GENERATING PARAMETER COMMENTS AND INTEGRATING WITH METHOD SUMMARIES

Giri prasad Sridhara, Lori Pollock and K. Vijay-Shanker

Natural Language Program Analysis Laboratory
Computer and Information Sciences
University of Delaware
U.S.A

19th IEEE International Conference on Program Comprehension (ICPC)
Kingston, Canada: June 22-24, 2011
Comments are Useful for Comprehension

- Often developers need:
  - high level overview of method’s intent
  - may not have time to read entire method
- Method signature may not always clarify intent

```
public static void main(String[] args)
```

- Summary comments for a method can help
  - Developer written or
  - Automatically generated [Our previous work: ASE 2010 *]

```
/** Start MetaServer **/
public static void main(String[] args)
```

* ACM SIGSOFT Distinguished Paper
Summary Comments to Parameter Comments

- For conciseness, generated summary may omit describing parameter.

```java
/** Start MetaServer */
public static void main(String[] args)
```

- Concise Summary: Does not describe parameter.

- Developer may need to know parameter role:
  - Parameter comments can help.

- Parameter comment must convey:
  - how the parameter is used in achieving a method’s intent
  - i.e., it must be connected with method summary.

- We automatically generate:

  ```java
  @param args: create meta server using args
  ```

  - Clarifies the parameter’s role
  - Connected with method’s intent (summary) via “meta server”
Benefits of Generating Parameter Comments

- Overcome lack of developer written comments
  - Only 19% of 99,000 parameters had a developer written @param comment [Across 18 projects]
- Ensure comments do not get obsolete
- Ensure better commenting by developers
  - Developers more likely to comment in an already well-commented project [Marin 05]
Towards generating @param comments

@summary: start meta server

```java
public static void main(String[] args)
    int port = -1;
    try {
        port = Integer.parseInt(args[0]);
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("Usage: java net.s");
        System.exit(-1);
    }
    catch (NumberFormatException e) {
    System.out.println("Usage: java net.s");
    System.exit(-1);
}   MetaServer metaServer = null;
    try {
        metaServer = new MetaServer(port);
    } catch (IOException e) {
    logger.warning("Could not create M");
    System.exit(-1);
}   
    metaServer.start();
```
Generating @param comments: Connecting parameter use to method intent

```java
public static void main(String[] args)
    int port = -1;
    try {
        port = Integer.parseInt(args[0]);
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("Usage: java net.server <port>");
        System.exit(-1);
    } catch (NumberFormatException e) {
        System.out.println("Usage: java net.server <port>");
        System.exit(-1);
    }
    MetaServer metaServer = null;
    try {
        metaServer = new MetaServer(port);
    } catch (IOException e) {
        logger.warning("Could not create MetaServer");
        System.exit(-1);
    }
    metaServer.start();
```
Our Contributions

Developed Heuristics to Automatically:

- Identify main role of a formal parameter in a method
- Connect parameter with method’s computational intent
- Generate succinct parameter comment phrases

Evaluated Accuracy, Utility and Necessity:

Experienced developers judged parameter comments

Outline for talk
Challenge: Identify Primary Role of a Parameter

- If parameter is used in multiple statements, which is the “primary” usage?

```java
void buildResourceItem(ResourceType r, TreeNode parent) {
    ImageIcon icon = library.getScaledBonusImageIcon(r, 0.75f);
    TreeNode n = new TreeNode(new TreeItem(r, r.name, icon));
    parent.add(n);
}
```

- Need to estimate “closeness” of each use to method’s intent: How do we
  - represent method’s computational intent?
  - measure “closeness”?
  - use “closeness” to identify parameter’s main role?
Representing a Method’s Computational Intent

- Method signature may not always provide intent
- Developer written summary comments:
  - Not all methods have them
  - Not easy to automatically find if comment is:
    - a summary
    - up to date with code
- Automatically generate summaries
- Use generated summary to represent method intent
  - Also use statements from which summary was created
Challenge:
Linking Parameter Comment to Method’s Intent

- Java Doc Writing Guidelines:
  - Parameter comments must follow method summary
- Thus, param comments are *always* in *context* of summary. Need to:
  - ensure that generated param comments are placed in context of summary
  - identify “linking context”: variables & statements

- How can we:
  - automatically identify linking context?
  - generate phrases linking param comment with summary?
  - ensure generated comment is not verbose?
Process Foundation: Traditional Program Analysis and Text Preprocessing

Method M signature & body

Split Identifiers into Words [Enslen et al 09]

Expand Abbreviated Words [Hill et al 08]

Build Abstract Syntax Tree
Construct Control Flow Graph
Create Def-Use chains
Generating parameter comments requires identification of a method’s action and its arguments.

Software Word Usage Model (SWUM) to identify linguistic elements in a method

For each formal parameter $fp$ of $M$

1: Find $\text{AllUsesSet}$ i.e. all statements in which $fp$ is used

2: Prune unimportant statements from $\text{AllUsesSet}$

3: For each remaining statement in $\text{AllUsesSet}$: Estimate closeness to $M$’s intent

4: Select statements closest to $M$’s intent

5: Generate phrases for selected statements

6: Apply transformations to get concise parameter comments

OUTPUT:
Parameter comments connected to $M$’s summary
Process: Step 2 : Prune Uses

- **Aim:** Find important use(s) of parameter
- **Different Heuristics:**
  - **Ex:** Prune uses that are less likely to be executed
    - References are typically not null

[Ball & Larus Heuristics : PLDI 93]

```java
void startGUI(String guiName, String[] args) {
    assertTrue(guiName != null, "guiName must be non-null")
    assertTrue(args != null, "args must be non-null")
    IMegaMekGUI mainGui = getGui(guiName);
    if (mainGui == null)
        displayMessageAndExit(UNKNOWN_GUI_MESSAGE+guiName);
    else {
        StringBuffer message = new StringBuffer("Starting GUI"
        dumpArgs(message, args);
        displayMessage(message.toString());
        mainGui.start(args);
    }
}
```

Assertion Statement: Prune Use

Primary Use: Not Pruned

Infrequently Executed

Frequently Executed

Prune Use

Other heuristics for pruning uses
Process

INPUT:
• Method M
• M’s Structural & Linguistic Representations
• M’s Computational Intent (generated summary for M)

For each formal parameter fp of M

1: Find AllUsesSet i.e. all statements in which fp is used
2: Prune unimportant statements from AllUsesSet
3: For each remaining statement in AllUsesSet: Estimate closeness to M’s intent
4: Select statements closest to M’s intent
5: Generate phrases for selected statements
6: Apply transformations to get concise parameter comments

OUTPUT:
Parameter comments connected to M’s summary
Aim: Among non-pruned uses, find most important use(s) of parameter

Different closeness estimation heuristics
- Parameter described in summary text
- Parameter used in statement selected for summary
- ...
- Link to summary via intermediate variables
- ...

Closer to computational intent
Process: Step 3: Closeness Estimation Heuristic 1

Parameter already appears in summary text

```c
void removeWindow(Shell shell) {
    shells.remove(shell);
    notifyRemoveListeners(shell);
}
```

- Summary clarifies role of param shell
- Can use summary phrase “as is” for param comment
- Can transform phrase using transformation templates

We automatically generate:
@summary: Remove given shell from shells
Param used in statement selected for summary

Similar to the previous case except: param *does not* appear in final summary text

```java
ParserResult importToOpenBase(String argument) {
    ParserResult result = importFile(argument);
    if (result != null)
        result.setToOpenTab(true);
    return result;
}
```

@ summary:
Import file and get parser result

- For conciseness, "argument" was omitted from summary
- @param comment: Augment summary phrase with parameter

We automatically generate:
@param argument: import file *using given argument string* and get parser result
Process: Step 3: Closeness Estimation Heuristic 6

Link to summary via intermediate variables

```java
public static void main(String[] args) {
    int port = -1;
    try {
        port = Integer.parseInt(args[0]);
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("Usage: java net.server args[0]");
        System.exit(-1);
    } catch (NumberFormatException e) {
        System.out.println("Usage: java net.server args[0]");
        System.exit(-1);
    }
    MetaServer metaServer = null;
    try {
        metaServer = new MetaServer(port);
    } catch (IOException e) {
        logger.warning("Could not create MetaServer using port");
        System.exit(-1);
    }
    metaServer.start();
}
```

@summary: start meta server

@param args: parse integer & get port using given args. Create meta server using port

Concise Version:

```
@param args: create meta server, using args

"port": intermediate variable
```

Other heuristics
### Process

**INPUT:**
- Method M
- M’s Structural & Linguistic Representations
- M’s Computational Intent (generated summary for M)

For each formal parameter fp of M

1: Find AllUsesSet i.e. all statements in which fp is used
2: Prune unimportant statements from AllUsesSet
3: For each remaining statement in AllUsesSet: Estimate closeness to M’s intent
   - **4: Select statements closest to M’s intent**
5: Generate phrases for selected statements
6: Apply transformations to get concise parameter comments

**OUTPUT:**
Parameter comments connected to M’s summary
Use of Closeness Estimation Heuristics

Param $r$: Used on lines 2, 3
Line 2 or Line 3 to describe it?

- Line 2: *No* direct link to summary. But,
  - From $r$, get *icon*: Line 2
  - From *icon*, get $n$ [*TreeNode*] which is in summary: Line 3
  - Linking: 2 Statements needed

- Line 3: *Link* to summary via variable, $n$, (*TreeNode*)
  - Linking: Only one statement needed
  - Thus, usage is *closer* to intent of “buildResourceItem”

- Thus, we choose Line 3 to describe param $r$

@summary: *add tree node to given parent tree node*

@summary: Does *not* describe param $r$

@summary: add tree node to given parent tree node

**Summary:**
- @param $r$: *create tree node using $r$*
Process

INPUT:
• Method M
• M’s Structural & Linguistic Representations
• M’s Computational Intent (generated summary for M)

For each formal parameter fp of M

1: Find AllUsesSet i.e. all statements in which fp is used
2: Prune unimportant statements from AllUsesSet
3: For each remaining statement in AllUsesSet: Estimate closeness to M’s intent
4: Select statements closest to M’s intent
5: Generate phrases for selected statements
6: Apply transformations to get concise parameter comments

OUTPUT:
Parameter comments connected to M’s summary
Process: Step 5: Generate Phrases

Methods perform actions ➔ Generate verb phrase

Semantics:
- Action
- Theme
- Secondary Argument

Generate Phrases for Selected Statements using Templates
INPUT:
• Method M
• M’s Structural & Linguistic Representations
• M’s Computational Intent (generated summary for M)

For each formal parameter fp of M

1: Find AllUsesSet i.e. all statements in which fp is used
2: Prune unimportant statements from AllUsesSet
3: For each remaining statement in AllUsesSet: Estimate closeness to M’s intent
4: Select statements closest to M’s intent
5: Generate phrases for selected statements
6: Apply transformations to get concise parameter comments

OUTPUT:
Parameter comments connected to M’s summary
Process: Step 6: Apply Phrase Transformations

- For a summary, we generate verb phrases
  
  Add given menu to menu bar

- By convention: param comments emphasize parameter
  
  @param menu : which is added to menu bar

- Transform the verb phrases to put emphasis on param, using relative clauses

- Transformation templates
IF formal-parameter corresponds to theme in the generated verb-phrase AND verb-phrase has a secondary-argument THEN

transformed phrase = which is <pastParticiple(action)>  
<preposition in secondary-argument> 
<remaining part of secondary-argument>

Where m is a formal parameter to the enclosing method

mb.add(m) → UD-GenPhrase → add given menu to menu bar

Above transformation template applies

@ param m : which is added to menu bar

Other transformation templates exist
Evaluation

- **Accuracy:** of generated param comments

- **Standalone-Utility:** of param comments in explaining param’s role in method

- **Integrated-Utility:** of phrases added to summary in explaining the overall method intent

- **Necessity:** of phrases added to summary (for param integration)
Evaluation Set-up

- **Programmers (Evaluators):**
  - 9 Java Programmers: Post Docs & Grad Students
  - Expert or advanced programmers
  - Programming experience: 4 -- 20 years; Median: 12 years

- **6 large open source Java projects**

- **Evaluated param comments & summaries for 18 methods**
  - 33 parameters across 18 methods
  - 34 modified/added phrases for param integration with summary
    - 22 new phrases added to existing summaries
    - 12 existing summary phrases augmented

- **Subjectivity:** 3 human opinions per method

- **Avoided biasing evaluators:**
  By not giving a definition or an example of a parameter comment
Results: Accuracy of Parameter Comments

Developers opinion on:
Accuracy of Parameter Comments

<table>
<thead>
<tr>
<th>Opinion</th>
<th># Individual Opinions</th>
<th>Majority of 3 Opinions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accurate</td>
<td>89</td>
<td>32</td>
</tr>
<tr>
<td>Slightly Inaccurate</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>Inaccurate</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>99</td>
<td>33</td>
</tr>
</tbody>
</table>

33 parameters and 99 Individual Responses
Results: Standalone-Utility

Developers opinion on:
Utility of param comments in explaining param’s role in method

<table>
<thead>
<tr>
<th>Opinion</th>
<th># Opinions</th>
<th>% Opinions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critically Important</td>
<td>47</td>
<td>47.5</td>
</tr>
<tr>
<td>Important</td>
<td>42</td>
<td>42.5</td>
</tr>
<tr>
<td>Neither important nor unimportant</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Not important</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Detrimental</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>99</td>
<td>100</td>
</tr>
</tbody>
</table>

33 parameters and 99 Individual Opinions
Results: Integrated-Utility

Developers opinion on:
Utility of the phrases added to summary in explaining overall intent of method

<table>
<thead>
<tr>
<th>Opinion</th>
<th># Opinions</th>
<th>% of opinions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critically Important</td>
<td>16</td>
<td>29.6</td>
</tr>
<tr>
<td>Important</td>
<td>25</td>
<td>46.3</td>
</tr>
<tr>
<td>Neither important nor unimportant</td>
<td>10</td>
<td>18.5</td>
</tr>
<tr>
<td>Not important</td>
<td>12</td>
<td>3.7</td>
</tr>
<tr>
<td>Detrimental</td>
<td>1</td>
<td>1.9</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>54</strong></td>
<td><strong>100</strong></td>
</tr>
</tbody>
</table>

18 methods and 54 Individual Opinions
## Results: Necessity

Developers opinion on:  
Necessity of phrases added to summary  
(for parameter integration)

<table>
<thead>
<tr>
<th>Opinion</th>
<th># Individual Opinions</th>
<th>Majority of 3 Opinions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Necessary</td>
<td>69</td>
<td>25</td>
</tr>
<tr>
<td>Tolerable</td>
<td>30</td>
<td>9</td>
</tr>
<tr>
<td>Unnecessary</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>102</strong></td>
<td><strong>34</strong></td>
</tr>
</tbody>
</table>

34 phrases added for integration & 102 Individual Opinions

*Overall Encouraging Results*
Our Other Work in Perspective

- Summary Comment Generation for Java Methods [ASE 2010]
  - Identified individual statements for summary
  - Generated phrases for each statement
  - For conciseness, summary may omit parameter
  - This work complements the summary generation work
    Enables more descriptive summaries with parameter role explained
High Level Action Identification and Description [ICSE 2011]

```java
for (int x = 0, n = vAttacks.size(); x < n; x++) {
    WeaponAttackAction waa = vAttacks.elementAt(x);
    float fDanger = getExpectedDamage(g, waa);
    if (fDanger > fHighest) {
        fHighest = fDanger;
        waaHighest = waa;
    }
}
```

Generated Description After High Level Action Detection

```plaintext
// Find weapon attack action object
// (in vectorAttacks) with highest expected damage
for (int x = 0, n = vAttacks.size(); x < n; x++) {
```

Focus is not on clarifying parameter role
Related Work

- Comments help in program comprehension
  - Takang et al 96, Tenny 88
- Obviate comments by descriptive identifier names [Fowler 99]
  - But descriptive names $\rightarrow$ long names
  - Long names hinder readability [Binkley et al 08]
- Encourage and facilitate writing comments
  - Prompt to enter comments [Erickson 82]
  - Documentation-first approach [Knuth 84]
- Generate comments
  - Comments from specifications [Robillard 86]
  - Exception Doc [Buse & Weimer 08]
  - API cross reference [Long et al 09]

Limitation:
Not applicable for legacy code / do not generalize to param comments

- Key Statements [Harman et al 02]
  - Focus not on identifying main parameter role
Future Work

- Use high level action identification & description for parameter comment generation
- Develop additional heuristics
- Evaluate with inexperienced programmers
- Explore porting to other languages
Synopsis of Contributions

- Generated parameter comments:
  - explaining parameter’s role in achieving method’s intent
  - First technique to do so
- Positive feedback from experienced developers

Acknowledgement: NSF Grants CCF-0702401 & CCF-0915803
Other parameter comments

We generate other parameter comments as well

```java
addToolTo(tb, editor, ..., "createTriangle", labels);
addToolTo(tb, editor, ..., "createEllipse", labels);
addToolTo(tb, editor, ..., "createEllipse", labels);
addToolTo(tb, editor, ..., "createDiamond", labels);
```

Different call sites: Fourth parameter always begins with “create”

```java
public void addToolTo(JToolBar tb, DrawingEditor editor,
        Tool tool, String labelKey,
        ResourceBundleUtil labels) {
```

Augment parameter comment for “labelKey”

```java
@param labelKey : ..... Starts with “create”
```

Benefits:

Alerting developer that there is a convention followed
addToolTo(tb, editor, ..., "createEllipse", labels);
public void addToolTo(JToolBar tb, DrawingEditor editor,
        Tool tool, String labelKey,
        ResourceBundleUtil labels) {

        addToolTo(tb, editor, ..., "createEllipse", labels);
        addToolTo(tb, editor, ..., "createDiamond", labels);
        addToolTo(tb, editor, ..., "createTriangle", labels);

        addToolTo(tb, editor, ..., "createDiamond", labels);
        addToolTo(tb, editor, ..., "createTriangle", labels);
        addToolTo(tb, editor, ..., "createEllipse", labels);
Comments are Useful for Comprehension

- Often developers need a high level overview of the computational intent of a method without having to read the entire method
  - Ex: A concern location tool may return a method signature in response to a user query

- When the signature does not provide enough details to distinguish relevance to the task:
  - A leading summary comment of the method can help
  - The summary can be developer written or automatically generated [ASE 2010]

```java
/**
 * @param args The command-line options.*
 */
public static void main(String[] args) {
```

- The above summary is good as it provides a succinct abstraction of the “main” method’s computational intent
- However, the summary does not clarify the role of the parameter, args

Giriprasad Sridhara
Parameter Comments

/** Creates and starts a new MetaServer. *
 * @param args The command-line options. */
public static void main(String[] args) {

- Developers use @param tag to write comments that convey additional information about a parameter
- Here, the developer’s @param comments does not clarify how the parameter is used
- We believe that it is desirable for a parameter comment to suggest how the parameter is used in achieving the computational intent of the method

@param args: using which, meta server is created

Clarifies the parameter’s role & Connected with the method’s intent (summary) via the phrase “meta server”
Our Other Work in Perspective

- **Summary Comment Generation for Java Methods**  
  [ASE 2010]
  - Identified *individual* statements for a method summary
  - Generated phrases for each statement in isolation
  - In order to be concise, generated summary may omit mentioning the role of the parameter
  - *This work enables more descriptive summaries with the role of the parameter explained*

- **High level action identification and description**  
  [ICSE 2011]
  - Identified high level actions and descriptions need not always contain a description of the parameter role

Giriprasad Sridhara
Overview of Automated Process

INPUT:

• Method M signature and body
• Structural & Linguistic Representations of M
• Computational Intent of M (i.e., generated summary for M)

For each formal parameter $fp$ of M

1: Find AllUsesSet i.e. all statements in which $fp$ is used
2: Prune unimportant statements from AllUsesSet such that cardinality is at least 1
3: For each remaining statement in AllUsesSet:
   Estimate closeness to M’s computational intent
   Identify linking context information
4: Select statements closest to M’s computational intent
5: Generate Phrases for selected statements and additional statements in the linking context
6: Apply transformations to create concise parameter comments

OUTPUT:

Comment phrases for each formal parameter of M which are connected to summary of M
Use of Closeness Estimation Heuristics

```java
void buildResourceItem(ResourceType r, TreeNode parent) {
    ImageIcon icon=library.getScaledBonusImageIcon(r, 0.75f);
    TreeNode n = new TreeNode(new TreeItem(r,r.name,icon));
    parent.add(n);
}
```

@summary: add tree node to given parent tree node

Param `r`: Used on lines 2, 3

Summary: Does NOT describe param `r`

Line 2 or Line 3 to describe param `r`?

- **Line 3:** Link to summary via variable, `n`, (`TreeNode`)  
  - Linking: Only one statement needed
  - Thus, we choose Line 3 to describe the parameter, `r`

Line 2: No direct link to summary
- From `r`, we get `icon` (line 2)
- From `icon`, we get `n` (line 3)
- Linking: 2 Statements needed

@summary: add tree node to given parent tree node

@summary: add tree node to given parent tree node

@summary: add tree node to given parent tree node
Connecting parameter use to method intent

/** Creates and starts a new MetaServer.
 * @param args The command-line options.*
 */
public static void main(String[] args) {
    int port = -1;
    try {
        port = Integer.parseInt(args[0]);
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("Usage: java new MetaServer \
                        -p <port> -h <host> \
                        [other options] ");
        System.exit(-1);
    } catch (NumberFormatException e) {
        System.out.println("Usage: java new MetaServer \
                        -p <port> -h <host> \
                        [other options] ");
        System.exit(-1);
    }
    MetaServer metaServer = null;
    try {
        metaServer = new MetaServer(port);
    } catch (IOException e) {
        logger.warning("Could not create MetaServer\n                        on port: " + port);
        System.exit(-1);
    }
    metaServer.start();
}

We can generate:
Get port using the given args

But not connected with the method’s intent (summary)

We automatically generate:
@param args: create meta server using the param

Connected with the method’s intent (summary) via the phrase “meta server”

Developer’s summary:
No mention of Parameter’s role

Developer’s @param comments:
No mention how the parameter is used
Parameter Comments (?)

- Motivate: often devs like to get a gist wo reading entire method

- Need to see a good leading comment ie summary + param comments (preferably several params)

- Signature without body

- What do you lose if you don’t have param comments but only summary ie Why param comments are important? (black out the param comments)

- Statistics on how many methods don’t have param comments.

- Previously we were looking at summaries but now we are looking at param comments
Approach: Step 3: Closeness Estimation Heuristics

- Ubiquitous method
- Fig 7
Approach: Step 3: Closeness Estimation Heuristics

- Link to summary via variable in summary phrase
- Fig 2
Approach: Step 3:
Closeness Estimation Heuristics

- Link to summary via variable in summary statement but not in phrase
- Fig 8