Инструментальные средства информационных систем

Группа: БИН-19-1

Автор: Сафиуллина Н.Ф.

Лекция 7 Инструменты документирования

Содержание

Введение

Документация важна. В IT эта фраза кажется заезженной и есть сомневающиеся в ее правдивости или в ее релевантности в настоящее время.

Но даже сторонники Agile не отрицают важность документации, помните Agile ценности? *Что говорится там про документацию?* Они просто говорят, что работающий код важнее.

1 Документация

1.1 За и против

Документация бывает разных видов. Сторонники и противники документации возможно рассматривают разные виды документации.

Противники документации аргументируют свою позицию в основном следующим образом:

– чтобы вести документацию надо выбрать систему для ее ведения, а это не так просто, существует множество возникающих требований, таких как доступность через интернет, удобство добавления новых данных, наличие поиска и так далее;

– нужно будет описывать каждый процесс, т.е. время разработки каждой функциональной единицы кода будет как минимум в два раза дольше;

– при изменении функциональности какой-то части кода придется переписывать документацию, поддержание актуальности документации тоже трудоемкий процесс;

– есть же исходный код, зачем тратить время на документацию, можно прочитать как работает процесс по коду.

Доводы весомые и часто они берут верх и компании вовсе не ведут документацию. Считается что время разработки самый ключевой фактор. Бизнесу нужны разработки “ещё вчера”, так как нужно опередить конкурентов, поэтому этап документирования часто просто отбрасывается.

Сторонники документации приводят следующие доводы:

– имея документацию, легко обучить новичка (на что противники отвечают, что не легко, не легко прочитать документацию размером с “Войну и мир”, причем без всякого сюжета);

– имея документацию проще производить изменение какого-то процесса, понятно что, где и как нужно доработать, по документации видно как реализовано сейчас, “как работает сейчас”, и можно выстроить шаги, чтобы прийти к тому “как нужно чтобы работало”;

– документация поможет избежать изобретения и разработки того, что уже написано;

– игнорирование этапа документирования является “ловушкой будущего”, т.е. в будущем это окажет отрицательное влияние.

Как думаете какое влияние окажет в будущем отсутствие документации?

Во-первых, в будущем сейчас понятный код превращается в легаси код без документации. Который никто не понимает. Никто не хочет работать в компаниях с легаси кодом и без документации. (А устаревший стек технологий, если такой имеется, добавляет масло в огонь.)

Второй отрицательный эффект заключается в том, что доработки от пользователей и их технические задания будут выставлены в виде “хотелок”, “хотим, чтобы…”, так как пользователи не знают “как работает сейчас”, и это негде узнать, они не смогут описать что нужно поменять. Например, пользователь-бухгалтер видит сумму в итоге отчета такую-то, а должна быть другая (по его расчетам), он просто напишет: “ошибка в процессе, сумма должна быть другой”. Программист смотрит в код и код вполне логичен, а как получить сумму, которую хочет видеть пользователь, не понятно. Придется попросить пользователя описать, как он получил новое значение и, далее происходит сравнение кода программы и алгоритма предоставленного пользователем. Поиск отличий и поиск путей устранить отличия и написать новую реализацию, которая будет работать правильно. Все это отнимает много времени. При этом надо понять почему раньше было сделано иначе, и скорее всего, раньше было сделано тоже правильно, просто не были предусмотрены все случаи.

Третий отрицательный эффект – разработчики постоянно будут отвечать на вопросы пользователей, “как работает”, “как получился результат”, “какая логика процесса”, “почему так сделано”. Т.е. разработчики будут тратить время на объяснение, как работает программа. Потому что только они могут прочитать документацию в виде кода программы. А если писавший код уже уволился? И пришли новые люди. А если код написан без единой строчки комментария? Чтение чужого легаси кода превращается в трудное испытание. И в данной ситуации, у меня нет статистики, но очень часто код понимают неправильно и доработки ломают работающий процесс.

Очевидно, что документация нужна, вопрос другой: какая она должна быть? Ведь действительно, толстая книга с описанием всего всего не совсем подходит.

1.2 Виды документации

Первое что появляется в начале разработки – это техническое задание. Даже если это всего пара предложений, их нужно сохранить, как описание, что мы вообще делаем. Технические задания бывают разные, бывают полноценные проектные задания с описанием всех функциональных требований, диаграммами, планами выполнения, с описанием пользовательских интерфейсов и схем данных для баз данных. А бывает техническое задание в виде “хотелки”, а в процессе разработки задание начнёт обрастать требованиями, как это получается при применении Agile.

Итак первый вид документации, который надо сохранить:

1) Техническое задание

Конечно, в процессе разработки первоначальное техническое задание может поменяться. Иногда оно может так измениться, что в результате получится совсем другой продукт. Это не значит что сделано неправильно, просто вначале не было до конца ясно что нужно сделать. Все маленькие добавления к изначальному техническому заданию тоже нужно сохранить. Не требуется переписывать каждый раз исходное техзадание, новые требования можно сохранять списком пользовательских историй – backlog.

Т.е. второй вид документации, это перечень всех добавленных требований:

2) Пользовательские истории, users story.

Желательно присваивать каждой истории свой идентификатор. Чтобы можно было ссылаться на конкретную историю при объяснении почему сделано так и кто был заказчиком в комментариях кода или в описании алгоритмов работы.

Итак, у нас есть техническое задание, есть набор каких-то требований и историй, приступаем к написанию кода, появляется третий тип документации, комментарии в коде программы. В комментарии следует описывать не только что делает конструкция кода, но и почему.

Например,

```
если (валюта счета == валюта договора)
то: (провести транзакцию)
иначе: (отклонить транзакцию)
```

Выше мы видим пример комментария к коду, но по нему не понятно почему так. Следует дополнить комментарий описанием:

```
# соглашение №0102, мы не можем проводить операции в валюте
# отличающейся от валюты указанной в договоре с клиентом
если (валюта счета == валюта договора)
то: (провести транзакцию)
иначе: (отклонить транзакцию)
```

Хорошо будет еще добавить идентификатор пользовательской истории, в которой было указано добавить это условие.

Получается, третий вид документации:

3) Комментарии в коде

Параллельно разработке и написанию комментариев в коде можно создавать четвертый вид документации – сценарий. Это либо пользовательский сценарий, т.е. куда нажимать и что вводить, либо это может быть сценарий взаимодействия, если у нас не пользовательское приложение.

Алгоритмы расчета каких-то параметров могут поменяться, но сценарии работы меняются редко. Причем не будем путать с пользовательской документацией или руководством пользователя. Сценарий – это краткое описание, куда нажимать и что должно получиться, без описания внутренней логики.

Итак, четвертый вид документации:

4) Сценарии

Сценарии можно размещать в help для пользователя к элементам программы, в виде кратких справок, которые будут пояснять, что тут вообще происходит, что нужно сделать.

Выше мы упомянули руководство пользователя. И если два предыдущих вида документации создаются во время разработки, руководство пользователя пишется в конце. Иногда техническое задание по соглашению считается также руководством пользователя, т.е. что просили сделать, то и есть. Портит эту идеальную картину список историй, некоторые истории могут прямо противоречить исходному заданию. И всё-таки для экономии времени руководство пользователя можно составить из технического задания и историй.

Пятый вид документации:

5) Руководство пользователя

Сестра-близнец документации пользователя – это руководство разработчика.

На самом деле существует ГОСТ регулирующий виды документации, который называется ГОСТ 19 – ЕСПД (единая система программной документации), в нем примерно 32 пункта, т.е. описание 32 видов документации. В этом ГОСТе руководство программиста сопровождается также руководством системного программиста, руководством по техническому обслуживанию и еще множеством другой технической документации.

Для большинства компаний и их проектов достаточно будет хотя бы тех 7 видов, которые тут перечислены.

Итак, шестой вид документации:

6) Руководство разработчика

Руководство разработчика и руководство пользователя на самом деле похожи, в них одни и те же объекты, функции и процедуры рассматриваются с разных сторон. Например, в руководстве пользователя описана какая-то форма с полями, пользователю в руководстве поясняется, что существует проверка на введенные значения, нельзя указать валюту, которая не указана в договоре с этим пользователем. А в руководстве разработчика, тоже самое добавляется ссылкой на функцию проверки, указывается модуль, где находится форма и функция проверки, указывается имя функции проверки, описываются ее параметры и возвращаемые значения.

И наконец, нужен документ адресная книга. Допустим вводится какое-то новое решение. Презентация нового решения: “Ребята, мы написали телеграм-бота, которые позволяет нашим линейным сотрудникам написать нам напрямую заявку на ремонт оборудования и прикрепить фотографию”, бот реализует следующие алгоритмы и имеет следующие функциональные возможности. В презентации все прекрасно, остается один вопрос.

Какой вопрос, как думаете, если вы будете та команда, которая будет поддерживать работу бота?

Где все это находится? Какой сервер осуществляет обслуживание бота? Где находится исходный код?

Т.е. нужно описание с адресами и ссылками, где посмотреть реализацию решения и где оно само работает. Также потребуются логины и пароли.

Седьмой вид документации:

7) Документ адресный справочник или маршрутизатор.

Кроме этого, как было сказано выше, существует еще множество возможных полезных документов и описаний, например, сопроводительная записка или набор тестовых данных. В документации должна быть нарисована концептуальная и функциональная модели.

Конкретная ситуация и наличие времени покажет какой набор описаний будет необходимо и достаточно сделать. Немаловажно, что документацию надо поддерживать в актуальном состоянии.

2 Различные инструменты документирования

(Написать об инструментах ведения документации, например: wiki, wordpress, bitrix, joomla, gitbook)

3 Документирование кода Python

Всегда приятно одним выстрелом убить двух зайцев, а в документации очень интересно воспользоваться возможностью из комментариев кода получить описание процессов. Комментарии в коде это минимум, что точно нужно будет делать, а использовать их для автоматической генерации документации вдвойне приятно.

Кроме того, если документация и код находятся рядом это очень удобно.

3.1 Doxygen

Согласно официальному сайту doxygen.

https://www.doxygen.nl/index.html

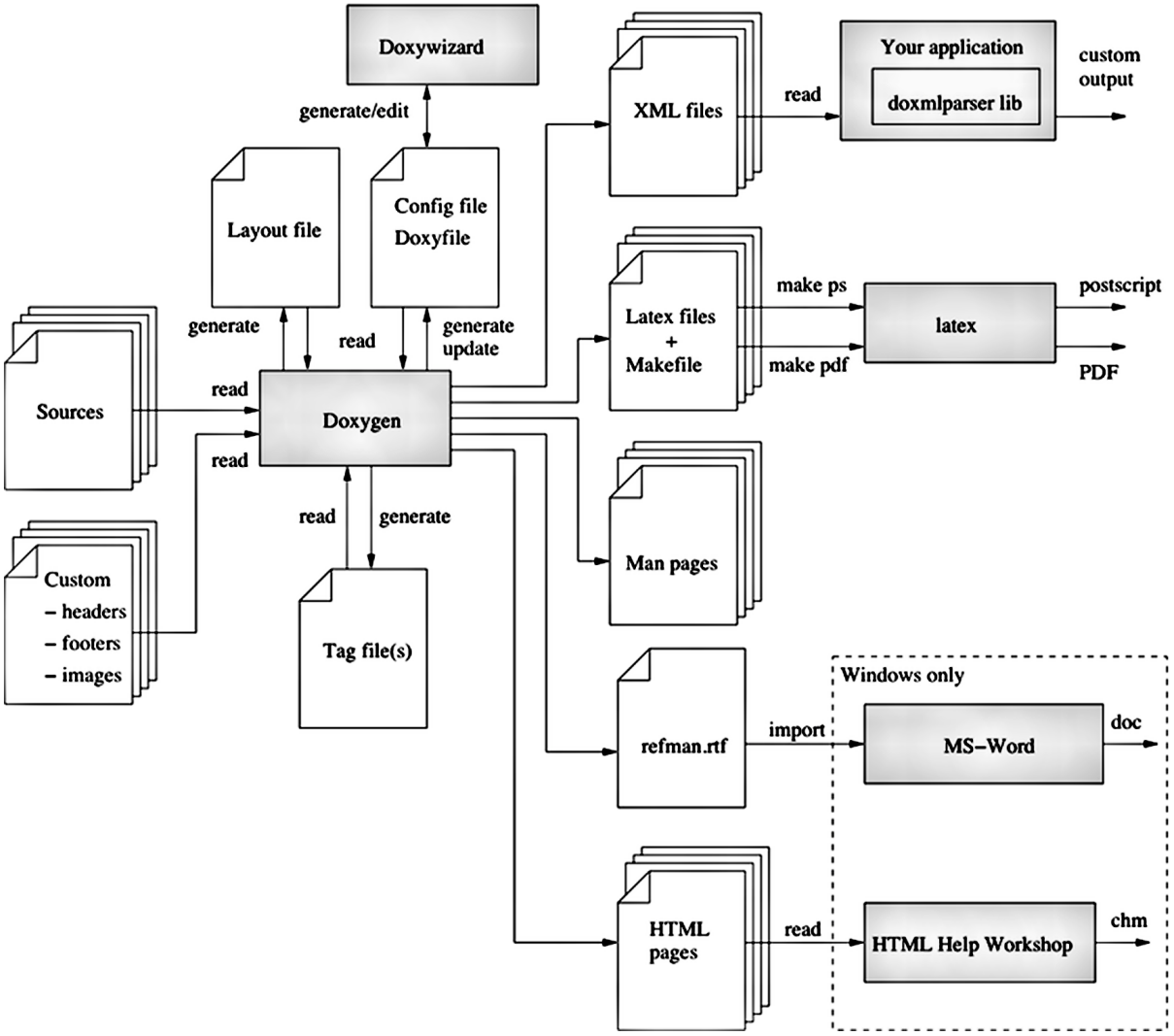
Doxygen — это стандартный инструмент для создания документации из аннотированных источников (т.е. имеющих описание, комментарии) C++, но он также поддерживает другие популярные языки программирования, такие как C, Objective-C, C#, PHP, Java, Python, IDL (разновидности Corba, Microsoft и UNO/OpenOffice).), Fortran и в некоторой степени D. Doxygen также поддерживает язык описания оборудования VHDL.

Можно скачать инсталлятор бинарного пакета для Windows. В установке ничего менять не придется, она простая. Можно скачать бинарный пакет без установщика, в архиве.

Исполняемый файл doxygen— это основная программа, которая анализирует исходный код и генерирует документацию.

Doxygen может помочь тремя способами:

1. Он может генерировать интерактивную документацию в формате HTML и/или автономное справочное руководство из набора исходных файлов с комментариями. Существует также поддержка создания документации в формате RTF (MS-Word), PostScript, PDF с гиперссылками.
2. Можно настроить doxygen для извлечения структуры кода из исходных файлов без комментариев. Это полезная функция для быстрого ориентирования в больших дистрибутивах. Doxygen также может визуализировать отношения между различными элементами с помощью графов зависимостей, диаграмм наследования и диаграмм взаимодействия, которые генерируются автоматически.
3. Можно использовать doxygen для создания обычной документации.



3.1.1 Настройка Doxygen

Распознавание документации в коде doxygen делает на основании данных заданных в конфигурационном файле.

После установки doxygen в меню появится приложение – doxywizard. Это приложение является графическим редактором файла конфигурации.

Файл конфигурации также можно создать из командной строки:

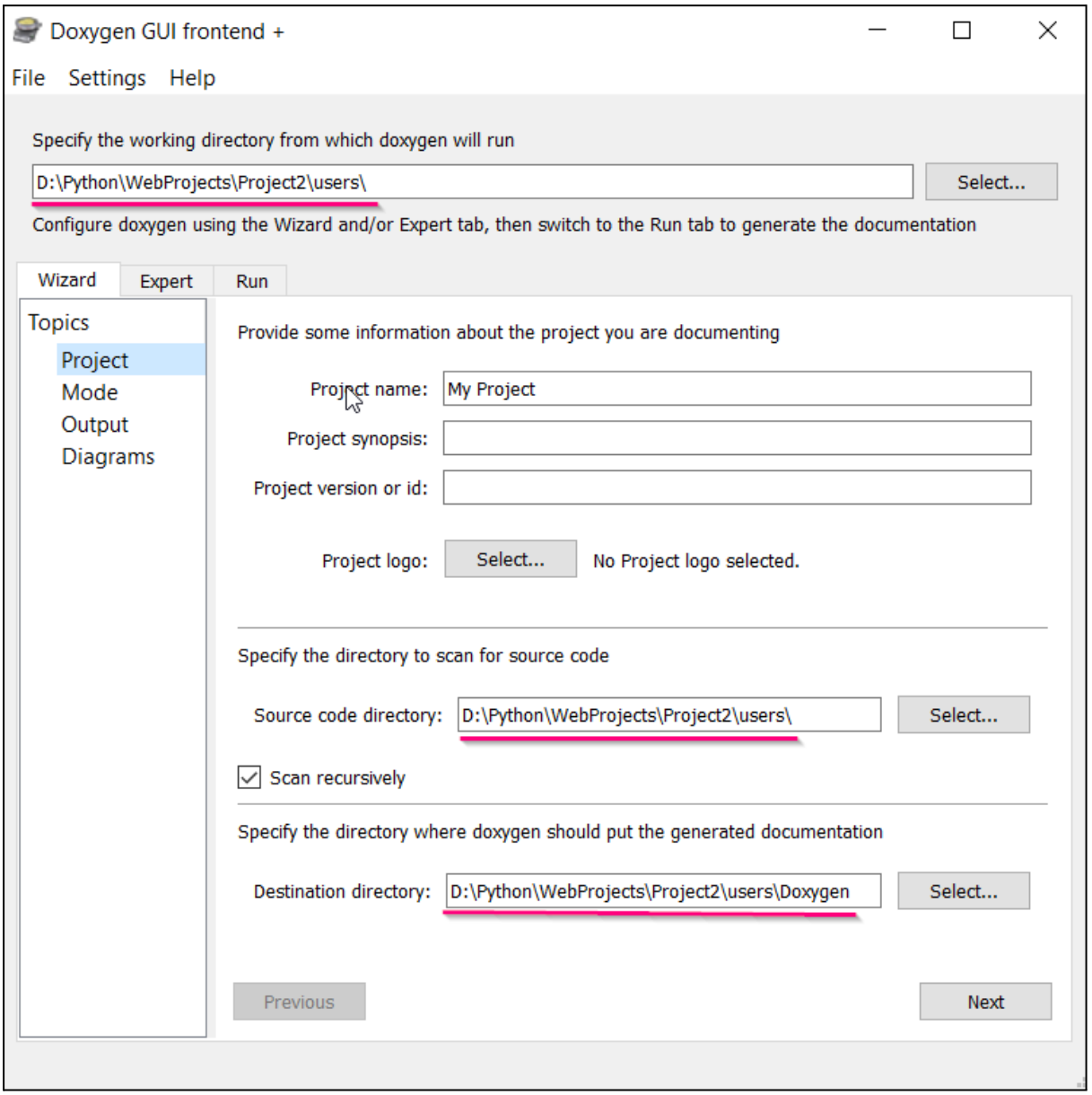
```
doxygen -g <файл конфигурации>
```

Но проще всего конечно сформировать файл конфигурации с помощью doxywizard. Для стандартной настройки, не в экспертном режиме, нужно пройти несколько шагов.

На первом шаге выбираем папку где лежит наш проект который надо задокументировать, это поле подписанное:

```
Specify the work directory from which doxygen will run
```

Поле Project name не имеет отношение к настоящему имени вашего проекта, это имя проекта документации, можно ввести любое. Оно будет написано на титульном листе документации, так что нужно выбрать такое имя, которое будет отображать сущность проекта.



Мы должны указать папку где лежит код:

`Source code directory`

Если код программы находится в дереве каталогов, то следует поставить галочку:

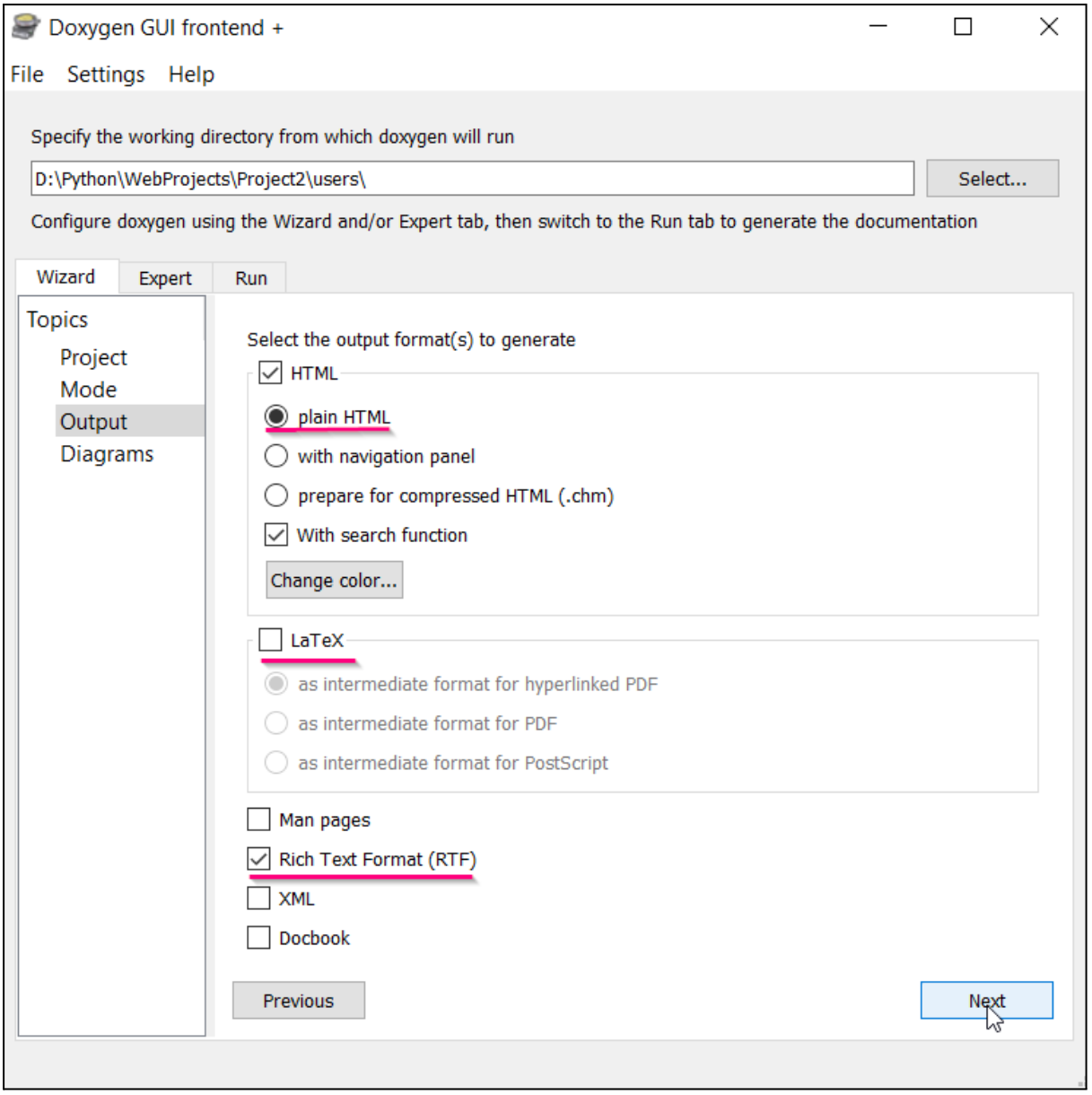
`Scan recursively`

И последнее нужно выбрать куда сохранить документацию в поле:

`Destination directory`

Второй шаг – оставляем все по умолчанию.

На третьем шаге надо выбрать формат будущей документации. Можно оставить предлагаемый формат HTML, отказаться от LaTeX формата и отметить галочкой, что нужен ещё формат RTF.



Таким образом мы должны получить и набор HTML ссылающихся друг на друга страниц и текстовый документ, который может прочитать текстовый редактор.

На четвертом шаге оставляем все, как есть. Мы попадем на вкладку Run. На это вкладке есть несколько полезных кнопок, например:

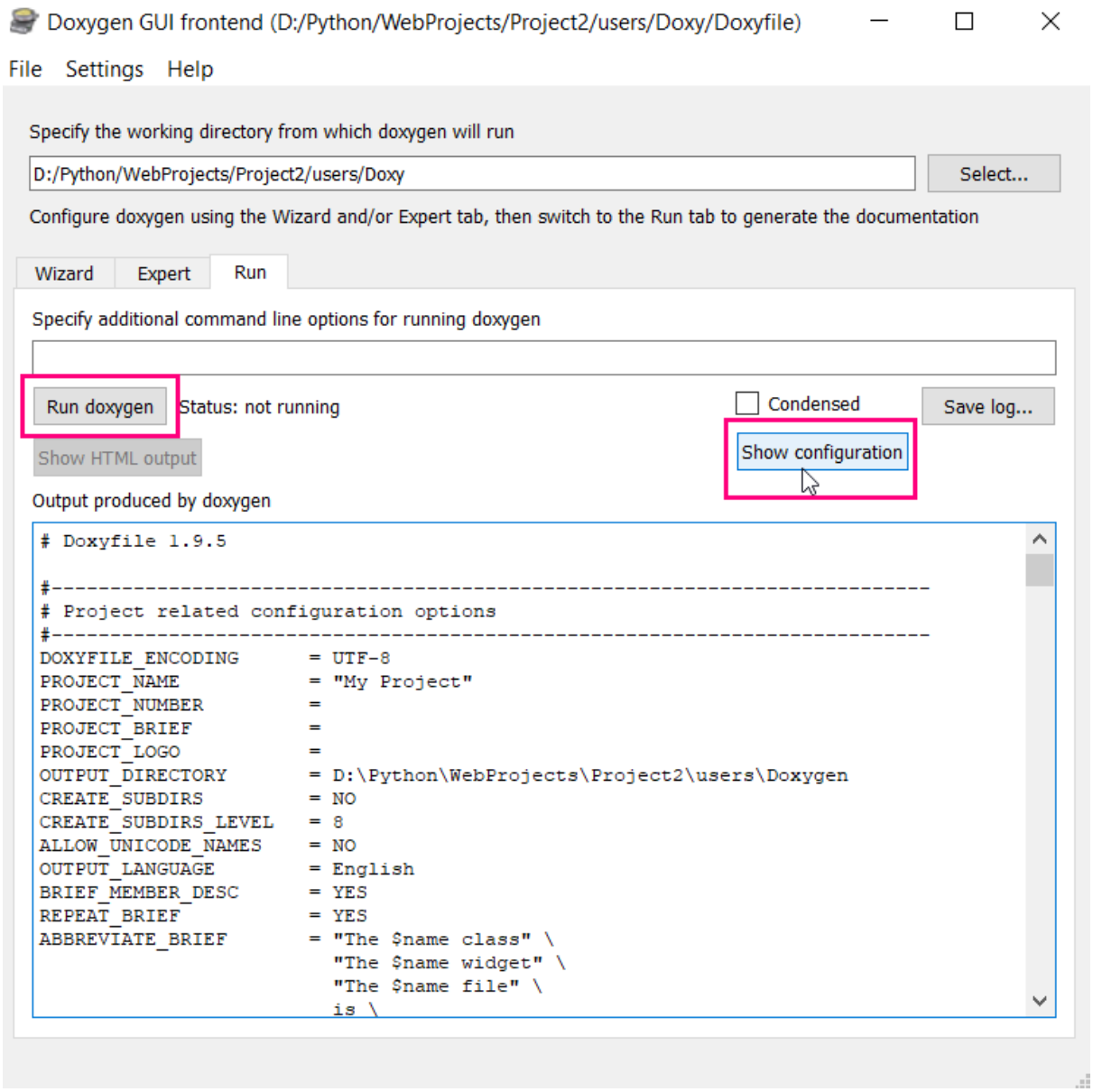
`Show configuration`

показывает параметры, которые у нас получились с помощью конструктора, эти параметры будут записаны в файл конфигурации.

Кнопка:

`Run doxygen`

запустить процесс создания документации.



Рассмотрим что представляет собой конфигурационный файл. Это обычный текстовый файл с директивами, густо снабженный комментариями к каждому параметру. Начинается файл с текста:

```
# Doxyfile 1.9.5
```

```
# This file describes the settings to be used by the documentation
system
# doxygen (www.doxygen.org) for a project.
#
# All text after a double hash (##) is considered a comment and is
placed in
# front of the TAG it is preceding.
#
# All text after a single hash (#) is considered a comment and
will be ignored.
# The format is:
# TAG = value [value, ...]
# For lists, items can also be appended using:
# TAG += value [value, ...]
# Values that contain spaces should be placed between quotes (\"
\").
```

где сказано, что этот файл описывает настройки, которые будут использоваться для документирования системы.

Если полистать файл, то можно найти полезные и понятные параметры.

Куда сохранить документацию:

```
OUTPUT_DIRECTORY = D:\Python\WebProjects\Project2\users\Doxygen
```

Использовать docstring Python:

```
PYTHON_DOCSTRING = YES
```

Docstring — это строковый литерал, который располагают сразу за объявлением модуля, функции, класса или метода. Существует множество инструментов автоматической генерации документации, которые используют Python docstring.

Каталог откуда взять исходные файлы проекта:

```
INPUT = D:\Python\WebProjects\Project2\users
```

Какие файлы использовать для поиска кода программы:

```
FILE_PATTERNS = *.c \ ...
                *.py \
                *.pyw \
```

Мы указывали, что документацию нужно создать в HTML страничках, и вот это отображено в конфигурации:

```
GENERATE_HTML = YES
HTML_OUTPUT            = html
HTML_FILE_EXTENSION    = .html
```

И также мы указывали, что нам нужен Rich Text Format файл:

```
GENERATE_RTF = YES
RTF_OUTPUT   = rtf
```

Если комментарии написаны на русском языке, то рекомендую открыть вкладку Expert (продвинутая настройка) и указать язык:

```
OUTPUT_LANGUAGE = Russian
```

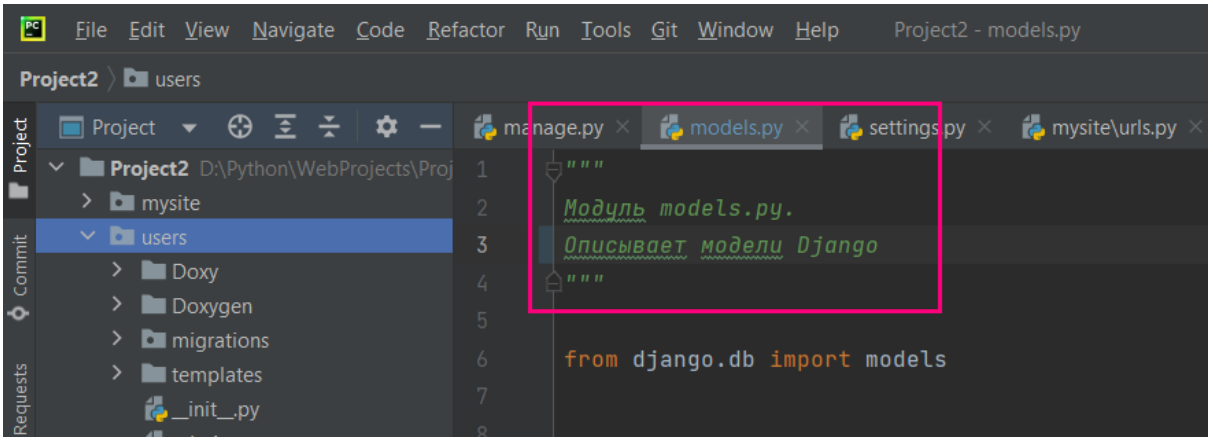
Остается убедиться соответствуют ли комментарии в тексте программы. Нужно создать docstring и подписать код. Рассмотрим что такое docstring.

3.1.2 Python Docstring

Рассмотрим что надо сделать в Python, чтобы Doxygen распознал ваши комментарии как документацию и составил описание.

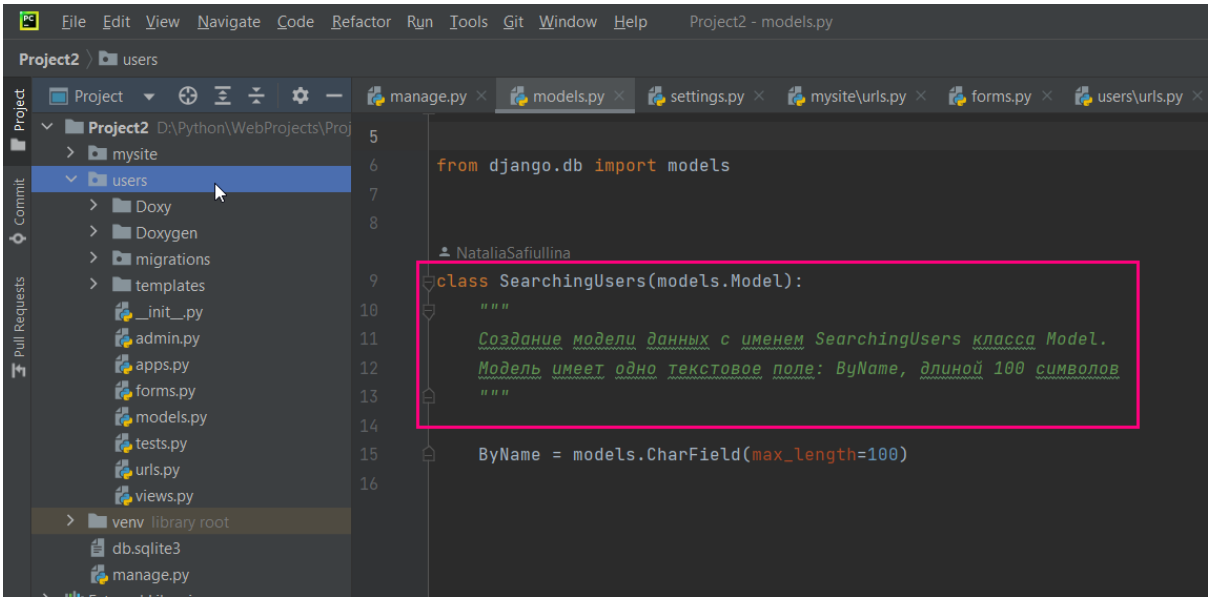
Как было сказано выше Docstring это строковый литерал, которые следует сразу за объявлением модуля, функции, класса или метода. Проще говоря это строка комментария.

Чтобы описать модуль напишите комментарий в начале модуля. Первой строкой без отступов. Можно использовать многострочный комментарий. Пример:



Текст описания может быть достаточно длинным.

Для описания класса следует поместить комментарий сразу за объявлением класса:



Например, весь модуль models.py выглядит следующим образом:

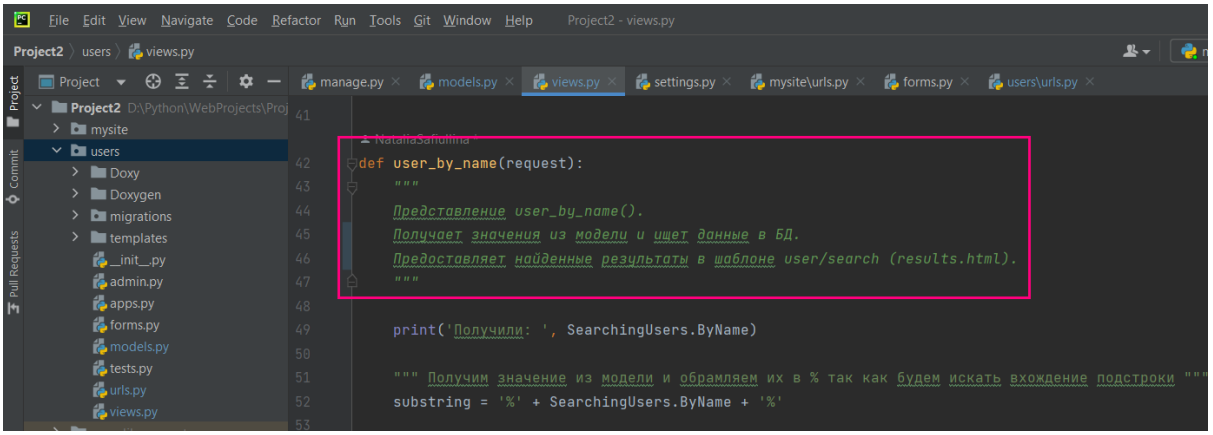
```
"""
Модуль models.py.
Описывает модели Django
"""
```

```
from django.db import models


class SearchingUsers(models.Model):
    """
    Создание модели данных с именем SearchingUsers класса Model.
    Модель имеет одно текстовое поле: ByName, длиной 100 символов
    """

    ByName = models.CharField(max_length=100)
```

Описание функции:




```
def user_by_name(request):
    """
    Представление user_by_name().
    Получает значения из модели и ищет данные в БД.
    Предоставляет найденные результаты в шаблоне user/search
(results.html).
    """
```

Таким образом нужно описать все модули, классы и функции приложения, затем можно запустить автоматическое создание документации.

Вместо “”” можно использовать ##, например:

```
## Модуль models.py.
# Описывает модели Django
```

3.1.3 Результат Doxygen

После того как мы нажмем кнопку Run doxygen, исходная папка будет отсканирована, указанные файлы с кодом разобраны и мы получим в указанной папке документацию в том виде, в каком выбирали.

Например мы выбирали html и rtf, и папка назначения была:

```
OUTPUT_DIRECTORY = D:\Python\WebProjects\Project2\users\Doxy
```

В результате работы Doxygen в этой папке создаст подпапки с именами HTML и RTF, и в них положит получившуюся документаци.

Чтобы запустить html документацию, нужно найти файл index.html.

(*Показать на телевизоре, что получилось.*)

3.2 Автодокументирование инструментами Python

На самом деле любой объект кода Python имеет атрибут __doc__, который Python хранит с другими прочими параметрами этого объекта. Если мы оформляем комментарии в виде Docstring, то этот атрибут будет заполнен этими комментариями. И мы можем получить справку об этом объекта.

```
print(user_by_name.__doc__)
```

Получим:

```
    Представление user_by_name().
    Получает значения из модели и ищет данные в БД.
    Предоставляет найденные результаты в шаблоне user/search
(results.html).
```

У Python есть библиотека для чтения документации по атрибутам объектов __doc__ – PyDoc.

Для создания документации по коду служит пакет PyMent. Первое что надо сделать это, конечно, установить пакет:

```
D:\Python\>pip install pyment
```

Затем чтобы создать документацию вызываем команду:

```
D:\Python\WebProjects\Project2\users>pyment views.py
```

В папке с программой появятся файлы с именами типа таких:

```
views.py.patch
```

Т.е. к имени модуля добавится patch. Файл будет иметь примерно следующий вид:

```
# Patch generated by Pyment v0.3.3

--- a/views.py
+++ b/views.py
@@ -12,8 +12,10 @@


 def get_name(request):
Представление get_name(). Отрисовывает форму и сохраняет
введенные значения в модель.
+
+    :param request:
+

     print('Получен метод запроса: ', request.method)
```

Видно, что pyment извлекает название функции и атрибут __doc__ этой функции и сохраняет в файл. Все остальные параметры и возможности pyment можно узнать из справки.

3.3 Другие инструменты автодокументирования

https://wiki.python.org/moin/DocumentationTools

- pdoc, a simple Python 3 command line tool and library to auto-generate API documentation for Python modules. Supports Numpydoc / Google-style docstrings, doctests, reST directives, PEP 484 type annotations, custom templates ...
- pdoc3, a fork of pdoc for Python 3 with support for Numpydoc / Google-style docstrings, doctests, LaTeX math, reST directives, PEP 484 type annotations, custom templates ...
- PyDoc, a documentation browser (in HTML) and/or an off-line reference manual. Also in the standard library as pydoc.
- pydoctor, a replacement for now inactive Epydoc, born for the needs of Twisted project.