INF

FAKULTÄT FÜR
INFORMATIK

Use-Case-Specific Source-Code Documentation for Feature-Oriented Programming

Sebastian Krieter, Reimar Schröter, Wolfram Fenske, Gunter Saake

University of Magdeburg, Germany

21. January 2015

SPONSORED BY THE

Federal Ministry
of Education
and Research

BMBF Grant 01IS14017B

Motivation

Feature-Oriented Programming (FOP) - An example

Chat

```
1
2  // original method definition
3  public static void send(String line) {
4      sendObject(toTextMessage(line));
5  }
6
7  // method refinement                          Commands
8  public static void send(String line) {
9      if (line.startsWith("/")) {
10         // interpret command ...
11     } else {
12         original(line);
13     }
14 }
```

Motivation

Feature-Oriented Programming (FOP)

- Generate products by composing different sets of features

$\Rightarrow$ In long term, reduces the amount of implementation effort

Motivation

Feature-Oriented Programming (FOP)

- Generate products by composing different sets of features

⇒ In long term, reduces the amount of implementation effort

Source-Code Documentation

- Important for source-code comprehension

⇒ Prevents bugs and loss of development time

Motivation

Feature-Oriented Programming (FOP)

- Generate products by composing different sets of features

⇒ In long term, reduces the amount of implementation effort

Source-Code Documentation

- Important for source-code comprehension

⇒ Prevents bugs and loss of development time

↘ ↙

FOP: **No** support for source-code documentations

- Code separated in feature modules
- Refinements of classes, methods, and fields

Documentation in FOP

Documentation use cases in FOP

- Several developers are addressed

$\Rightarrow$ Different viewpoints on a software product line (SPL)

$\Rightarrow$ Different documentation types required

Documentation in FOP

Documentation use cases in FOP

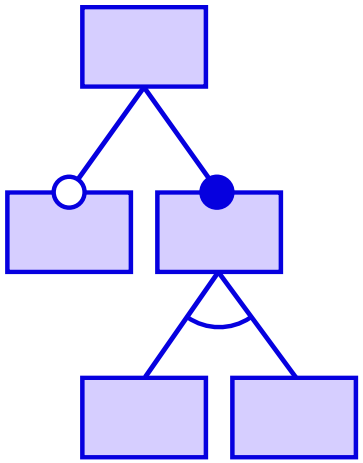
- Several developers are addressed

$\Rightarrow$ Different viewpoints on a software product line (SPL)

$\Rightarrow$ Different documentation types required

Documentation types

- Product
- Feature
- Meta
- Context interface



Documentation in FOP

Documentation use cases in FOP

- Several developers are addressed

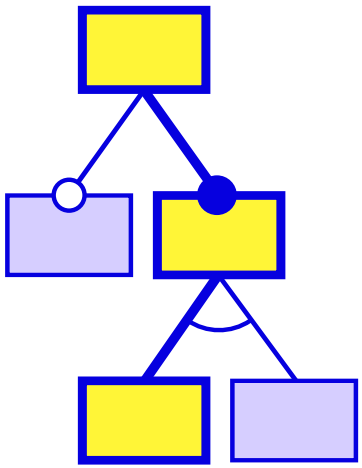
⇒ Different viewpoints on a software product line (SPL)

⇒ Different documentation types required

Documentation types

- **Product**
  - Documentation for a single product

  ⇒ Using an SPL's product in other projects
- Feature
- Meta
- Context interface



Documentation in FOP

Documentation use cases in FOP

- Several developers are addressed

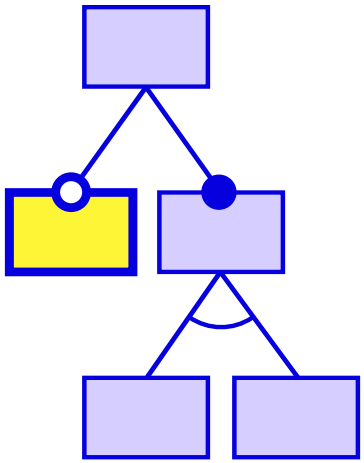
⇒ Different viewpoints on a software product line (SPL)

⇒ Different documentation types required

Documentation types

- Product
- **Feature**
  - Documentation for a single feature

  ⇒ Reusing a feature
- Meta
- Context interface



Documentation in FOP

Documentation use cases in FOP

- Several developers are addressed

⇒ Different viewpoints on a software product line (SPL)

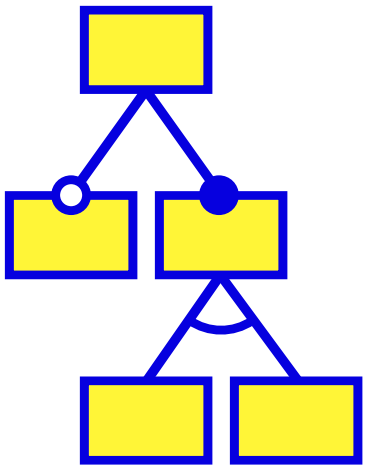
⇒ Different documentation types required

Documentation types

- Product
- Feature
- **Meta**
  - Documentation for a whole SPL

  ⇒ Implementing / Maintaining an SPL

  ⇒ Reusing an SPL (e.g., for multi product lines)
- Context interface



Documentation in FOP

Documentation use cases in FOP

- Several developers are addressed

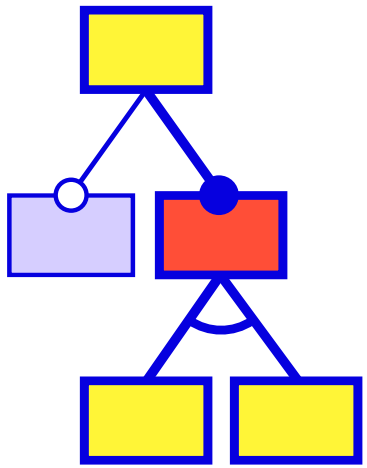
⇒ Different viewpoints on a software product line (SPL)

⇒ Different documentation types required

Documentation types

- Product
- Feature
- Meta
- **Context interface**
  - Documentation for a certain context

  ⇒ Implementing / Maintaining an SPL



Documentation in FOP

Documentation use cases in FOP

- Several developers are addressed

$\Rightarrow$ Different viewpoints on a software product line (SPL)

$\Rightarrow$ Different documentation types required

Documentation types

- Product
- Feature
- Meta
- Context interface

How to generate a documentation for one documentation type?

Use usual documentation approaches?

Straightforward Approaches

Feature documentation - Features contain all information

Chat

```
1  /**
2   * Sends a message to the server.
3   * Creates a new {@link TextMessage} and
4   * sends it to the server.
5   * @param line the message content.
6   *   The message may contain any character.
7   */
8  public static void send(String line) { ... }
9
```

Commands

```
10 /**
11  * Sends a message to the server.
12  * Can be used to trigger user commands.
13  * @param line the message content.
14  *   If the message starts with a /, the
15  *   whole line is interpreted as user command.
16  */
17 public static void send(String line) { ... }
```

Straightforward Approaches

Chat

```
1  /**
2   * Sends a message to the server.
3   * Creates a new {@link TextMessage} and
4   * sends it to the server.
5   * @param line the message content.
6   *   The message may contain any character.
7   */
8  public static void send(String line) { ... }
9
```

Commands

```
10 /**
11  * Sends a message to the server.
12  * Can be used to trigger user commands.
13  * @param line the message content.
14  *   If the message starts with a /, the
15  *   whole line is interpreted as user command.
16  */
17 public static void send(String line) { ... }
```

⇒ *Problem*: Redundant information

Straightforward Approaches

Chat

```
1  /**
2   * Sends a message to the server.
3   * Creates a new {@link TextMessage} and
4   * sends it to the server.
5   * @param line the message content.
6   *     The message may contain any character.
7   */
8  public static void send(String line) { ... }
9
```

Commands

```
10 /**
11  * Sends a message to the server.
12  * Can be used to trigger user commands.
13  * @param line the message content.
14  *     If the message starts with a /, the
15  *     whole line is interpreted as user command.
16  */
17 public static void send(String line) { ... }
```

⇒ *Problem*: Inconsistent information

Flexible Documentation Approach

New information structuring

- Distinction of information type
  - **Feature-independent** or **feature-specific** information

  ⇒ Reduces redundancy

  ⇒ Enables different handling of informations
- Assignment of **priorities**
  - Information with low priorities can be overridden

  ⇒ Helps to avoid conflicts in the comments
- Realization as new keywords

Flexible Documentation Approach

New information structuring

- Distinction of information type
  - **Feature-independent** or **feature-specific** information

  ⇒ Reduces redundancy

  ⇒ Enables different handling of informations
- Assignment of **priorities**
  - Information with low priorities can be overridden

  ⇒ Helps to avoid conflicts in the comments
- Realization as new keywords

Algorithm to generate documentation types on demand

- Uses adapted syntax as input

Flexible Documentation Approach

Adapted Javadoc syntax - Example

Chat

```
1  /**{@general 0}
2   * Sends a message to the server.
3   * @param line the message content.
4   */
5  /**{@feature 0}
6   * Creates a new {@link TextMessage} and
7   * sends it to the server.
8   * @param line The message may contain any character.
9   */
10 public static void send(String line) { ... }
```

Flexible Documentation Approach

Adapted Javadoc syntax - Example

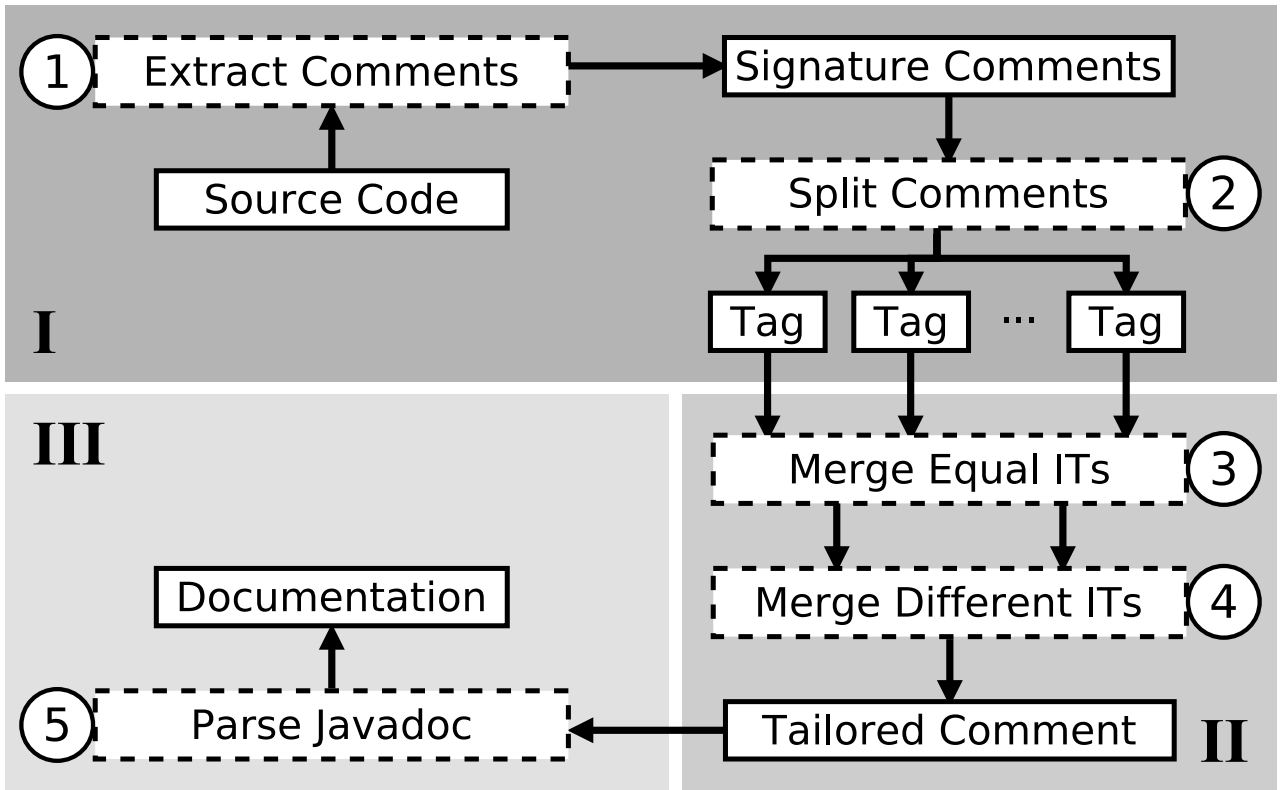
```
1  /**{@general 0}                                          Chat
2   * Sends a message to the server.
3   * @param line the message content.
4   */
5  /**{@feature 0}
6   * Creates a new {@link TextMessage} and
7   * sends it to the server.
8   * @param line The message may contain any character.
9   */
10 public static void send(String line) { ... }
11 /**{@feature 0}                                          Commands
12  * Can be used to trigger user commands.
13  */
14 /**{@feature 1}
15  * @param line If the message starts with a /, the
16  *   whole line is interpreted as user command.
17  */
18 public static void send(String line) { ... }
```

Generation Algorithm

Input: Developer specifies desired documentation type

⇒ Algorithm determines corresponding feature and signature set

Output: Documentation files (e.g., HTML)



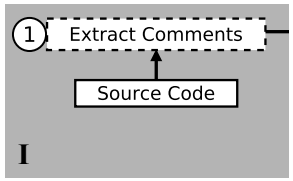
Generation Algorithm

Example for product documentation
(product contains both features)

```
1  /**{@general 0}                                          Chat
2   * Sends a message to the server.
3   * @param line the message content.
4   */
5  /**{@feature 0}
6   * Creates a new {@link TextMessage} and
7   * sends it to the server.
8   * @param line The message may contain any character.
9   */
10 public static void send(String line) { ... }
11 /**{@feature 0}                                          Commands
12  * Can be used to trigger user commands.
13  */
14 /**{@feature 1}
15  * @param line If the message starts with a /, the
16  *   whole line is interpreted as user command.
17  */
18 public static void send(String line) { ... }
```

Generation Algorithm

1. Create a list of all comments for a signature



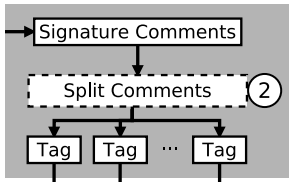
Chat

```
1  /**{@general 0}
2   * Sends a message to the server.
3   * @param line the message content.
4   */
5  /**{@feature 0}
6   * Creates a new {@link TextMessage} and
7   * sends it to the server.
8   * @param line The message may contain any character.
9   */
10 public static void send(String line) { ... }
```

Commands

```
11 /**{@feature 0}
12  * Can be used to trigger user commands.
13  */
14 /**{@feature 1}
15  * @param line If the message starts with a /, the
16  *   whole line is interpreted as user command.
17  */
18 public static void send(String line) { ... }
```

Generation Algorithm



2. Splits comments into single tags

Chat

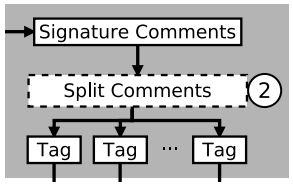
```
1  /**{@general 0}
2   * Sends a message to the server.
3   * @param line the message content.
4   */
5  /**{@feature 0}
6   * Creates a new {@link TextMessage} and
7   * sends it to the server.
8   * @param line The message may contain any character.
9   */
10 public static void send(String line) { ... }
```

Commands

```
11 /**{@feature 0}
12  * Can be used to trigger user commands.
13  */
14 /**{@feature 1}
15  * @param line If the message starts with a /, the
16  *   whole line is interpreted as user command.
17  */
18 public static void send(String line) { ... }
```

Generation Algorithm

3. Merge tags with *equal* information types

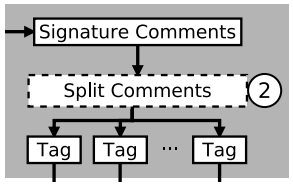


Chat

```
1  /**{@general 0}
2   * Sends a message to the server.
3   * @param line the message content.
4   */
5  /**{@feature 0}
6   * Creates a new {@link TextMessage} and
7   * sends it to the server.
8   * @param line The message may contain any character.
9   */
10 public static void send(String line) { ... }
11 /**{@feature 0}
12  * Can be used to trigger user commands.
13  */
14 /**{@feature 1}
15  * @param line If the message starts with a /, the
16  *   whole line is interpreted as user command.
17  */
18 public static void send(String line) { ... }
```

Commands

Generation Algorithm



3. Merge tags with *equal* information types

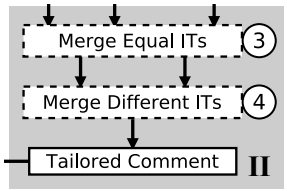
Chat

```
1  /**{@general 0}
2   * Sends a message to the server.
3   * @param line the message content.
4   */
5  /**{@feature 0}
6   * Creates a new {@link TextMessage} and
7   * sends it to the server.
8   * @param line The message may contain any character.
9   */
10 public static void send(String line) { ... }
```

Commands

```
11 /**{@feature 0}
12  * Can be used to trigger user commands.
13  */
14 /**{@feature 1}
15  * @param line If the message starts with a /, the
16  *   whole line is interpreted as user command.
17  */
18 public static void send(String line) { ... }
```

Generation Algorithm

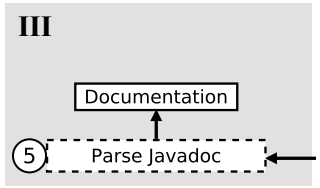


4. Merge tags with *different* information types

```
1  /**{@general 0}
2   * Sends a message to the server.
3   * @param line the message content.
4   */
5  /**{@feature 0}
6   * Creates a new {@link TextMessage} and
7   * sends it to the server.
8   * Can be used to trigger user commands.
9   * @param line If the message starts with a /, the
10  *   whole line is interpreted as user command.
11  */
12 public static void send(String line) { ... }
```

Generation Algorithm

5. Parse comments with Javadoc tool




```
1  /**
2   * Sends a message to the server.
3   * Creates a new {@link TextMessage} and
4   * sends it to the server.
5   * Can be used to trigger user commands.
6   * @param line the message content.
7   *    If the message starts with a /, the
8   *    whole line is interpreted as user command.
9   */
10 public static void send(String line) { ... }
```

Evaluation

Evaluation Objects

- Prototype for FeatureHouse[1] product lines integrated in FeatureIDE[2]
- Two example product lines **Chat** and **Snake**
- Comparison of the **input size** for each documentation type (Measured in number of characters)

Evaluation Procedure

- Generate input for straightforward approaches
- Compare to our input for the documentation type

[1] http://www.infosun.fim.uni-passau.de/spl/apel/fh/
[2] http://fosd.de/fide

Evaluation

Input - Our Method

```
1  /**{@general 0}                                          Chat
2   * Sends a message to the server.
3   * @param line the message content.
4   */
5  /**{@feature 0}
6   * Creates a new {@link TextMessage} and
7   * sends it to the server.
8   * @param line The message may contain any character.
9   */
10 public static void send(String line) { ... }
11 /**{@feature 0}                                          Commands
12  *   Can be used to trigger user commands.
13  */
14 /**{@feature 1}
15  *   @param line If the message starts with a /, the
16  *   whole line is interpreted as user command.
17  */
18 public static void send(String line) { ... }
```

Input - Straightforward Approaches
(*feature documentation*)

Chat

```
1  /**
2   * Sends a message to the server.
3   * Creates a new {@link TextMessage} and
4   * sends it to the server.
5   * @param line the message content.
6   *   The message may contain any character.
7   */
8  public static void send(String line) { ... }
9
```

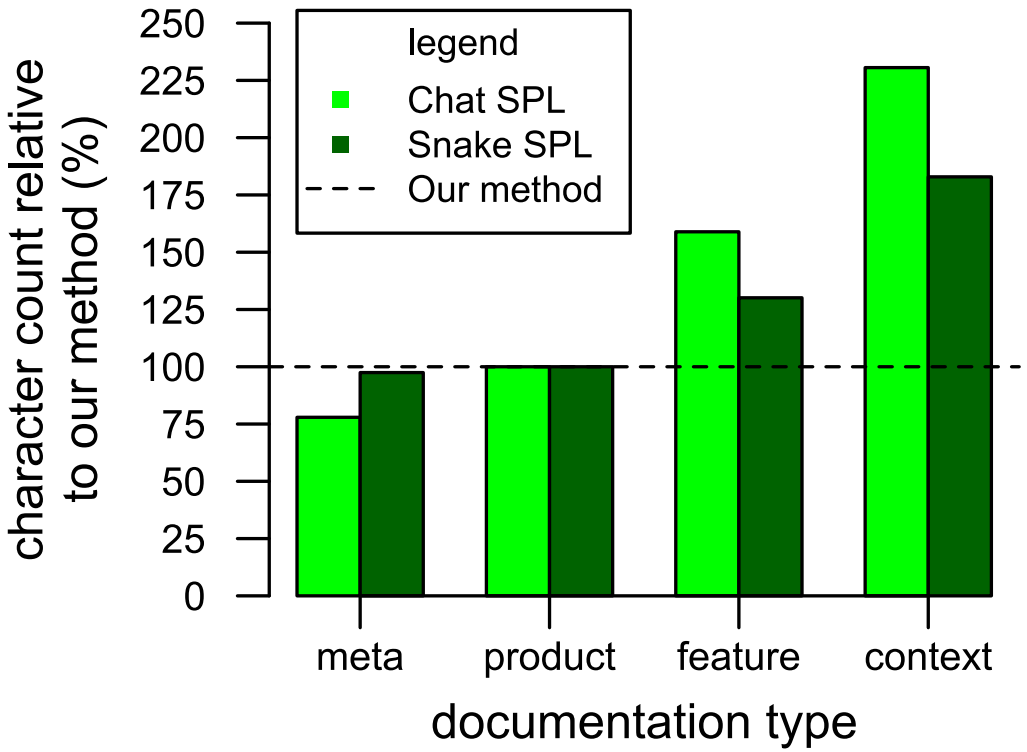
Commands

```
10 /**
11  * Sends a message to the server.
12  * Can be used to trigger user commands.
13  * @param line the message content.
14  *   If the message starts with a /, the
15  *   whole line is interpreted as user command.
16  */
17 public static void send(String line) { ... }
```

Evaluation Results

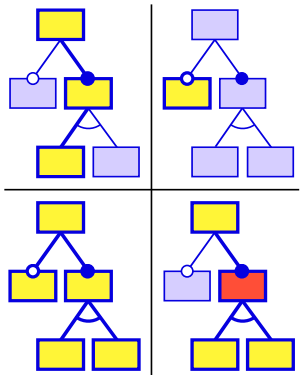
Input size of straightforward approaches (relative to our method)

Type	Chat	Snake
Meta	78%	97%
Product	∼100%	∼100%
Feature	157%	130%
Context	227%	182%
Sum	563%	510%



Contribution

Identification of four different **documentation types**

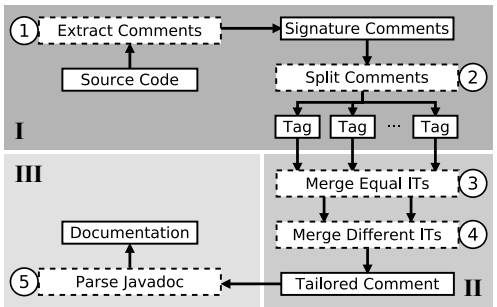


Introduced **new Javadoc syntax** to structure information

- Efficiency tested with evaluation

⇒ Especially useful when generating more than one documentation type

Developed algorithm to **generate each documentation type**



Future work

- Support for other SPL techniques
- Tool support for SPL developers