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

Sebastian Krieter, Reimar Schröter, Wolfram Fenske, Gunter Saake
University of Magdeburg, Germany
Feature-Oriented Programming (FOP) - An example

```java
// original method definition
public static void send(String line) {
    sendObject(toTextMessage(line));
}

// method refinement
public static void send(String line) {
    if (line.startsWith("/")) {
        // interpret command ...
    } else {
        original(line);
    }
}
```
Feature-Oriented Programming (FOP)

• Generate products by composing different sets of features

⇒ 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
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 use cases in FOP

- Several developers are addressed
  ⇒ Different viewpoints on a software product line (SPL)
  ⇒ Different documentation types required
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 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 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 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 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 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

How to generate a documentation for one documentation type?
Use usual documentation approaches?
**Feature documentation** - Features contain all information

```java
/**
 * Sends a message to the server.
 * Creates a new `{@link TextMessage}` and
 * sends it to the server.
 * @param line the message content.
 * The message may contain any character.
 */

public static void send(String line) {
    ...
}

/**
 * Sends a message to the server.
 * Can be used to trigger user commands.
 * @param line the message content.
 * If the message starts with a `/`, the
 * whole line is interpreted as user command.
 */

public static void send(String line) {
    ...
}
```
Straightforward Approaches

```java
/**
 * Sends a message to the server.
 * Creates a new {@link TextMessage} and sends it to the server.
 * @param line the message content.
 * The message may contain any character.
 */

public static void send(String line) {
    ...}
```

⇒ Problem: Redundant information
Straightforward Approaches

```java
/**
 * Sends a message to the server.
 * Creates a new {@link TextMessage} and sends it to the server.
 * @param line the message content.
 * The message may contain any character.
 */

public static void send(String line) {
}
```

- **Chat**

```java
/**
 * Sends a message to the server.
 * Can be used to trigger user commands.
 * @param line the message content.
 * If the message starts with a /, the whole line is interpreted as user command.
 */

public static void send(String line) {
}
```

- **Commands**

⇒ 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
Adapted Javadoc syntax - Example

```java
/**\{@general 0\} 
 * Sends a message to the server.
 * @param line the message content.
 */

/**\{@feature 0\} 
 * Creates a new \{@link TextMessage\} and sends it to the server.
 * @param line The message may contain any character.
 */

public static void send(String line) {
    ... 
}
```
Adapted Javadoc syntax - Example

```java
/**{@general 0}
 * Sends a message to the server.
 * @param line the message content.
 */
/**{@feature 0}
 * Creates a new {@link TextMessage} and sends it to the server.
 * @param line The message may contain any character.
 */
public static void send(String line) { ... }
/**{@feature 0}
 * Can be used to trigger user commands.
 */
/**{@feature 1}
 * @param line If the message starts with a /, the whole line is interpreted as user command.
 */
public static void send(String line) { ... }
```
**Input:** Developer specifies desired documentation type

⇒ Algorithm determines corresponding feature and signature set

**Output:** Documentation files (e.g., HTML)
Example for product documentation  
(product contains both features)

```java
/**{@general 0} */
* Sends a message to the server.
* @param line the message content.
*/

/**{@feature 0} */
* Creates a new {@link TextMessage} and
* sends it to the server.
* @param line The message may contain any character.
*/

public static void send(String line) {
    // ...
}

/**{@feature 0} */

/* Can be used to trigger user commands. */

/**{@feature 1} */
* @param line If the message starts with a /, the
* whole line is interpreted as user command.
*/

public static void send(String line) {
    // ...
}
```
1. Create a list of all comments for a signature

```java
/**{@general 0}
 * Sends a message to the server.
 * @param line the message content.
 */

/**{@feature 0}
 * Creates a new {@link TextMessage} and sends it to the server.
 * @param line The message may contain any character.
 */

public static void send(String line) {
    //...}
```

Chat

Commands

```java
/**{@feature 0}
 * Can be used to trigger user commands.
 */

/**{@feature 1}
 * @param line If the message starts with a /, the whole line is interpreted as user command.
 */

public static void send(String line) {
    //...}
```
2. Splits comments into single tags

```java
/**
 * Sends a message to the server.
 * @param line the message content.
 */

/**
 * Creates a new {@link TextMessage} and sends it to the server.
 * @param line The message may contain any character.
 */

public static void send(String line) {
    // ...
}

/**
 * Can be used to trigger user commands.
 */

/**
 * @param line If the message starts with a /, the whole line is interpreted as user command.
 */

public static void send(String line) {
    // ...
}
```

Chat

Commands
3. Merge tags with *equal* information types

```java
/**{@general 0}
 * Sends a message to the server.
 * @param line the message content.
 */

/**{@feature 0}
 * Creates a new {@link TextMessage} and sends it to the server.
 * @param line The message may contain any character.
 */

public static void send(String line) {
  ...
}

/**{@feature 0}
 * Can be used to trigger user commands.
 */

/**{@feature 1}
 * @param line If the message starts with a /, the
 * whole line is interpreted as user command.
 */

public static void send(String line) {
  ...
}
```
3. Merge tags with *equal* information types

```java
/** {@general 0} */
* Sends a message to the server.
* @param line the message content.
*/

/** {@feature 0} */
* Creates a new {@link TextMessage} and sends it to the server.
* @param line The message may contain any character.
*/

public static void send(String line) { ... }

/** {@feature 0} */
* Can be used to trigger user commands.
*/

/** {@feature 1} */
* @param line If the message starts with a /, the whole line is interpreted as user command.
*/

public static void send(String line) { ... }
```
4. Merge tags with different information types

```java
/**{@general 0}
 * Sends a message to the server.
 * @param line the message content.
 */

/**{@feature 0}
 * Creates a new {@link TextMessage} and sends it to the server.
 * Can be used to trigger user commands.
 * @param line If the message starts with a /, the whole line is interpreted as user command.
 */

public static void send(String line) {
    ...}
```
5. Parse comments with Javadoc tool

```java
/**
 * Sends a message to the server.
 * Creates a new {@link TextMessage} and sends it to the server.
 * Can be used to trigger user commands.
 * @param line the message content.
 * If the message starts with a /, the whole line is interpreted as user command.
 */

public static void send(String line) {
    ...}
```
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
Input - Our Method

```java
/**{ @general 0 }
 * Sends a message to the server.
 * @param line the message content.
 */

/**{ @feature 0 }
 * Creates a new { @link TextMessage } and
 * sends it to the server.
 * @param line The message may contain any character.
 */

public static void send(String line) {

/**{ @feature 0 }
 * Can be used to trigger user commands.
 */

/**{ @feature 1 }
 * @param line If the message starts with a /, the
 * whole line is interpreted as user command.
 */

public static void send(String line) {
```

Chat

Commands
**Input - Straightforward Approaches**

*(feature documentation)*

```java
/**
 * Sends a message to the server.
 * Creates a new {@link TextMessage} and sends it to the server.
 * @param line the message content.
 * The message may contain any character.
 */
public static void send(String line) {
  ...
}

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

*Chat*

*Commands*
### Input size of straightforward approaches (relative to our method)

<table>
<thead>
<tr>
<th>Type</th>
<th>Chat</th>
<th>Snake</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meta</td>
<td>78%</td>
<td>97%</td>
</tr>
<tr>
<td>Product</td>
<td>~100%</td>
<td>~100%</td>
</tr>
<tr>
<td>Feature</td>
<td>157%</td>
<td>130%</td>
</tr>
<tr>
<td>Context</td>
<td>227%</td>
<td>182%</td>
</tr>
<tr>
<td>Sum</td>
<td>563%</td>
<td>510%</td>
</tr>
</tbody>
</table>

**Legend**
- **Chat SPL**
- **Snake SPL**
- **Our method**

![Bar chart showing character count relative to our method for different documentation types](chart.png)
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