Документирование кода и Doxygen

Дмитрий Джус

АК-101

27 апреля 2011

План

Зачем документировать код

- Код приходится читать гораздо чаще, чем писать
- Работа в коллективе предусматривает отсутствие скрытых знаний
- Программируем осмысленно
- Помимо сопроводительной записки к библиотеке, простейшая документация — комментарии в коде

Основные принципы

- Документация и код — разные представления одной сущности
- Не повторяйте очевидное.
  Бесполезные комментарии только вредят:

```
/// Increase i
i++;
```

- Важно поддерживать актуальность описаний
- У хорошего кода хорошая документация

Жизнь и смерть

- Код живёт, пока с ним кто-то может работать
- Как работать с непонятным кодом?

Простой текст

- Рукописные заметки в сторонних текстовых файлах (`README`, `/usr/src/linux/Documentation`)
- + Просто и быстро писать
- – Отсутствие структурирования
- – Может устареть

Структурные форматы

- Какой-либо специальный формат для описания документации (LaTeX, DocBook, Texinfo и другие)
- Используются и заточенные под конкретный проект XML-языки (Gentoo Guide-XML)
- Экспорт в разные форматы представления (HTML, PDF, info и прочие)
- + Структурность
- – Может устареть
- – Неудобно писать

Документация из комментариев

- Всё началось с Javadoc, теперь есть Doxygen, gtk-doc и другие
- Основная идея: внешняя программа обрабатывает исходные тексты программы, генерируя по ним документацию
- В комментариях помимо обычного текста используются конструкции со специальным значением
- + Пишем код и документацию одновременно
- + Обработка автоматизирована
- + Поддерживаются разные языки программирования
- + Несколько форматов вывода: HTML, PDF, man, XML

Простой пример

- Doxygen извлекает информацию о функциях, классах и методах, а также комментарии вида

```
/// comment
/** comment */
```

```
/// Discrete Fourier transform for N points
void dft(float *x_real, float *x_imag, const int N);
```

Как пользоваться

- Создаём в корне проекта конфигурационный файл (`Doxyfile`)

```
doxygen -g
```

- Изменяем настройки в `Doxyfile`
- Запускаем doxygen

```
doxygen
```

- Смотрим получившуюся документацию

Ссылки

- Doxygen обрабатывает всё дерево проекта и может создавать перекрестные ссылки

```
/// Fast Fourier transform for N points
/// @see dft
void fft(float *x_real, float *x_imag, const int N);
```

Документация методов

- Ключевое слово `@param` задаёт описание аргумента метода

```
/// Fast Fourier transform for N points
///
/// @param x_real Real parts
/// @param x_image Imaginary parts
/// @param N Points in sample
void fft(float *x_real, float *x_imag, const int N);
```

- Ключевое слово `@return` ставится перед описанием возвращаемого значения, а `@throws` — перед вызываемым исключением

```
/// @return Amount of prime numbers in array
/// @throws PrimeCountException
int count_primes(int *n, const int len);
```

Списки задач

- Ключевые слова `@todo` и `@bug` позволяют отметить существующие проблемы в коде

```
/// @todo Rewrite using Miller–Rabin algorithm
int count_primes(int *n, const int len);
```

```
/// @bug Fails when no primes are present
int count_primes(int *n, const int len);
```

- Опции `GENERATE_TODOLIST` и `GENERATE_BUGLIST` управляют генерацией списков задач или проблем по коду

Генерация графов зависимостей

- Опции `HAVE_DOT`, `CLASS_GRAPH`, `CALL_GRAPH`, `CALLER_GRAPH`, `GRAPHICAL_HIERARCHY` управляют генерацией графов зависимостей между классами и вызовов функций
- Используется утилита `dot` из пакета Graphviz
- Ключевое слово @dot позволяет вставить вручную описание графа

Многие другие возможности

- Индексы по имеющимся в проекте методам, классам
- Форматирование текста документации (@b, @c, @e, а также подмножество HTML)
- HTML-шаблоны настраиваются
- Условная вставка документации в зависимости от выходного формата