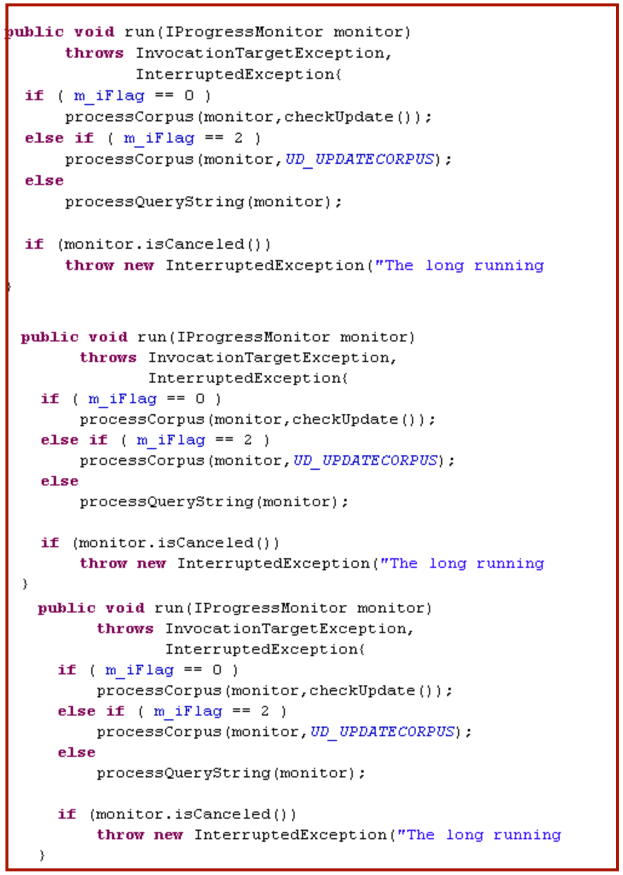
- Documents can be at different granularities (e.g., methods, classes, files)



```
public void run(IProgressMonitor monitor)
      throws InvocationTargetException,
             InterruptedException{
  if ( m_iFlag == 0 )
      processCorpus(monitor,checkUpdate());
  else if ( m_iFlag == 2 )
      processCorpus(monitor,UD_UPDATECORPUS);
  else
      processQueryString(monitor);

  if (monitor.isCanceled())
      throw new InterruptedException("The long running
}

  public void run(IProgressMonitor monitor)
        throws InvocationTargetException,
               InterruptedException{
    if ( m_iFlag == 0 )
        processCorpus(monitor,checkUpdate());
    else if ( m_iFlag == 2 )
        processCorpus(monitor,UD_UPDATECORPUS);
    else
        processQueryString(monitor);

    if (monitor.isCanceled())
        throw new InterruptedException("The long running
  }

    public void run(IProgressMonitor monitor)
          throws InvocationTargetException,
                 InterruptedException{
      if ( m_iFlag == 0 )
          processCorpus(monitor,checkUpdate());
      else if ( m_iFlag == 2 )
          processCorpus(monitor,UD_UPDATECORPUS);
      else
          processQueryString(monitor);

      if (monitor.isCanceled())
          throw new InterruptedException("The long running
    }
```



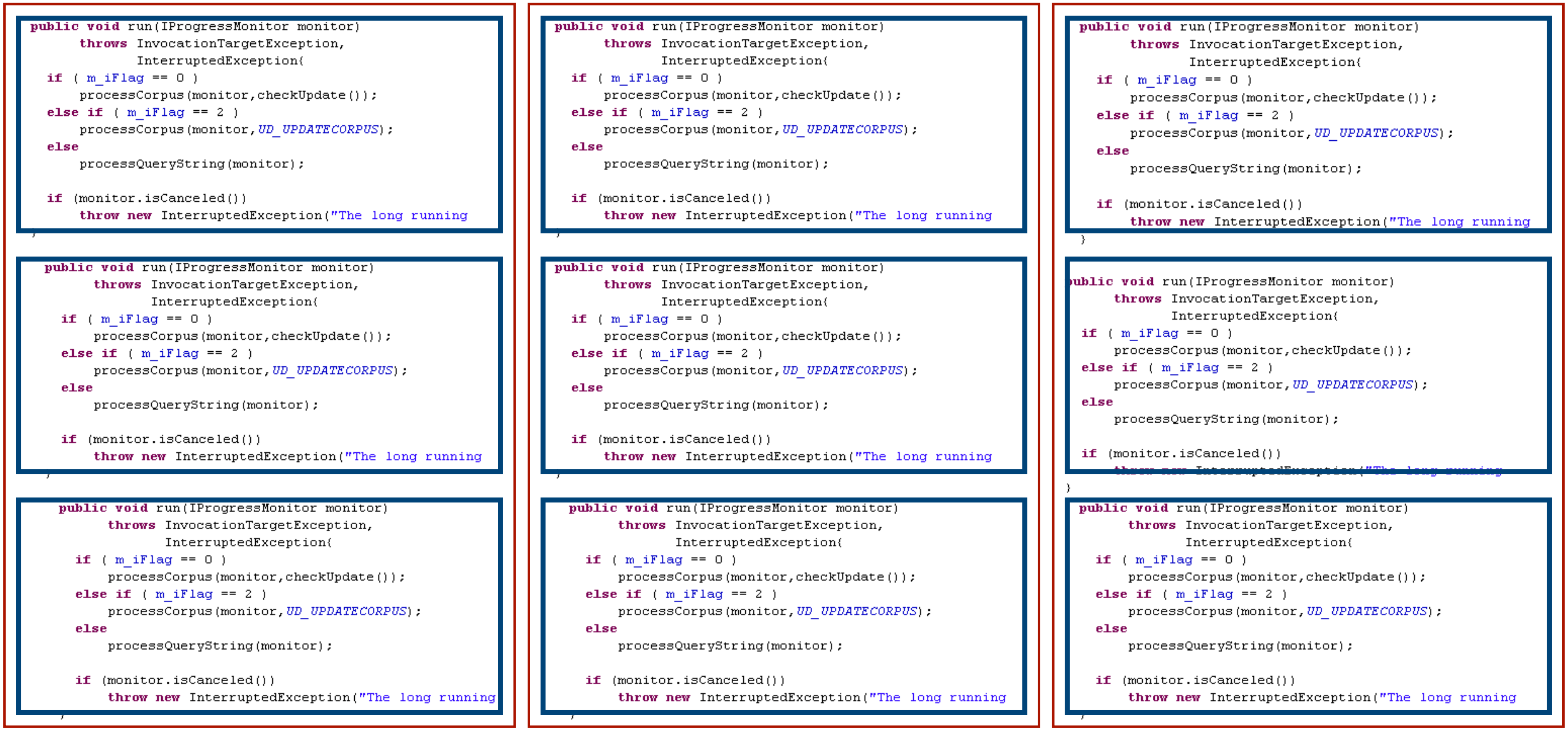
```
public void run(IProgressMonitor monitor)
      throws InvocationTargetException,
             InterruptedException{
  if ( m_iFlag == 0 )
      processCorpus(monitor,checkUpdate());
  else if ( m_iFlag == 2 )
      processCorpus(monitor,UD_UPDATECORPUS);
  else
      processQueryString(monitor);

  if (monitor.isCanceled())
      throw new InterruptedException("The long running
}

public void run(IProgressMonitor monitor)
      throws InvocationTargetException,
             InterruptedException{
  if ( m_iFlag == 0 )
      processCorpus(monitor,checkUpdate());
  else if ( m_iFlag == 2 )
      processCorpus(monitor,UD_UPDATECORPUS);
  else
      processQueryString(monitor);

  if (monitor.isCanceled())
      throw new InterruptedException("The long running
}

  public void run(IProgressMonitor monitor)
        throws InvocationTargetException,
               InterruptedException{
    if ( m_iFlag == 0 )
        processCorpus(monitor,checkUpdate());
    else if ( m_iFlag == 2 )
        processCorpus(monitor,UD_UPDATECORPUS);
    else
        processQueryString(monitor);

    if (monitor.isCanceled())
        throw new InterruptedException("The long running
  }
```



```
public void run(IProgressMonitor monitor)
      throws InvocationTargetException,
             InterruptedException{
  if ( m_iFlag == 0 )
      processCorpus(monitor,checkUpdate());
  else if ( m_iFlag == 2 )
      processCorpus(monitor,UD_UPDATECORPUS);
  else
      processQueryString(monitor);

  if (monitor.isCanceled())
      throw new InterruptedException("The long running
}

  public void run(IProgressMonitor monitor)
        throws InvocationTargetException,
               InterruptedException{
    if ( m_iFlag == 0 )
        processCorpus(monitor,checkUpdate());
    else if ( m_iFlag == 2 )
        processCorpus(monitor,UD_UPDATECORPUS);
    else
        processQueryString(monitor);

    if (monitor.isCanceled())
        throw new InterruptedException("The long running
  }
    public void run(IProgressMonitor monitor)
          throws InvocationTargetException,
                 InterruptedException{
      if ( m_iFlag == 0 )
          processCorpus(monitor,checkUpdate());
      else if ( m_iFlag == 2 )
          processCorpus(monitor,UD_UPDATECORPUS);
      else
          processQueryString(monitor);

      if (monitor.isCanceled())
          throw new InterruptedException("The long running
    }
```

# Parsing Source Code and Extracting Documents

- Documents can be at different granularities (e.g., methods, classes, files)



```
public void run(IProgressMonitor monitor)
      throws InvocationTargetException,
             InterruptedException{
   if ( m_iFlag == 0 )
       processCorpus(monitor,checkUpdate());
   else if ( m_iFlag == 2 )
       processCorpus(monitor,UD_UPDATECORPUS);
   else
       processQueryString(monitor);

   if (monitor.isCanceled())
       throw new InterruptedException("The long running
}
```

```
public void run(IProgressMonitor monitor)
      throws InvocationTargetException,
             InterruptedException{
   if ( m_iFlag == 0 )
       processCorpus(monitor,checkUpdate());
   else if ( m_iFlag == 2 )
       processCorpus(monitor,UD_UPDATECORPUS);
   else
       processQueryString(monitor);

   if (monitor.isCanceled())
       throw new InterruptedException("The long running
}
```

```
public void run(IProgressMonitor monitor)
      throws InvocationTargetException,
             InterruptedException{
   if ( m_iFlag == 0 )
       processCorpus(monitor,checkUpdate());
   else if ( m_iFlag == 2 )
       processCorpus(monitor,UD_UPDATECORPUS);
   else
       processQueryString(monitor);

   if (monitor.isCanceled())
       throw new InterruptedException("The long running
}
```

```
public void run(IProgressMonitor monitor)
      throws InvocationTargetException,
             InterruptedException{
   if ( m_iFlag == 0 )
       processCorpus(monitor,checkUpdate());
   else if ( m_iFlag == 2 )
       processCorpus(monitor,UD_UPDATECORPUS);
   else
       processQueryString(monitor);

   if (monitor.isCanceled())
       throw new InterruptedException("The long running
}
```

```
public void run(IProgressMonitor monitor)
      throws InvocationTargetException,
             InterruptedException{
   if ( m_iFlag == 0 )
       processCorpus(monitor,checkUpdate());
   else if ( m_iFlag == 2 )
       processCorpus(monitor,UD_UPDATECORPUS);
   else
       processQueryString(monitor);

   if (monitor.isCanceled())
       throw new InterruptedException("The long running
}
```

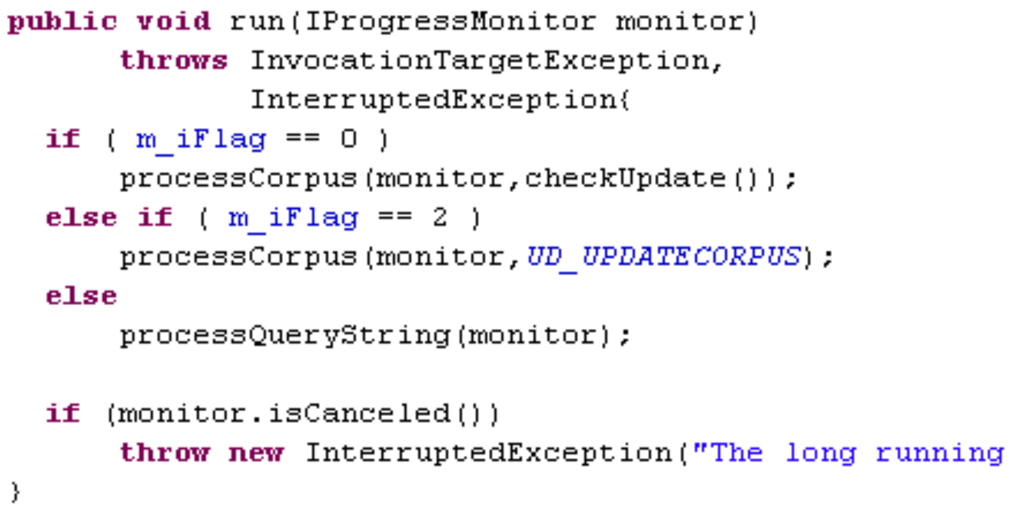
```
public void run(IProgressMonitor monitor)
      throws InvocationTargetException,
             InterruptedException{
   if ( m_iFlag == 0 )
       processCorpus(monitor,checkUpdate());
   else if ( m_iFlag == 2 )
       processCorpus(monitor,UD_UPDATECORPUS);
   else
       processQueryString(monitor);

   if (monitor.isCanceled())
       throw new InterruptedException("The long running
}
```

```
public void run(IProgressMonitor monitor)
      throws InvocationTargetException,
             InterruptedException{
   if ( m_iFlag == 0 )
       processCorpus(monitor,checkUpdate());
   else if ( m_iFlag == 2 )
       processCorpus(monitor,UD_UPDATECORPUS);
   else
       processQueryString(monitor);

   if (monitor.isCanceled())
       throw new InterruptedException("The long running
}
```

```
ublic void run(IProgressMonitor monitor)
      throws InvocationTargetException,
             InterruptedException{
   if ( m_iFlag == 0 )
       processCorpus(monitor,checkUpdate());
   else if ( m_iFlag == 2 )
       processCorpus(monitor,UD_UPDATECORPUS);
   else
       processQueryString(monitor);

   if (monitor.isCanceled())
       throw new InterruptedException("The long running
}
```

```
public void run(IProgressMonitor monitor)
      throws InvocationTargetException,
             InterruptedException{
   if ( m_iFlag == 0 )
       processCorpus(monitor,checkUpdate());
   else if ( m_iFlag == 2 )
       processCorpus(monitor,UD_UPDATECORPUS);
   else
       processQueryString(monitor);

   if (monitor.isCanceled())
       throw new InterruptedException("The long running
}
```

# Source Code is Text Too



```
public void run(IProgressMonitor monitor)
      throws InvocationTargetException,
             InterruptedException{
  if ( m_iFlag == 0 )
      processCorpus(monitor,checkUpdate());
  else if ( m_iFlag == 2 )
      processCorpus(monitor,UD_UPDATECORPUS);
  else
      processQueryString(monitor);

  if (monitor.isCanceled())
      throw new InterruptedException("The long running
}
```

public void run IProgressMonitor monitor throws
InvocationTargetException InterruptedException if m_iFlag
processCorpus monitor checkUpdate else if m_iFlag
processCorpus monitor UD_UPDATECORPUS else
processQueryString monitor if monitor isCancelled throw
new InterruptedException the long running

# Text Normalization

- Break up the text in words or “tokens”
- Question: “what is a word” ?

- Problem cases
  - Numbers: “M16”, “2001”
  - Hyphenation: “MS-DOS”, “OS/2”
  - Punctuation: “John’s”, “command.com”
  - Case: “us”, “US”
  - Phrases: “venetian blind”

# Splitting

- Splitting: decomposing identifiers into their compound words
- Identifiers may use of division markers (e.g., camelCase and _)
- Examples:
  - `getName` -> ‘get’, ‘Name’
  - `getMAXstring` -> ‘get’, ‘MAX’, ‘string’
  - `ASTNode` -> ‘AST’, ‘Node’
  - `account_number` -> ‘account’, ‘number’
  - `simpletypename` -> ‘simple’, ‘type’, ‘name’

# Stop Words

- Very frequent words, with no power of discrimination (e.g., language keywords)

- Typically function words, not indicative of content

- The stop words set depends on the document collection and on the application (e.g., language keywords)

# Stemming

- Identify morphological variants, creating “classes”
  - system, systems
  - forget, forgetting, forgetful
  - analyse, analysis, analytical, analysing

- Replace each term by the class representative (root or most common variant)

# Abbreviations expansion

- Expand abbreviations to the corresponding full word
- Single versus multi-word abbreviations
- Examples:
  - `mess` -> ‘message’
  - `src` -> ‘source’
  - `regex` -> ‘regular expression’
  - `ASCII` -> ‘American Standard Code for Information Interchange’
  - `auth` -> ‘authenticate’ OR ‘author’

TASK

# Improving the Quality of the Code Lexicon

✓ Identifying poor quality identifiers

✓ Identifying naming inconsistencies

SUBTASK

# Identifying Poor Quality Identifiers

- Task: Identifying identifiers that are difficult to understand, unclear, meaningless, etc.
- Examples:
  - aSz
  - foo
  - Variables `path` and `absolutePath`
  - Variables `file` of type `File` and `String`

INPUT

# Identifying Poor Quality Identifiers

- Source code
- Mapping between program identifiers and domain concepts
- Standard lexicon dictionary (a dictionary of allowed terms)
- Synonym/abbreviation dictionary

OUTPUT

# Identifying Poor Quality Identifiers

- Identifiers with poor quality
- Suggestions to improve the identifiers

# Identifying Poor Quality Identifiers

- Splitting

TECHNIQUE

# Identifying Poor Quality Identifiers

- Non-standard lexicon based on concepts

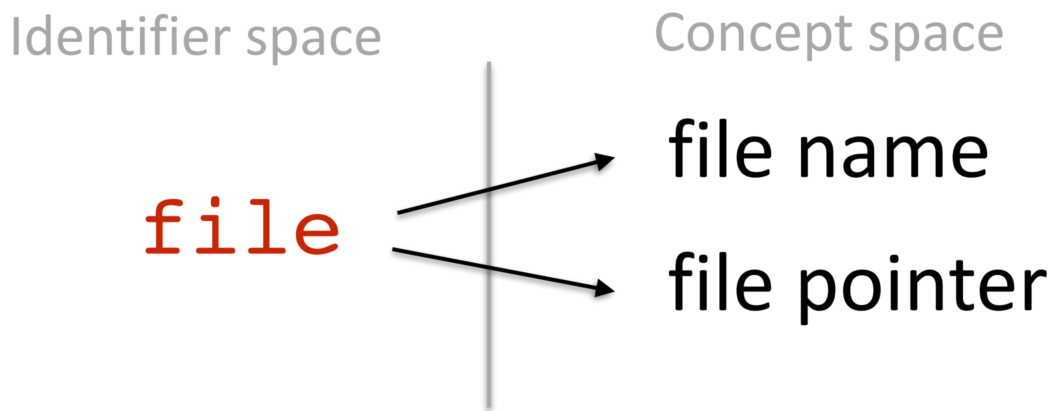
| | |
|---|---|
| meaningless: | foo |
| synonyms: | a**Copy** and print**Replica** |
| abbreviations: | aSz // a: array, Sz: size |

TECHNIQUE

# Identifying Poor Quality Identifiers

- Inconsistencies based on the concepts (cont.)

Homonym:



Identifier space

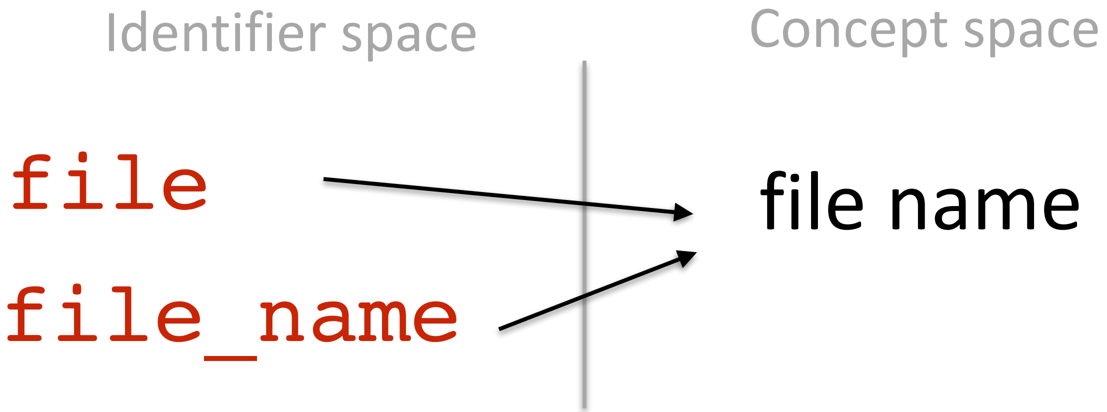
Concept space

`file`

file name

file pointer

Synonym:



Identifier space

Concept space

`file`

`file_name`

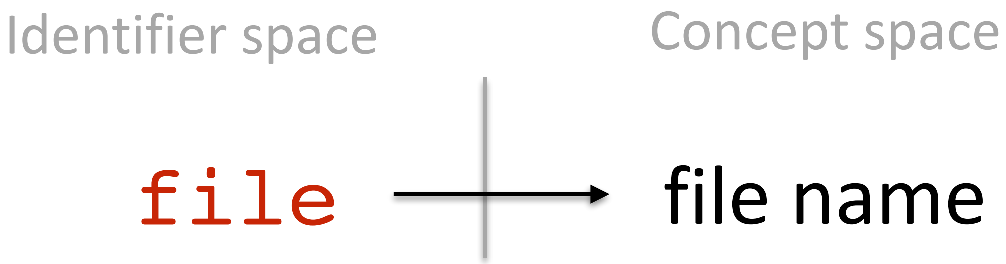
file name

TECHNIQUE

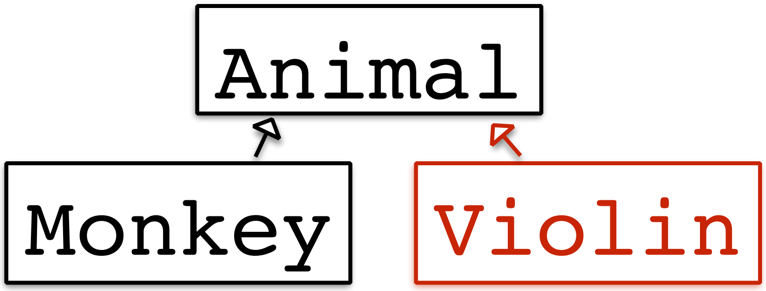
# Identifying Poor Quality Identifiers

- Inconsistencies based on the concepts (cont.)

Conciseness violation:



No hyponymy in a class hierarchy:



# Identifying Poor Quality Identifiers

- Inconsistencies based on the concepts (cont.)
  - Identified using:
    - identifiers to concept mapping
    - identifier inclusion (syntactic conciseness and consistency)
    - ontology
    - number of characters
    - string similarity

TECHNIQUE

# Identifying Poor Quality Identifiers

- Syntactical standardization

```
class:    Compute     // must be a noun
method:   addition    // must be a verb
```

| | | | |
|---|---|---|---|
| FunctionId | ::= | [Context] (Action \| PropertyCheck \| Transformation) | |
| Context | ::= | Qualifier <noun> | |
| Qualifier | ::= | (<adjective> \| <noun>)* | |
| Action | ::= | SimpleAction \| ComplexAction | |
| SimpleAction | ::= | DirectAction \| IndirectAction | |
| ComplexAction | ::= | ActionOnObject \| DoubleAction | |
| **IndirectAction** | ::= | Qualifier <noun> ActionSpecifier | {Head word = <noun>} |
| **DirectAction** | ::= | <verb> ActionSpecifier | {Head word = <verb>} |
| ActionSpecifier | ::= | (<adjective> \| <adverb> \| <preposition> Qualifier <noun>)* | |
| . . . | | | |

# Identifying Poor Quality Identifiers

- Other types of measures

| | |
|---|---|
| overloaded identifiers: | `saveAndPrint` |
| spelling errors: | `Examlpe` |
| useless type: | `String nameString` |

  - Identified using POS analysis, grammatical relations, spell checker, identifier containment

EVALUATION

# Identifying Poor Quality Identifiers

- Case study with quantitative and qualitative analyses
- Precision of detected poor quality identifiers

TOOLS USED

# Identifying Poor Quality Identifiers

- Semantic relations: WordNet or manual
- POS tagging:
  - Minipar or manual
  - WordNet
- Spell checker: Jazzy

SUBTASK

# Identifying Naming Inconsistencies

- Task: Identify entities where the name is inconsistent with the type, functionality, or documentation.
- Examples:
  - method named `isValid` with return type `void`
  - method named `isNavigateForwardEnabled` documented as backward navigation
  - method named `iterator` whose implementation is only creating and returning an object

INPUT

# Identifying Naming Inconsistencies

- Project bytecode
- Source code

OUTPUT

# Identifying Naming Inconsistencies

- Inconsistencies
- Suggested solution

# Identifying Naming Inconsistencies

- Splitting

TECHNIQUE

# Identifying Naming Inconsistencies

- Contrast the name and type of an entity

| | |
|---|---|
| opposite name and type: | `EnterTransport`<br>`exitTransport(..)` |
| set method returns: | `Dimension`<br>`setBreadth(..)` |
| says many, contains one: | `boolean statistics` |

TECHNIQUE

# Identifying Naming Inconsistencies

- Contrast the name and comment of an entity

  opposite name and comment:

  ```
  // … default exclude …
  String INCLUDE_NAME_DEFAULT
  ```

- Defined through a grounded theory approach
- Identified using POS analysis, general ontology, grammatical relations

TECHNIQUE

# Identifying Naming Inconsistencies

- Contrasting the name and implementation of an entity

## Semantic profile of an “iterator” method:

These methods often **call** other methods with the same name and create objects. They never **return** void, **write** parameter values to fields or call themselves recursively, and very rarely write to fields or return parameter values, and rarely have parameters, contain loops, use local variables, do runtime type-checking or casting, return field values, have branches or have multiple return points.

```
public Iterator iterator() throws
        DomainRegistryException{…}
```

TECHNIQUE

# Identifying Naming Inconsistencies

- Contrasting the name and implementation of an entity (cont.)

```
public void isCaching(boolean value) {
   this.caching = value; }
```

Name: **is**-<adjective>

Implementation: **set**-<adjective>:

returns void, writes field, parameter to field.

```
        isCaching => setCaching
```

# Identifying Naming Inconsistencies

- Contrasting the name and implementation of an entity (cont.)
  - Defined empirically
  - Identified using POS analysis

EVALUATION

# Identifying Naming Inconsistencies

- Detection precision
- Developers’ perception

TOOLS USED

# Identifying Naming Inconsistencies

- Semantic relations:
  - WordNet
- POS tagging:
  - WordNet
  - Stanford’s POS Tagger

TASK

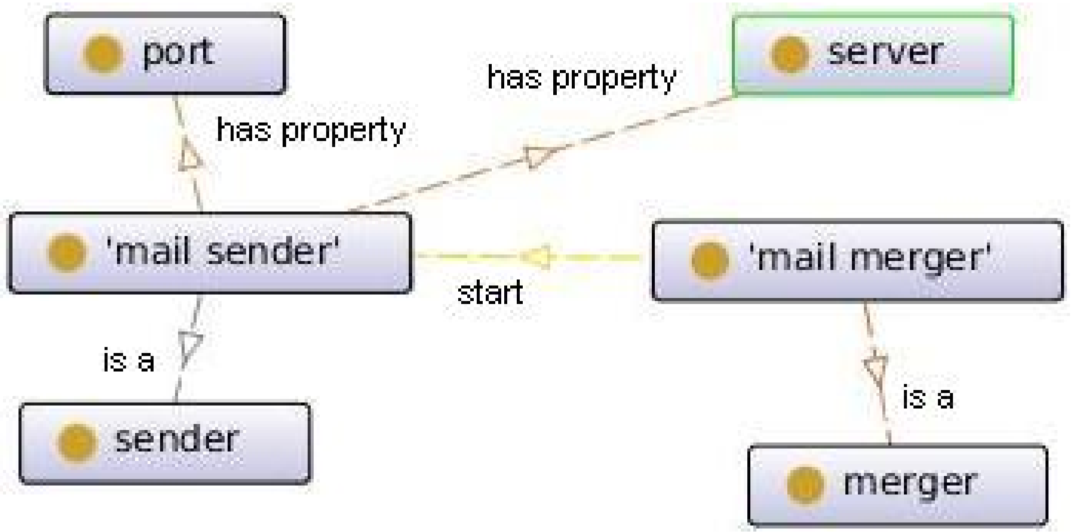
# Building Software Ontologies

- ✓ Domain ontology
- ✓ Identifying semantically related words

TASK

# Extracting Domain Concepts

- Task: automatically extracting domain concepts and relations from source code
- Examples:



INPUT

# Extracting Domain Concepts

- Source code
- Documentation (e.g., user manuals, web sites)

OUTPUT

# Extracting Domain Concepts

- Domain concepts and ontological relations

PREPROCESSING

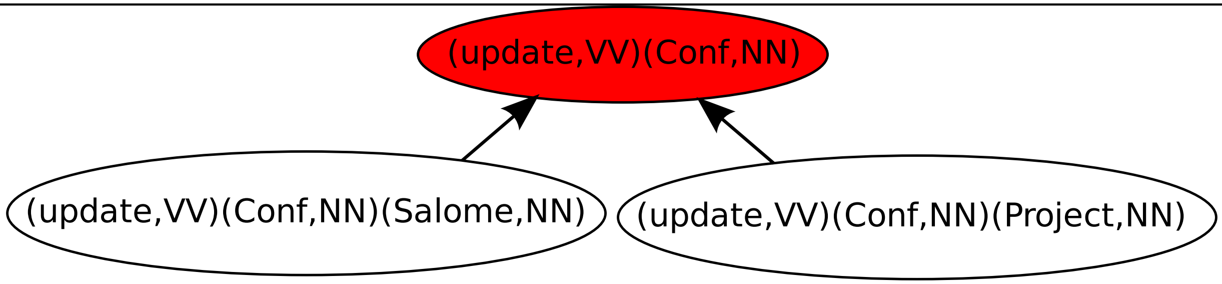
# Extracting Domain Concepts

- Splitting
- Abbreviation expansion
- Stop words removal
- Stemming

TECHNIQUE

# Extracting Domain Concepts

- Hypernym/hyponym relations using the longest common prefix

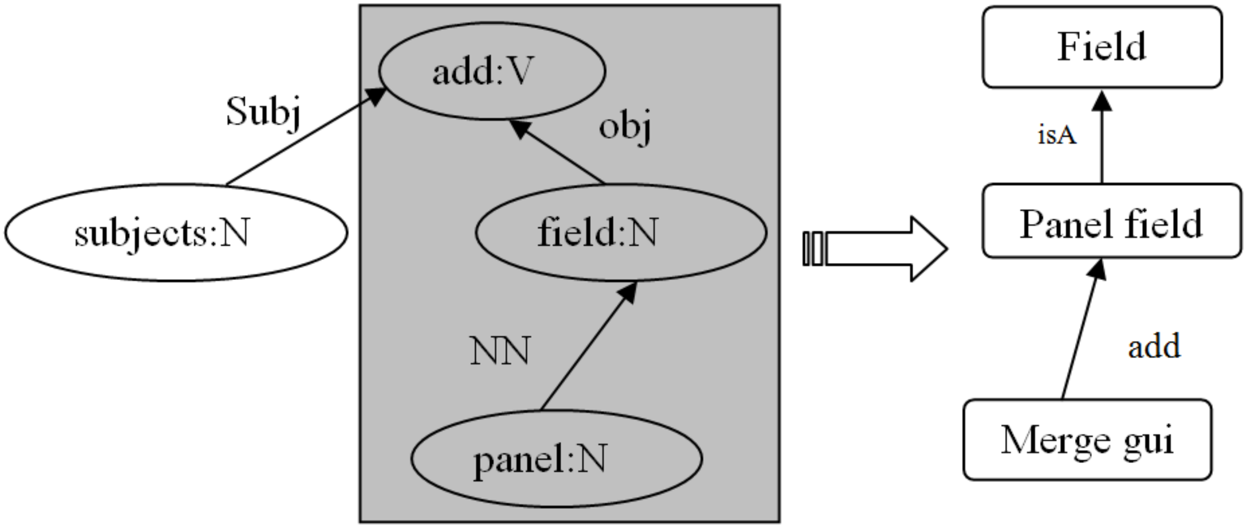
| Step | Identifier 1 | Identifier 2 |
|---|---|---|
| | *updateSalomeConf* | *updateProjectConf* |
| **Tokenization** | *update, Salome, Conf* | *update, Project, Conf* |
| **POS tagging** | *(update,VV),(Salome,NN),(Conf,NN)* | *(update,VV),(Project,NN),(conf,NN)* |
| **Dependency sorting** | *(update,VV),(conf,NN),(salome,NN)* | *(update,VV),(conf,NN),(project,NN)* |
| **Lexical expansion** | *(update,VV),(conf,NN)* | |
| **Lexical relations** | *hypo(updateSalomeConf,updateConf)*<br>*hypo(updateProjectConf,updateConf)* | |
| **Lexical view** |   (update,VV)(Conf,NN)<br>(update,VV)(Conf,NN)(Salome,NN)<br>(update,VV)(Conf,NN)(Project,NN) | |

TECHNIQUE

# Extracting Domain Concepts

- Sentence templates based on constraints for different types of entities
- Example: method `addPanelField` defined in class `MergeGui` generates sentence:

"Subjects add panel field"



TECHNIQUE

# Extracting Domain Concepts

- Filter the ontology using terms based on:
  - keywords
  - pLSI
  - LDA
- A concept is considered as a domain concept if all the terms in the concept name are matched

EVALUATION

# Extracting Domain Concepts

- Precision of the POS tagging
- Number of connected components
- Case study: navigating the concepts for query reformulation in the context of bug location
- Precision and recall of the extracted domain concepts compared to a gold set
- Qualitative analysis

TOOLS USED

# Extracting Domain Concepts

- POS tagging:
    - Minipar
    - WordNet
- Grammatical relations:
    - Minipar
    - TreeTagger
- Topic modeling: Dragon Toolkit

SUBTASK

# Identifying Semantically Related Words

- Task: Identifying pairs of words that are semantically related, e.g., same or opposite meaning
- Examples:
  - call - invoke
  - size - capacity
  - serialize - deserialize
  - header - trailer
  - `makeFullMap` - `makeEmptyMap`

INPUT

# Identifying Semantically Related Words

- Project description and tags extracted from a hosting site
- Source code

OUTPUT

# Identifying Semantically Related Words

- Similar words
- Ranked list of similar tags

PREPROCESSING

# Identifying Semantically Related Words

- Splitting
- Stop words removal
- Stemming

TECHNIQUE

# Identifying Semantically Related Words

- Similarity between terms (VSM with tf-idf)

$$sim(t_1, t_2) = w_1 \times dsim(t_1, t_2) + w_2 \times tsim(t_1, t_2)$$

- Hierarchical taxonomy of tags using based on the similarity between terms using a clustering algorithm

TECHNIQUE

# Identifying Semantically Related Words

- High similarity between pairs of sentences containing at least one common word

"None **mounted** file for this track.”

"None **accessible** file for this track.”

“If you do not have **apr_pool_clear** in a wrapper”

“If you do not have **apr_pool_destroy** in a wrapper”.

# Identifying Semantically Related Words

- High similarity between pairs of sentences containing at least one common word (cont.)

$$\mathit{Similarity\,Measure} = \frac{\text{Number of Common Words in the Two Sequences}}{\text{Total Number of Words in the Shorter Sequence}}$$

- Thresholds are used to filter pairs of related words
- Support measure: +1 when a pair is discovered from different sentences
- Improved similarity using idf

TECHNIQUE

# Identifying Semantically Related Words

- Frequency of comment-code word pairs of main action verbs for methods

```
/** Searches an attribute.*/
XMLAttribute findAttribute(…){…}
```

```
/** Cancels the current HTTP request.*/
void jsxFunction abort(){…}
```

# Identifying Semantically Related Words

- Frequency of comment-code word pairs of main action verbs for methods (cont.)
  - Filter descriptive leading comments
  - Identify documented action from a leading comment
  - Identify the main action from the name of a method

EVALUATION

# Identifying Semantically Related Words

- Precision of the identified pairs of words
- User study evaluating a subset of the identified pairs on a Likert scale.
- Sensitivity evaluation for thresholds (precision and recall)

TOOLS USED

# Identifying Semantically Related Words

- Stanford’s POS Tagger for comments
- Custom POS Tagger for method names
- WordNet

TASK

# Generating Documentation Automatically

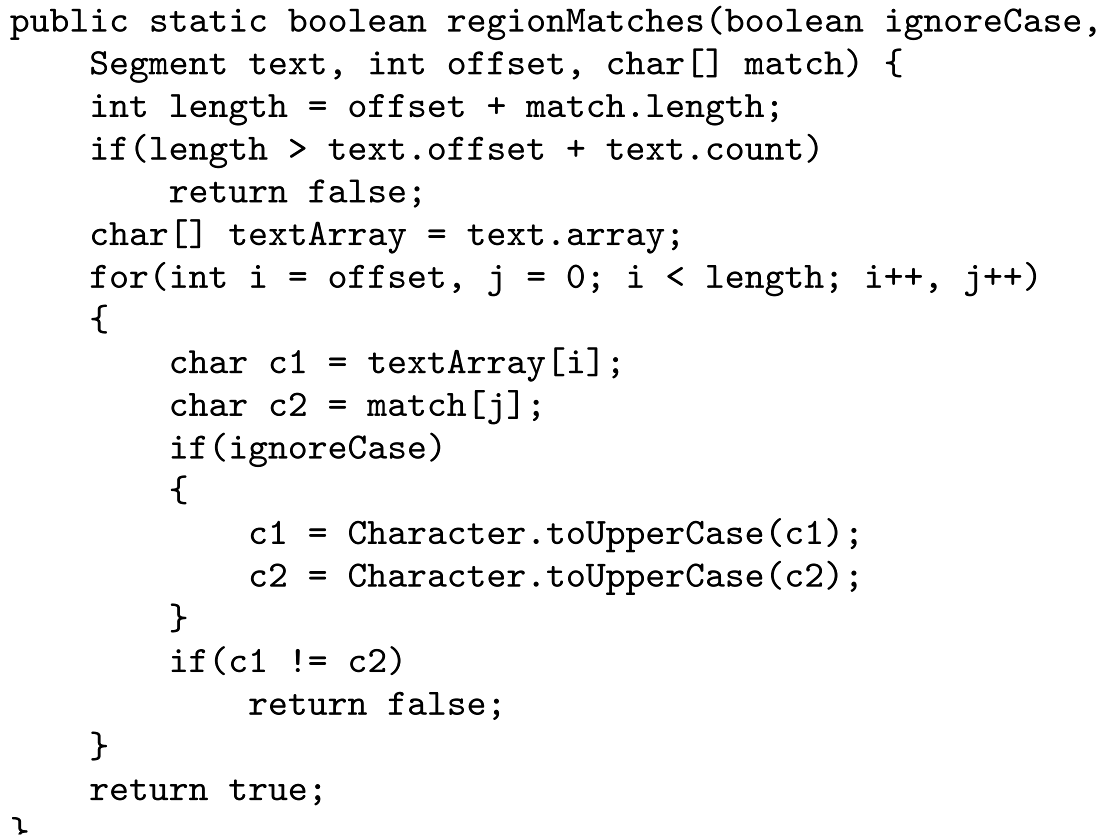
✓ Extracting a set of important keywords

✓ Generating natural language sentences

SUBTASK

# Extracting a Set of Important Keywords

- Task: Identify the keywords that best represent a software artifact
- Example: {“match”, “text”, “ignorecase"}



```
public static boolean regionMatches(boolean ignoreCase,
    Segment text, int offset, char[] match) {
    int length = offset + match.length;
    if(length > text.offset + text.count)
        return false;
    char[] textArray = text.array;
    for(int i = offset, j = 0; i < length; i++, j++)
    {
        char c1 = textArray[i];
        char c2 = match[j];
        if(ignoreCase)
        {
            c1 = Character.toUpperCase(c1);
            c2 = Character.toUpperCase(c2);
        }
        if(c1 != c2)
            return false;
    }
    return true;
}
```

INPUT

# Extracting a Set of Important Keywords

- Source code
- Execution traces

OUTPUT

# Extracting a Set of Important Keywords

- Sets of keywords that best represent each
  - Class
  - Method
  - Execution trace segment

# Extracting a Set of Important Keywords

- Splitting
- Stop words removal
- Stemming

TECHNIQUE

# Extracting a Set of Important Keywords

- Compare IR-techniques
- Eye-tracking experiment to decide on the importance of terms
- IR-techniques: VSM, LSI, LDA
- Weighting schemes: tf, tf-idf, log, and binary-entropy

# Extracting a Set of Important Keywords

- Developers assessing the quality of the summaries
- Comparison with manually summarized artifacts

SUBTASK

# Generating Natural Language Sentences

- Task: Generating natural language sentences summarizing a software artifact.
- Examples
  - Method summary: “Export plan component to svg.”
  - Class summary: “An AbstractPlayer extension for m player handlers. This entity class consists mostly of mutators to the m player handler's state. …”
  - Release note: “New class SearcherLifetimeManager implementing Closeable. …”

INPUT

# Generating Natural Language Sentences

- Project source code/bytecode
- Set of releases
- Issue tracker
- Version control repository

OUTPUT

# Generating Natural Language Sentences

- Natural language sentences representing
  - method comments
  - class comments
  - release notes
  - commit notes

PREPROCESSING

# Generating Natural Language Sentences

- Splitting
- Abbreviation expansion

TECHNIQUE

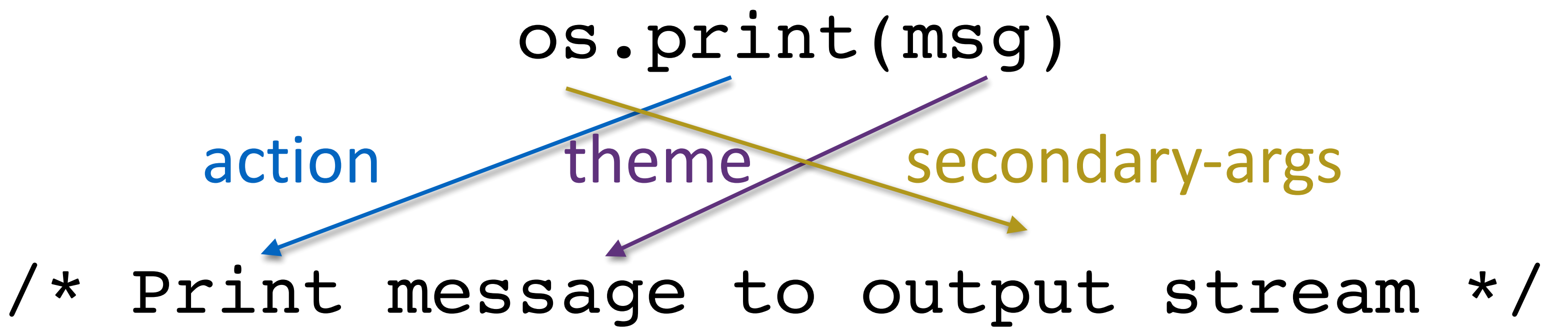
# Generating Natural Language Sentences

- Method summaries
    - Statement selection
        - Ending statements
        - Statement with a method call with the same action
        - Conditional expressions
        - …

# Generating Natural Language Sentences

- Method summaries (cont.)
  - Sentence templates
    - E.g., method call template

action theme secondary-args
and get return-type [if M returns a value]



```
os.print(msg)
```

```
/* Print message to output stream */
```

TECHNIQUE

# Generating Natural Language Sentences

- Class summaries based on class and method stereotypes
  - Filtering using
    - Stereotypes
    - Access-level

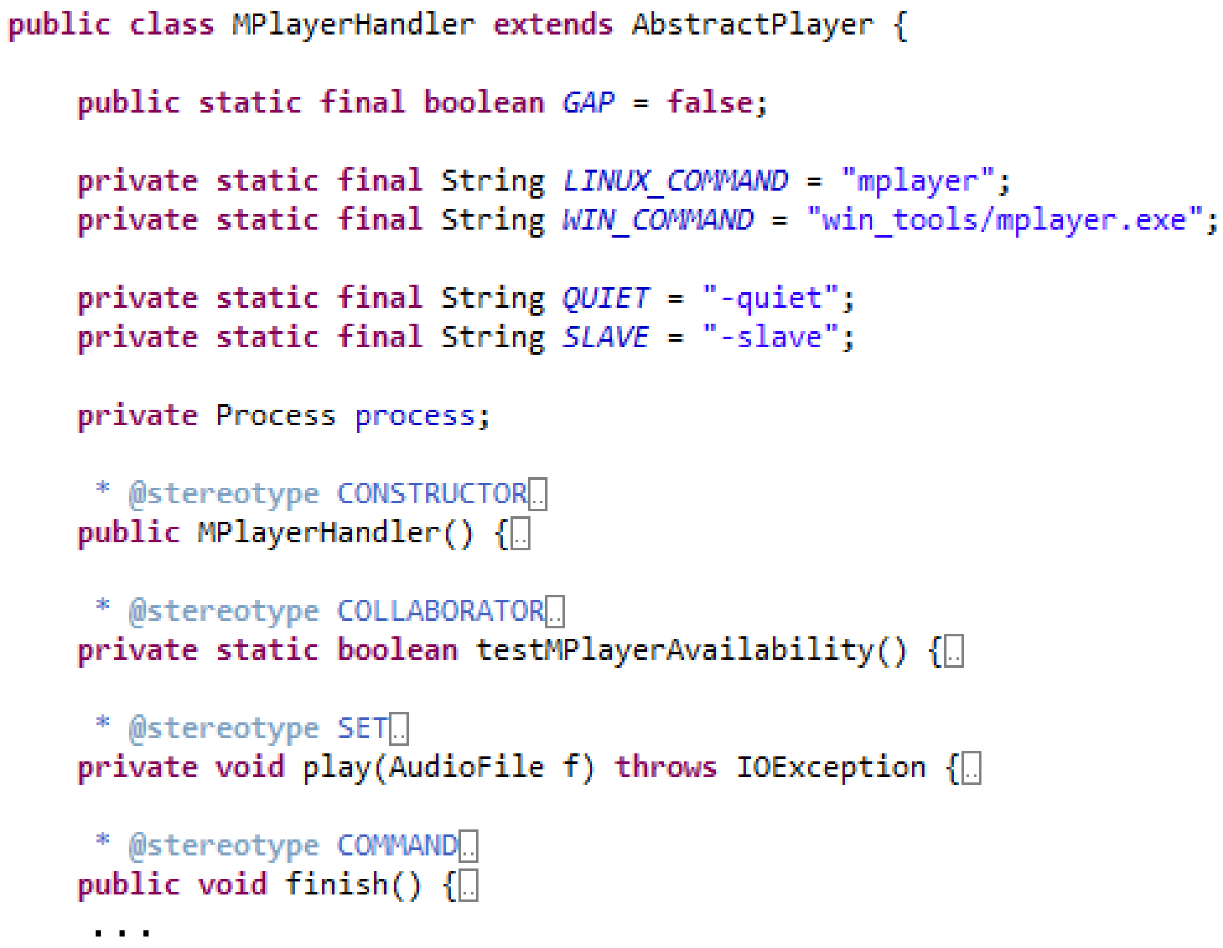
TECHNIQUE

# Generating Natural Language Sentences

- Class summaries based on class and method stereotypes
  - Text generation
    - General description
    - Stereotype description
    - Behavior description
    - Inner classes enumeration

# Generating Natural Language Sentences

- Class summaries based on class and method stereotypes

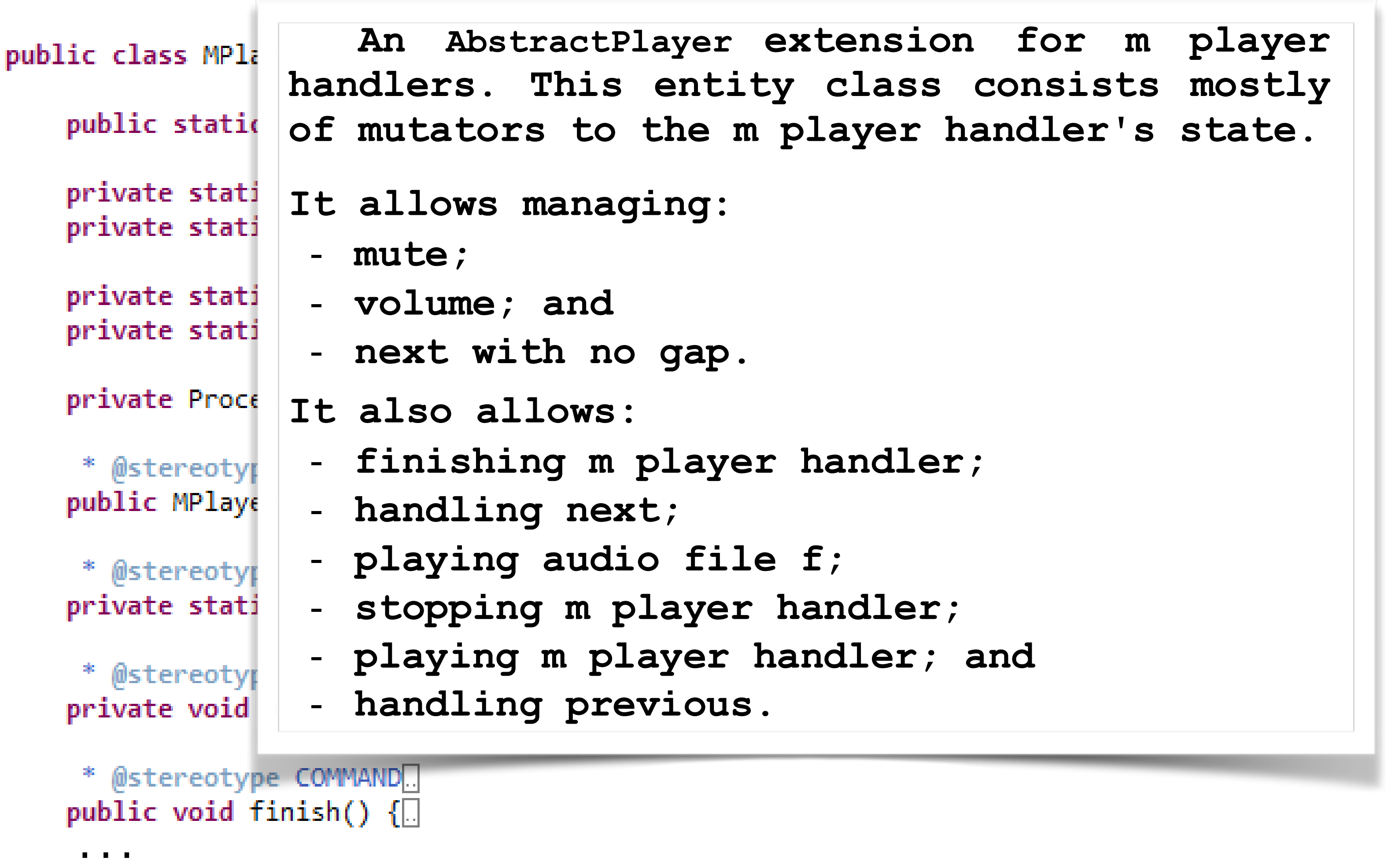


```
public class MPlayerHandler extends AbstractPlayer {

    public static final boolean GAP = false;

    private static final String LINUX_COMMAND = "mplayer";
    private static final String WIN_COMMAND = "win_tools/mplayer.exe";

    private static final String QUIET = "-quiet";
    private static final String SLAVE = "-slave";

    private Process process;

     * @stereotype CONSTRUCTOR[..]
    public MPlayerHandler() {[..]

     * @stereotype COLLABORATOR[..]
    private static boolean testMPlayerAvailability() {[..]

     * @stereotype SET[..]
    private void play(AudioFile f) throws IOException {[..]

     * @stereotype COMMAND[..]
    public void finish() {[..]
     ...
```

TECHNIQUE

# Generating Natural Language Sentences

- Class summaries based on class and method stereotypes



```
public class MPla

    public static

    private stati
    private stati

    private stati
    private stati

    private Proce

     * @stereotyp
    public MPlaye

     * @stereotyp
    private stati

     * @stereotyp
    private void

     * @stereotype COMMAND
    public void finish() {
     ...
```

An AbstractPlayer extension for m player handlers. This entity class consists mostly of mutators to the m player handler's state.

It allows managing:
- mute;
- volume; and
- next with no gap.

It also allows:
- finishing m player handler;
- handling next;
- playing audio file f;
- stopping m player handler;
- playing m player handler; and
- handling previous.

TECHNIQUE

# Generating Natural Language Sentences

- Release notes by organizing changes hierarchically and by using sentence templates
  - Identifying and prioritizing code changes from the versioning systems
    - Files added, removed, moved
    - Classes added, removed, renamed, moved
    - Methods changed (signature, visibility, source code, or set of thrown exceptions)
    - …

# Generating Natural Language Sentences

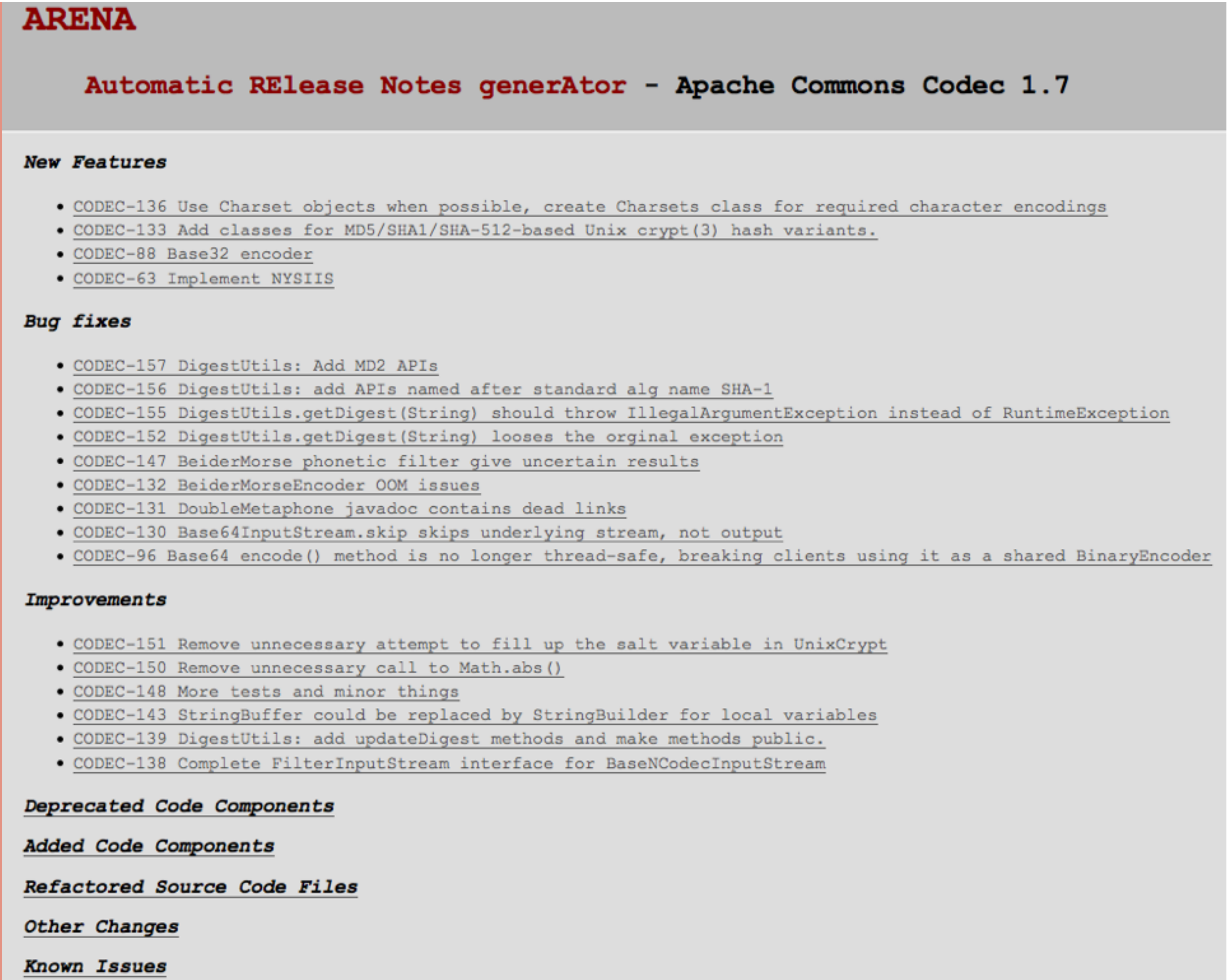
- Release notes by organizing changes hierarchically and by using sentence templates
  - Sentence templates
    - Deleted file: “File <file name> has been removed.”
    - Added class: class summaries (JSummarizer)

TECHNIQUE

# Generating Natural Language Sentences

- Release notes by organizing changes hierarchically and by using sentence templates
  - Other changes considered
    - Licensing
    - Documentation
    - Libraries
    - Refactorings
    - Issues

TECHNIQUE



ARENA

## Automatic RElease Notes generAtor - Apache Commons Codec 1.7

### *New Features*

- CODEC-136 Use Charset objects when possible, create Charsets class for required character encodings
- CODEC-133 Add classes for MD5/SHA1/SHA-512-based Unix crypt(3) hash variants.
- CODEC-88 Base32 encoder
- CODEC-63 Implement NYSIIS

### *Bug fixes*

- CODEC-157 DigestUtils: Add MD2 APIs
- CODEC-156 DigestUtils: add APIs named after standard alg name SHA-1
- CODEC-155 DigestUtils.getDigest(String) should throw IllegalArgumentException instead of RuntimeException
- CODEC-152 DigestUtils.getDigest(String) looses the orginal exception
- CODEC-147 BeiderMorse phonetic filter give uncertain results
- CODEC-132 BeiderMorseEncoder OOM issues
- CODEC-131 DoubleMetaphone javadoc contains dead links
- CODEC-130 Base64InputStream.skip skips underlying stream, not output
- CODEC-96 Base64 encode() method is no longer thread-safe, breaking clients using it as a shared BinaryEncoder

### *Improvements*

- CODEC-151 Remove unnecessary attempt to fill up the salt variable in UnixCrypt
- CODEC-150 Remove unnecessary call to Math.abs()
- CODEC-148 More tests and minor things
- CODEC-143 StringBuffer could be replaced by StringBuilder for local variables
- CODEC-139 DigestUtils: add updateDigest methods and make methods public.
- CODEC-138 Complete FilterInputStream interface for BaseNCodecInputStream

### *Deprecated Code Components*

### *Added Code Components*

### *Refactored Source Code Files*

### *Other Changes*

### *Known Issues*

EVALUATION

# Generating Natural Language Sentences

- Developers
  - Accuracy
  - Content Adequacy
  - Conciseness
  - Importance
  - In-field study

SUBTASK

# Generating Natural Language Sentences

- Tools used:
  - Software Word Usage Model (SWUM)
  - JSummarizer for generating class summaries

TASK

# Concept Location

- Task: determining the start of a change to the code based on a change request
- Change requests are most often formulated in terms of domain concepts
- Examples:
  - "Correct error that arises when trying to paste a text"

    -> find the location where the concept "paste" is implemented in the code
  - "Extend the print functionality to print also double-sided" -> locate where the "print" concept is implemented and extend it

TASK

# Concept Location

- Flavors:
  - Feature location
  - Bug location/localization
  - Concern location

INPUT

# Concept Location

- Source code
  - Identifiers
  - Comments
- Level of document granularity
  - File/class
  - Method/function
- Query
  - Manual
  - Automatic

OUTPUT

# Concept Location

- Ranked list of code elements
- Needs to be evaluated manually by developers
- Quality of output dependent on quality of source code naming conventions/ comments and of the query

PREPROCESSING

# Concept Location

- Text normalization (white space and non-textual tokens removal)
- Splitting
- Stop word removal
- Stemming
- POS Tagging

TECHNIQUE

# Concept Location

- **TR models**:
  - Vector Space Model (VSM)
  - Latent Dirichlet Allocation (LDA)
  - Latent Semantic Analysis (LSA)
  - Okapi BM25 and BM25F
- **NLP**:
  - Action-oriented identifier graph (AOIG)
  - Contextual search using POS tagging, phrase extraction and matching (noun, verb, prepositional phrases)
  - Ontology generation

# Concept Location

- Methodology
  - Studies with developers
    - Developers receive a change request and perform concept location, assisted by a particular tool we want to evaluate
    - Comparison between using the tool/approach and not using it

EVALUATION

# Concept Location

- Methodology
  - Reenactment – automated evaluation
    - Mine repositories for past changes
    - Match a change request (i.e., bug report or feature request) with patches and find the change set (i.e., methods or classes that changed)
    - Use the change request as the starting query
    - Success is achieved when one item in the change set is located
    - Comparison with previous approaches or with CL and without the tool

# Concept Location

- Metrics
  - IR metrics: *Precision*, MAP, MRR, etc. (Recall=1)
  - *Effectiveness* = Rank of first relevant code element (approximation of developer effort)

TOOLS USED

# Concept Location

- Lucene (VSM implementation)
- Mallet (LDA Implementation)
- Dragon Toolkit (SVD, LDA, Porter stemmer, Wordnet)

ISSUES AND SOLUTIONS

# Concept Location

- TR techniques require configuration
  - Based on previous work in IR
  - Based on previous work in SE
  - Heuristics based on empirical evidence
  - Using genetic algorithms to automatically configure TR for a dataset
- Hard to formulate queries
  - Automatic and semi-automatic query reformulation

IMPROVEMENTS

# Concept Location

- Combination with static, dynamic, historical analysis
- Combining results of different IR engines
- Clustering the software/results
  - Adds structure to the results
- Improvements of the IR engine or data
  - Smoothing filters, term boosting, etc.

TASK

# Traceability Link Recovery

- Task: recovering conceptual links between different types of artifacts (source code, documentation, user manuals, tests, design documents, etc.)
- Traceability: the ability to describe and follow the life of a requirement, in both a forward and backward direction [Gotel and Finkelstein 1994]

# Traceability Link Recovery

- Examples: traceability between:
  - Requirements and code
  - Design and code
  - Requirements and design
  - Requirements and test cases
  - Design and test cases
  - Bug reports and maintainers
  - Manual pages to code
  - Emails to code
  - Etc.

# CL vs. Traceability Link Recovery

- Similarities:
  - Both are instances of the *concept assignment problem*
  - Both formulated as TR problems
  - Similar user role: validation and relevance feedback

- Differences:
  - Different input and output -> different evaluation (recall important)
  - Variety of software artifacts
  - No user query

INPUT

# Traceability Link Recovery

- Two sets of of software artifacts (source and target)

- Granularity levels (classes, methods, files, paragraphs, etc.)

OUTPUT

# Traceability Link Recovery

- Ranked list of artifact pairs – candidate links

PREPROCESSING

# Traceability Link Recovery

- Text normalization (white space and non-textual tokens removal)
- Splitting
- Stop word removal (language specific – different for English, Italian, etc.)
- Stemming
- POS tagging (keep nouns)

TECHNIQUE

# Traceability Link Recovery

- *VSM*
- *LSI*
- *Probabilistic models*
- LDA
- Language models
- Jensen-Shannon (JS) Divergence
- Etc.

IMPROVEMENTS

# Traceability Link Recovery

- Relevance feedback to reformulate query
- N-grams (2-grams work better)
- Hierarchical modeling – leverage the hierarchical organization of artifacts
- Logical clustering to discover new links

EVALUATION

# Traceability Link Recovery

- Methodology: developers analyze the ranked list of artifact pairs
  - Analyze the entire list
  - Use a *cut poin*t and analyze the top list
  - Use a *threshold* and analyze the top list

# Traceability Link Recovery

- *Cut point*:
  - Constant: threshold on the number of recovered links
  - Variable: percentage of links that have to be retrieved

- *Threshold*:
  - Constant: a widely adopted threshold is ε = 0.70
  - Scale: percentage of the best similarity value between two artifacts.
  - Variable: projected from [0, 1] into [*min*, *max*], where *min* and *max* are the minimum and maximum similarity values in the ranked list

# Traceability Link Recovery

- Metrics
  - Recall
  - Precision
  - F-measure

TOOLS USED

# Traceability Link Recovery

- Lucene (VSM implementation)
- Mallet (LDA Implementation)
- Dragon Toolkit (SVD, LDA, Porter stemmer, Wordnet)

# Not all software engineering tasks are text retrieval problems

TASK

# Software Categorization

- Task: Assign a finite set of categories to software applications. Each category briefly describes a feature of the application.
- Examples:
  - Database → Apache Cassandra
  - Social → Instagram
  - Build-management → Apache Maven

INPUT

# Software Categorization

- Source code
- Software profiles or descriptions
- Bytecode (for Java applications)
- API calls

OUTPUT

# Software Categorization

- Relevant categories for each application
- Groups of similar applications
- Similarity between two applications

PREPROCESSING

# Software Categorization

- Splitting
- Stop words removal
- Stemming

# Software Categorization

- IR Techniques: LDA, LSI, VSM
- Classifiers: Naïve Bayes, Decision Trees, SVM
- Clustering algorithms: K-means

# Software Categorization

- Gold set:
  - Categorized previously assigned to applications
  - Developers´ opinion about the correctness of recommended categories.
- Precision, recall and F-measure.
- %TP and %FP for classifiers

TASK

# Defect prediction

- Task: Identify entities more likely to be faulty

INPUT

# Defect Prediction

- Project source code
- Level of granularity (class, method)

OUTPUT

# Defect Prediction

- For each entity predict
  - The probability of having at least one fault
  - Whether it is fault prone or fault free
  - The number of faults

PREPROCESSING

# Defect Prediction

- Splitting
- Stop words removal
- Stemming

TECHNIQUE

# Defect Prediction

- Lexical metrics (VSM with tf-idf, LSI with tf-idf, metrics for quality of identifiers)
- Check if lexical metrics capture different information compared to structural metrics
- Prediction models

EVALUATION

# Defect Prediction

- Case studies
- Comparison of prediction models

TOOLS USED

# Defect Prediction

- Semantic relations:
  - WordNet
- POS tagging:
  - Minipar

TASK

# Bug Triaging

- Bug classification
- Recommend developer(s)
- Summarization of bug reports

✓ Duplicate bug detection

TASK

# Duplicate Bug Detection

- Task: automatically detect bug reports concerning the same fault
- Examples:
  - Bug #21196: “I just see many description where people continuously requesting google for support urdu in Andriod …”
  - Bug #20161: “Hello I’m unable to read any type of urdu language text messages. Please add urdu language in future updates of android …"

INPUT

# Duplicate Bug Detection

- 2 bug reports
- 1 bug report

OUTPUT

# Duplicate Bug Detection

- True is the two bug reports are duplicate, false otherwise
- List of ranked top n most similar bug reports

# Duplicate Bug Detection

PREPROCESSING

- Splitting
- Stemming
- Stop words removal
- Synonym and abbreviation replacement
- Spelling error correction

# Duplicate Bug Detection

- Defining metrics based on the topic distribution (LDA) and machine learning classifiers
- VSM with cosine similarity (Dice, Jaccard)

EVALUATION

# Duplicate Bug Detection

- Evaluation:
  - accuracy
  - AUC
  - Kappa
  - recall rate
  - interviews

TOOLS USED

# Duplicate Bug Detection

- LDA: MALLET

TASK

# Team Management

✓ Identify distress or happiness

- Characterize personality of successful people
  - Stack Overflow (SO) users
  - Developers

SUBTASK

# Identify Distress or Happiness

- Task: Identify sentiments/emotions from a written communication
- Examples:
  - “That’s great work guys!” (Joy)
  - “Who are the stupid people who manages this group.” (Negative sentiment)

INPUT

# Identify Distress or Happiness

- Written communication, e.g.,
  - Mailing lists
  - Issue tracking systems

OUTPUT

# Identify Distress or Happiness

- Sentiment scores (1 per communication)
- Emotions (possibly more than 1 per communication)

PREPROCESSING

# Identify Distress or Happiness

- Filter out automatically sent emails
- Remove quoted parts of emails threads
- Filter out any non-natural language text

TECHNIQUE

# Identify Distress or Happiness

- Automatically assign a sentiment score per email (the most extreme, i.e., Max)
- Manually assign emotions to issue comments

# Identify Distress or Happiness

- User study
  - Feasibility of manually detecting emotions from issue tracking systems (inter-rater agreement)
  - Precision of the automatically assigned sentiment scores

TOOLS USED

# Identify Distress or Happiness

- SentiStrength

# Present and Future of NLP and TR for SE

- One of the fastest growing research areas in SE

- There is a need for more benchmarks and open data to support comparison to previous approaches

- Current trends:
  - Combining different approaches for better overall results
  - Adapting NLP and TR to the properties of individual SE datasets and tasks

# Evaluating/Adapting NLP and TR for SE

- Part-Of-Speech (POS) tagging
  - evaluating preprocessing templates
  - comparing POS taggers
  - technique for tagging identifiers
- English-based semantic similarity techniques
- Stemming
- Tuning TR parameters to individual SE datasets

# Slides and Additional Material

- http://www.cs.fsu.edu/~shaiduc/TRNLP