МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
ДОНЕЦКОЙ НАРОДНОЙ РЕСПУБЛИКИ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к лабораторным занятиям по дисциплине «Профессиональная практика программной инженерии»

для обучающихся по
направлению подготовки 09.03.04 «Программная инженерия»
профиль «Инженерия программного обеспечения» всех форм обучения

Донецк
2021

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
ДОНЕЦКОЙ НАРОДНОЙ РЕСПУБЛИКИ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

КАФЕДРА «ПРОГРАММНАЯ ИНЖЕНЕРИЯ ИМ. Л. П. ФЕЛЬДМАНА»

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ
к лабораторным занятиям по дисциплине «Профессиональная практика программной инженерии»

для обучающихся по
направлению подготовки 09.03.04 «Программная инженерия»
профиль «Инженерия программного обеспечения» всех форм обучения

РАССМОТРЕНО
на заседании кафедры
программной инженерии им.
Л.П. Фельдмана
Протокол № 3 от 18.11.2021 г.

УТВЕРЖДЕНО
на заседании Учебно-
издательского совета ДОННТУ
Протокол № ___ от _____ г.

Донецк
2021

УДК 378.14:004.45(076)
ББК 74.58:32.973-018.2я73
М54

Составители:
Филипишин Дмитрий Александрович – ассистент кафедры программной инженерии имени Л.П. Фельдмана ГОУВПО «ДОННТУ».

М54 **Методические указания к выполнению лабораторных работ по дисциплине «Профессиональная практика программной инженерии»** : для обучающихся по направлению подготовки 09.03.04 «Программная инженерия» профиль «Инженерия программного обеспечения» всех форм обучения / ГОУВПО «ДОННТУ», Каф. программной инженерии им. Л.П. Фельдмана ; сост. Д. А. Филипишин. – Донецк : ДОННТУ, 2021. – Систем. требования: Acrobat Reader. – Загл. с титул. экрана.

Методические указания направлены на развитие и закрепление у студентов навыков применения системы контроля версий Git, создания и использования компьютерной документации в форматах CHM и HTA, а также создания самодокументирующегося кода (DocBlock, DocBook).

При разработке лабораторных работ 1, 2 и 4 были задействованы ранее подготовленные для этого курса материалы Грищенко Виктором Игоревичем кандидатом технических наук, доцентом кафедры «Программная инженерия».

УДК 378.14:004.45(076)
ББК 74.58:32.973-018.2я73

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цикл лабораторных работ ориентирован на освоение современных систем контроля версий – программное обеспечение для облегчения работы с изменяющейся информацией. В данной дисциплине сделан акцент на групповую разработку модулируемого проекта, изучение средств построения само документируемого программного кода и инструментов создания справочных руководств ориентированных на ОС Windows.

Цель дисциплины состоит в изучении наиболее популярных систем контроля версий и средств управления ими для применения в различных предметных областях.

В задачи дисциплины входит: работа с репозиториями; сохранение и восстановление данных после внесенных изменений; отслеживание изменений; управление процессом разработки относительно произведенных изменений; создание самодокументирующегося кода; создание электронных справочных руководств.

В результате выполнения лабораторных работ студент должен:

знать основные наиболее распространенные системы управления версиями;

уметь выбирать из них наиболее подходящую относительно задач требуемой предметной области и составлять самодокументирующиеся справочные руководства;

владеть базовыми инструментами управления системами контроля версий и создания справочных руководств.

В методических указаниях приведены темы для выполнения 6-ти лабораторных работ, предусмотренных рабочей программой дисциплины.

Ниже приведена таблица распределения тем лабораторных работ по академическим часам, а также ссылки на используемую литературу.

№ п/п	Тема занятия	Объём, час.	Литература
1	Базовые принципы работы с системами контроля версий	4	1,2
2	Работа с ветками	6	1-5
3	Восстановление данных из ранних ревизий	6	1-5
4	Создание самодокументирующегося кода	6	6,7
5	Разработка ручной документации	6	8
6	Веб-сайт из репозитория	6	9,10
Итого:		34	

Лабораторная работа 1

Базовые принципы работы с системами контроля версий

Цель работы: получить практические навыки использования систем контроля версий.

Контрольные вопросы

1. Системы контроля версий: предназначение и основные принципы.
2. Централизованные системы контроля версий.
3. Распределенные системы контроля версий.
4. Git: базовый цикл разработки.

Методические указания

Системы контроля версий стали неотъемлемой частью жизни не только разработчиков программного обеспечения, но и всех людей, столкнувшихся с проблемой управления интенсивно изменяющейся информации и желающих облегчить себе жизнь. Вследствие этого, появилось большое число различных продуктов, предлагающих широкие возможности и предоставляющих обширные инструменты для управления версиями.

Разрабатывать проект большого масштаба без мощной и надежной системы контроля версий – невозможно. Наиболее распространённой системой контроля версий является Git.

Git – это гибкая, распределенная (без единого сервера) система контроля версий, дающая массу возможностей не только разработчикам программных продуктов, но и писателям для изменения, дополнения и отслеживания изменения «рукописей» и сюжетных линий, и учителям для корректировки и развития курса лекций, и администраторам для ведения документации, и для многих других направлений, требующих управления историей изменений.

У каждого разработчика, использующего Git, есть свой локальный репозиторий, позволяющий локально управлять версиями. Затем, сохраненными в локальный репозиторий данными, можно обмениваться с другими пользователями.

Часто при работе с Git создают центральный репозиторий, с которым остальные разработчики синхронизируются. В этом случае все участники проекта ведут свои локальные разработки и беспрепятственно скачивают обновления из центрального репозитория. Когда необходимые работы отдельными участниками проекта выполнены и отлажены, они, после

удостоверения владельцем центрального репозитория в корректности и актуальности проделанной работы, загружают свои изменения в центральный репозиторий.

Наличие локальных репозиториев также значительно повышает надежность хранения данных, так как, если один из репозиториев выйдет из строя, данные могут быть легко восстановлены из других.

Работа над версиями проекта в Git может вестись в нескольких ветках, которые затем могут с легкостью полностью или частично объединяться, уничтожаться, откатываться и разрастаться во все новые и новые ветки проекта.

Репозиторий – хранилище версий – в нем хранятся все документы вместе с историей их изменения и другой служебной информацией.

Рабочая копия – копия проекта, связанная с репозиторием.

Классификация по способу хранения данных

Централизованные: одно хранилище всего проекта, каждый пользователь копирует себе необходимые ему файлы из этого репозитория, изменяет и, затем, добавляет свои изменения обратно.

Децентрализованные: у каждого пользователя свой вариант (возможно не один) репозитория, присутствует возможность добавлять и забирать изменения из любого репозитория.

Терминология

Метка – именованная версия проекта.

Ветка – метка, имеющая историю развития.

Коммит, фиксация – сохранение изменений в репозитории.

Чеккаут, извлечение – получение рабочей копии (без истории).

Push, pull – отправка/получение изменений в/из другого репозитория.

Задание к лабораторной работе

Создать в выделенном студенту репозитории три каталога: отчеты, программа, команда.

Составить по выбранной теме из предложенного списка перечень требуемых для нормального функционирования программы компонентов/модулей и описать их базовую функциональность в текстовом файле. Составить отчет по лабораторной работе и загрузить в репозиторий вместе с описанием проекта.

Список тем для разрабатываемого проекта:

1. Чат, с отзывом системы и просмотром сообщений с обеих сторон.
2. Форум с ветками-обсуждениями, с обязательной реализацией групповых ролей: гость, посетитель, модератор, администратор.
3. Блокнот, с реализацией поддержки семантики языков программирования, учесть наличие тем оформления.
4. Блокнот, с проверкой орфографии разговорных языков (не менее двух: английский, русский), учесть наличие тем оформления.
5. Заметки, с возможностью объединения их в книги, экспорт в pdf и назначения ярлыков для быстрого поиска по хранилищу.
6. Переводчик, с реализацией автоматического определения языка из доступных в системе. Учесть и описать особенности отображения клиентской части для максимального удобства пользователя.
7. Планировщик задач, с реализацией финансовых расчетов и возможностью просмотра истории, разделяемой по временным отрезкам.
8. Браузер, с реализацией не менее 20-ти имеющихся средств современных браузеров, например, Google Chrome, не ниже 49-го релиза.
9. Векторный редактор, с реализацией не менее 20-ти программных средств доступных в CorelDraw Graphics Suite не ниже 2021 версии.
10. Растровый редактор, с реализацией не менее 20-ти программных средств доступных в Adobe Photoshop не ниже версии CS3.
11. Генератор фракталов, с реализацией базовых алгоритмов построения фрактальных изображений (алгоритмические, геометрические и фракталы на основе метода IFS).
12. 3d редактор, с реализацией не менее 20-ти программных средств доступных в 3ds Max / Maya / Cinema 4D.
13. Кросс платформенный 3d редактор, с реализацией не менее 20-ти программных средств доступных в Blender, с описанием особенностей использования для каждой из указанных ОС.
14. Пасьянс, с возможностью игры по сети.
15. Тетрис, с возможностью игры по сети.
16. Эмулятор консольных игр (одной из dendy, sega megadrive, ps1/2/3/4, dreamcast), например, ePSXe, Gens, Snes.
17. Эмулятор портативных устройств (GameBoy, PSP, Pokemon Mini).
18. Конструктор видео игр, по аналогии с Unity.
19. Видео игра RPG, с реализацией механик, не менее 30-ти.
20. Видео игра шутер, с реализацией механик, не менее 30-ти.
21. Свой вариант программы (по согласованию с преподавателем).

Прядок выполнения работы

1. Зарегистрироваться на github.com.
2. Получить доступ к проекту у преподавателя.
3. Создать каталог проекта, в котором создать каталоги: отчеты, программа, команда.
4. Составить краткое описание разрабатываемых компонентов/модулей модулируемого проекта.
5. Составить отчет по выполнению лабораторной работы и загрузить его в репозиторий, в каталог «отчеты».
6. Загрузить в каталог «программа» описание компонентов/модулей разрабатываемого проекта.

Содержание отчёта

1. Титульный лист.
2. Краткое описание компонентов модулируемого проекта.
3. Описать шаги разработки программы и также загрузить в каталог «программа».
4. Вывод команды git log --pretty=format:\"%h %ad | %s%d [%an]\" --graph --date=short.
5. Вывод команды git diff для одной из ревизий.

Лабораторная работа 2

Работа с ветками

Цель работы: познакомиться с основами использования веток в системе контроля версий Git.

Контрольные вопросы

1. Ветвление в системах контроля версий.
2. Слияние веток.
3. Конфликты слияния в системах контроля версий.
4. Методы разрешения конфликтов.
5. Конфликты слияния в системах контроля версий.
6. Ветвление в системах контроля версий.

Методические указания

Можно сказать, что коммит это основной объект в любой системе управления версиями. В нем содержится описание тех изменений, которые вносит пользователь в код приложения. В Git коммит состоит из нескольких так называемых объектов. Для простоты понимания можно считать, что коммиты это односвязный список, состоящий из объектов в которых содержаться измененные файлы, и ссылка на предыдущий коммит.

У коммита есть и другие свойства. Например, дата коммита, автор, комментарий к коммиту и т.п. В качестве комментария обычно указывают те изменения, которые вносит этот коммит в код, или название задачи, которую он решает.

Git – это распределенная система управления версиями. Это значит, что у каждого участника проекта есть своя копия репозитория, которая находиться в папке ".git", которая расположена в корне проекта. Именно в этой папке хранятся все коммиты и другие объекты Git. Когда вы работаете с Git, он в свою очередь работает с этой папкой.

Завести новый репозиторий очень просто, это делается командой: git init.

Таким образом, у вас получится новый пустой репозиторий. Если вы хотите присоединиться к разработке уже имеющегося проекта, то вам нужно будет скопировать этот репозиторий в свою локальную папку с удаленного репозитория. Делается это командой: git clone <url удаленного репозитория>.

После чего в текущей папке появляется директория .git в которой и будет содержаться копия удаленного репозитория.

В целом работа с гитом выглядит так: меняем файлы в своей рабочей директории, затем добавляем эти изменения в staging area, используя команду: git add <имя файла>.

При этом можно использовать маски со звездочкой. Затем делаем коммит в своем локальном репозитории: git commit –m “Комментарий к коммиту”.

Когда коммитов накопится достаточно много, чтобы ими можно было поделиться, выполняем команду: git push. После чего коммиты уходят в удаленный репозиторий.

Если нужно получить изменения из удаленного репозитория, то нужно выполнить команду: git pull.

После этого, в нашем локальном репозитории появятся те изменения, которые были отправлены другими программистами.

Посмотреть коммиты можно при помощи команды: git log.

Формат ответа этой команды по дефолту не очень удобен. Вот такая команда выведет ответ в более читаемом виде: git log --pretty=format:"%H [%cd]: %an - %s" --graph --date=format:%c .

Чтобы закончить просмотр нужно нажать на клавишу q. Посмотреть, что находится в рабочей директории и staging area можно командой: git status.

Рабочую директорию можно переключить на предыдущее состояние, выполнив команду: git checkout <hash коммита>.

Пример выполнения

Программа пример «калькулятор».

Ранее описанные модули программы: клиентская часть, набор функционала для режимов «стандартный», «инженерный» и «программист».

Создаем для каждого модуля (режима) свою ветку.

Делаем несколько фиксаций по разработке программного кода в каждой из веток. Вливаем все ветки в master и фиксируем логи.

Задание к лабораторной работе

Разработать архитектуру в репозитории для разработки выбранного по заданию из 1 лабораторной работы проекта. Реализовать каждый из модулей в отдельных ветках, сделать не менее пяти фиксаций касательно разработки программного кода и после влить в ветку master. Зафиксировать лог ревизии для каждой из веток.

Удалить все созданные ветки и предоставить вывод команды git log для trunk после вливания всех веток.

Прядок выполнения работы

1. Создать ветку для первого модуля программы.
2. Последовательно переключить рабочую копию на созданную ветку.
3. Сделать не менее пяти фиксаций, касающихся добавления файлов и редактирования кода.
4. Проверить лог ревизии
   git log --pretty=format:\"%h %ad | %s%d [%an]\" --graph —date=short
5. Переключить рабочую копию на **master**.
   git checkout master
6. Повторить пункт 1 для оставшихся модулей.
7. Влить все ветки
   git merge <название ветки>
8. Получить конфликт слияния.
9. Разрешить конфликт и закоммитить результирующий код.
10. Посмотреть лог ревизии
    git log --pretty=format:\"%h %ad | %s%d [%an]\" --graph —date=short
11. Удалить все ветки, кроме **master**
    git branch –D <название ветки>

Содержание отчёта

1. Титульный лист.
2. Вывод команды git log --pretty=format:\"%h %ad | %s%d [%an]\" --graph --date=short, касающийся работы с каждой из веток.
3. Вывод команды git log для trunk после вливания всех веток.
4. Вывод команды git status, демонстрирующую конфликтную ситуацию.
5. Содержимое конфликтующего файла (достаточно только строк, в которых произошел конфликт)

Лабораторная работа 3

Восстановление данных из ранних ревизий

Цель работы: получить практические навыки в совместной разработке программного средствами системы контроля версий Git, а также использование инструмента восстановления предыдущих ревизий.

Контрольные вопросы

1. Конфликты слияния в системах контроля версий.
2. Ветвление в системах контроля версий.
3. Восстановление репозитория.
4. Восстановление ранее удаленной ветки.

Методические указания

Рассматривая модель разработки, необходимо иметь свою копию изначального репозитория, в которой и будет вестись работа, и изменения из которой и будут предлагаться затем автору изначального репозитория.

Скопировать копию из репозитория можно командой: ***git clone git@github.com:username/репозиторий.git***. Склонированный репозиторий имеет одну привязку к удалённому репозиторию, названную origin, которая указывает на вашу копию на GitHub, а не на оригинальный репозиторий, чтобы отслеживать изменения и в нём, вам нужно будет добавить другую привязку, названную, например, upstream (см. рис. 3.1).

```
cd репозиторий
git remote add upstream git://github.com/octocat/репозиторий.git
git fetch upstream
```

Рисунок 3.1 – команды для отслеживания изменений

По завершению работы, отправим файлы на GitHub командой: ***git push origin feature #Загружает изменения в текущей ветви в origin в ветвь feature***.

Более подробно по всем командам Git можно почитать в официальном руководстве по ссылке https://mirrors.edge.kernel.org/pub/software/scm/git/docs/

Удаленные ссылки – это ссылки (указатели) в удаленных репозиториях, включая ветки, теги и так далее. Полный список удаленных ссылок можно получить с помощью команды ***git ls-remote <remote>*** или команды ***git remote show <remote>*** для получения удаленных веток и дополнительной

информации. Тем не менее, более распространенным способом является использование веток слежения.

Ветки слежения – это ссылки на определенное состояние удаленных веток. Это локальные ветки, которые нельзя перемещать. Git перемещает их автоматически при любой коммуникации с удаленным репозиторием, чтобы гарантировать точное соответствие с ним.

Имена веток слежения имеют вид <remote>/<branch>. Например, если вы хотите посмотреть, как выглядела ветка master на сервере origin во время последнего соединения с ним, используйте ветку origin/master.

Получение локальной ветки из удаленной автоматически создает то, что называется «веткой слежения» (а ветка, за которой следит локальная называется «upstream branch»). Ветки слежения – это локальные ветки, которые напрямую связаны с удаленной веткой. Если, находясь на ветке слежения, выполнить git pull, то Git уже будет знать с какого сервера получать данные и какую ветку использовать для слияния.

При клонировании репозитория, как правило, автоматически создается ветка master, которая следит за origin/master. Однако, при желании вы можете настроить отслеживание и других веток – следить за ветками на других серверах или отключить слежение за веткой master.

Можно удалить ветку на сервере используя параметр –delete для команды git push. Все, что делает эта строка – удаляет указатель на сервере. Как правило, Git сервер хранит данные, пока не запустится сборщик мусора, поэтому если ветка была удалена случайно, чаще всего ее легко восстановить.

Если ветка была в локальном репозитории git за последние 30 дней, то можно найти ее в рефлоге используя следующую команду: ***git reflog***. Найти имя ветки в рефлоге и восстановить с помощью команды: ***git checkout –b branch_name HEAD@{x}***. Также можно проверить идентификатор фиксации и создать ответвление от этой точки фиксации: ***git checkout –b branch_name <commit id>***.

Восстановить также можно по SHA1 для коммита расположенного в конце удаленной ветки, а затем ***git checkout [sha]***. Таким образом, можно воссоздать ветку командой: ***git checkout –b [branchname]***. Также можно выполнить эту операцию за один шаг, с помощью команды: ***git checkout –b <branch> <sha>***.

Если ваши коммиты не находятся в вашем рефлоге, можно воспользоваться такой командой, которая переводит ветку в sha of (создает файл со всеми оборванными коммитами): ***git fsck --full --no-reflogs --unreachable --lost-found | grep commit | cut -d\ -f3 | xargs -n 1 git log -n 1 --***

pretty=oneline > .git/lost-found.txt. Также можно создать псевдоним для этой команды с целью многоразового исполнения, с помощью команды: ***git config --global alias.rescue '!git fsck --full --no-reflogs --unreachable --lost-found | grep commit | cut -d\ -f3 | xargs -n 1 git log -n 1 --pretty=oneline > .git/lost-found.txt'*** и использовать с ***git rescue***.

Задание к лабораторной работе

Внести изменения в программный проект одного из одногруппников относительно ранее подготовленного им описания компонентов/модулей. Зафиксировать изменения.

Восстановить свой репозиторий после изменения другим одногруппником средствами системы контроля версий git.

Прядок выполнения работы

1. Скопировать в каталог «команда» рабочую ветку одного из одногруппиков.
2. На основании описания модулей разрабатываемого проекта одногруппника составить план совместной разработки одного из них.
3. Сделать копию текущего репозитория, в которой и будет вестись работа.
4. Сделать не менее 10-ти фиксаций программного кода.
5. Заменить проект.
6. Дождаться когда эту же операцию сделают с вашим репозиторием, если еще не изменено, и восстановить до предыдущего состояния.

Содержание отчёта

1. Титульный лист.
2. Структура и ФИО программного кода проекта выбранного одногруппника.
3. Лог состояния своего репозитория до восстановления.
4. Лог состояния своего репозитория после восстановления.

Лабораторная работа 4

Создание самодокументирующегося кода

Цель работы: научиться добавлять в программный код специальным образом оформление докблок-комментарии, для последующей автоматической генерации API reference, а также познакомиться с форматом оформления документации DocBook.

Контрольные вопросы

1. Принципы организации само документирующегося кода.
2. Использование самодокументирующегося кода.
3. Особенности применения API reference.
4. DocBook: особенности, преимущества и недостатки.
5. DocBook: структура документа.

Методические указания

В open source проектах нередко встречается практика, когда статьи по документации хранятся в том же репозитории, что и основной код. Например, в библиотеке для генерации фейковых фикстур для PHP документация помещена в README файл; чтоб дочитать до конца, нужно немного поскролить. Популярный HTTP-клиент для PHP Guzzle хранит инструкции по применению в разных файлах в отдельной папке docs. Хранить документацию возле кода — это, конечно, хорошо и удобно. Один раз, скачав пакет вендора, у вас есть и код, и документация. Если ваша библиотека небольшая, если она стабильная и не предполагает в будущем постоянных изменений API, которые повлекут за собой постоянное переписывание документации, тогда можете смело размещать документацию в репозитории вашего проекта.

Но всё же всему есть разумный предел. Например, если вы затеяли создание собственного фреймворка, который пишется командой разработчиков, и планируете постоянные релизы, он должен быть полностью задокументирован, более того, документация должна быть переведена на несколько языков, и тогда помещать документацию в репозиторий проекта — не вариант. Потому что для документации характерны постоянные правки, доработки, переводы, исправление опечаток. Это все выливается в большое

количество коммитов-фиксов, которые засоряют историю проекта. Навигация по истории коммитов, где изменения кода теряются между изменениями документации, сложна и неудобна. В таком случае лучше создать отдельный репозиторий для документации, например, как это сделали для Symfony. GitHub, GitLab, Bitbucket также предоставляют встроенный инструмент WIKI, его фишкой является то, что он прикреплен к проекту, т.е. не является самостоятельным репозиторием. Но к нему также можно обращаться через Git, т.е. стянуть себе документацию, редактировать её в удобном для себе редакторе, группировать изменения в коммиты и отправлять на сервер, так же и получать свежие правки. Вот пример хорошо оформленной WIKI для библиотеки визуализации D3.js. Конечно, же можно создать сайт для своего продукта и разместить документацию на нем. Но если вы используете какой-либо способ из перечисленных выше, то вы сможете сгенерировать веб-страницы документации из вашего Git или WIKI репозитория, инструменты для этого есть. Если вы любитель комплексных решений, обратите внимание на Confluence от Atlassian. Возможности Confluence вышли далеко за пределы обычного WIKI-движка.

Для комментариев свойственные следующие характеристики

Неактуальность. Очень часто, меняя код, забывают поменять комментарий. Это особенно актуально, когда над одним участком кода трудятся несколько программистов. Комментарии есть, но они написаны одним из программистов, остальные не решаются изменить чужие комментарии либо ленятся, либо просто не обращают внимание. В результате, старый неактуальный комментарий только запутает нового человека в команде. Решение проблемы простое. Либо всегда следить за актуальностью комментариев, что потребует значительного внимания и стараний. Либо удалить неактуальный комментарий. Отсутствие комментария лучше, чем устарелый, неактуальный комментарий.

Избыточность. Это когда комментарий написан там, где он не нужен, где все понятно и без комментария.

Неполнота. Во время написания программы вы можете быстро зафиксировать свою мысль в виде комментария сразу в коде. Позже вы вернётесь к этому месту, комментарий напомнит вашу мысль, и вы сможете ее продолжить. После того, как мысль превратилась в код, неполный комментарий нужно убрать, либо превратить его во что-то более осмысленное. Другими словами, не заставляйте читателей догадываться, что вы имели в виду.

Недостоверность. Людям свойственно делать ошибки. Программисты их делают не только в коде, но и в комментариях. Либо из-за невнимательности, либо из-за усталости, либо из-за незнания иностранного языка в комментарий вносится путаница и дезинформация. К сожалению, от этого никто не застрахован. Единственное, что можно посоветовать в таком случае — это ответственно относиться к комментариям. Если вы уже решились что-то написать, то пишите грамотно. Перфекционизм в комментариях не помешает.

Неочевидность. Это когда в конкретном месте кода используются неизвестные или не очевидные термины.

Есть отдельный вид комментариев в PHP, который имеет свой устоявшийся стандарт — это докблоки (DocBlock). Для обработки докблоков существует инструмент phpDocumentor (ранее известен как phpDoc). Он умеет читать докблоки из кода и строить на их основе документацию. DocBlock — это комбинация DocComment и помещенных в него описаний по стандарту PHPDoc.

В PHP с помощью докблоков можно документировать такие элементы:

- Функции;
- Константы;
- Классы;
- Интерфейсы;
- Трейты;
- Константы классов;
- Свойства;
- Методы.

Важно также, что один докблок может быть применён только к одному структурному элементу, т.е. на каждую функцию — свой докблок, на переменную внутри функции — свой, для класса — свой.

DocBook

Подмножество языка разметки SGML или XML, предназначенное для разметки документов, такое же, как HTML для разметки веб-документов.

Преобразованием DocBook-документа в форматы, доступные для печатного или простого визуального представления (в том числе PDF, HTML, man-страницы) занимаются различные утилиты, обычно осуществляющие такое преобразование на основе настраиваемых шаблонов, или «таблиц стилей» (DSSSL или XSL), то есть происходит настоящая изоляция структуры документа от визуального представления.

В отличие от HTML-документа, DocBook-документ не рассматривается как конечный формат, поэтому, например. Один документ в этом формате после преобразования может выглядеть как один большой документ со сложной структурой, и как набор небольших простых документов-глав.

DocBook разрабатывался для создания технической документации, но может использоваться и в других целях (для создания сайтов, с преобразованием в HTML, для создания презентаций).

Примеры докблоков

Докблок для класса:

```
/**
 * Транспортное средство
 *
 * Класс описывает абстрактное транспортное
 * средство с базовыми характеристиками.
 *
 * @author      John Doe
 * @version     1.0.1
 * @copyright GNU Public License
 * @todo          Реализовать все методы
 */
class Vehicle {}
```

Докблок для свойства:

```
/**
 * Производитель транспортного средства
 *
 * Используем только простое символьное
 * имя производителя. Если будет
 * необходима детализация, создадим
 * класс Brand
 *
 * @var             string   $brand
 */
public $brand;
```

Докблок для метода/функции:

```
/**
 * Работа со свойством {@link $brand}
 *
 * Если аргумент определен, то устанавливается новое
 * значение свойства и возвращается указатель на
 * объект, иначе возвращается текущее значение
 * свойства.
 *
 * @param   string   $brand  Производитель
 * @return  mixed            Возвращает текущее значение
 *                           свойства или указатель на
 *                           объект
 */
public function brand($brand=NULL);
```

Пример XML кода для DocBook

```
<?xml version="1.0" encoding="UTF-8"?>
<book xml:id="simple_book"
xmlns="http://docbook.org/ns/docbook" version="5.0">
  <title>Very simple book</title>
  <chapter xml:id="chapter_1">
    <title>Chapter 1</title>
    <para>Hello world!</para>
    <para>I hope that your day is proceeding
<emphasis>splendidly</emphasis>!</para>
  </chapter>
  <chapter xml:id="chapter_2">
    <title>Chapter 2</title>
    <para>Hello again, world!</para>
  </chapter>
</book>
```

Задание к лабораторной работе

Внедрить в ранее разработанный программный код докблоки.

Разработать DocBook и проверить корректность созданного DocBook файла с помощью средств редактора или одного из онлайн валидаторов.

Прядок выполнения работы

1. Во всех файлах с исходным кодом добавить докблоки ко всем классам, методам, свойствам и функциям.

2. Закоммитить изменения в репозиторий.

3. Установить один из XML-редакторов (по желанию).

4. Оформить краткое описание разработанного продукта (не менее пяти абзацев).

5. Рассмотреть с примерами кода не менее трех сценариев использования вашего приложения.

6. Проверить корректность созданного DocBook файла с помощью средств редактора или одного из онлайн валидаторов, например, **mashuosoft.com/docbook/validator**

Содержание отчёта

1. Титульный лист.
2. Несколько примеров докблоков из программного кода разрабатываемого проекта.
3. Содержимое файла DocBook и демонстрация его работы.
4. Отчет, о корректности составленный в выбранном валидаторе.

Лабораторная работа 5

Разработка ручной документации

Цель работы: получить практические навыки в разработке справочного руководства в форматах CHM и HTA.

Контрольные вопросы

1. Формат HTML Help.
2. Особенности компиляции CHM файлов.
3. HTML Application.
4. Создание HTA-документов.

Методические указания

Формат HTML Help или CHM был разработан компанией Microsoft в 1997 г. Сегодня CHM остается стандартом справки для приложений, работающих в ОС Windows. Средство для просмотра CHM-файлов есть во всех версиях Windows 10. Более того, наличие справки в приложении часто рассматривается как один из показателей качества приложения. В этой статье я расскажу о том, как создать справку в формате CHM.

Справка в формате HTML Help (CHM) представляет собой скомпилированный HTML — автономный веб-сайт, сжатый и упакованный в файл формата CHM. Наряду со стандартным функционалом, таким как динамическое оглавление, указатель и полнотекстовый поиск, HTML Help может содержать и дополнительный функционал, например, избранное и т.д. Подробное описание формата можно посмотреть в статье Формат HTML Help. Создать CHM справку можно при помощи специальных программ, как платных, так и бесплатных.

Перечень некоторых бесплатных программ для создания CHM-справки:

1. MS HTML Help Workshop;
2. HelpNDoc;
3. HTM2CHM и другие.

HTA (HTML Application) - простая технология, позволяющая создавать полезные приложения даже без знания серьезных языков программирования.

В основе разработанной Microsoft технологии исполнения HTML-приложений вне веб-браузера, получившей название HTA

(HTML Application), заложена благородная идея: помочь пользователям, не обремененным познаниями в области программирования на C+, Visual Basic, Delphi и других "тяжелых" языках, создавать приложения, основанные на более известных языках сценариев VBScript и JavaScript, а также в стандартном HTML-коде. Понятно, что, запатентовав подобную технологию, корпорация попытается обеспечить ее поддержку другими операционными системами. И все бы ничего, если бы не формат файлов, с помощью которого планируется продвигать эту технологию.

Вообще-то, сам по себе формат HTA достаточно удобен. К примеру, именно на его базе построена справочная система Windows. Благодаря ему же обычные HTML-страницы запускаются без браузера и полностью отображаются в окне HTA-файла - со всеми ссылками, графикой, эффектами и скриптами. Создавать и редактировать подобные файлы гораздо проще и быстрее, чем, например, программу на C или Delphi. Удобство работы с HTA еще и в широкой доступности редакторов - для этого формата могут использоваться как обычные текстовые редакторы вроде Notepad, так и специализированные HTML-редакторы (наподобие FrontPage или Macromedia Dreamweaver).

Создание HTA-документов сводится к установке специальных атрибутов в разделе <head>…</head> обычной HTML-страницы, после чего остается только сохранить ее с расширением. hta. Вот пример листинга простого HTA-файла:

```
<HTML>
<HEAD>
<HTA:APPLICATION ID="HelloExample"
   BORDER="bold"
   BORDERSTYLE="complex"/>
<TITLE>HTA - Hello World</TITLE>
</HEAD>
<BODY>
<H2>HTA - Hello World</H2>
</BODY>
</HTML>
```

Рисунок 5.1 – Пример hta файла

При попытке открыть HTA-файл в браузере пользователю предлагается сохранить этот файл на жестком диске, так как браузер не интерпретирует расширение HTA. Windows же определяет HTA-файл как исполняемый и потому запускает его по щелчку на пиктограмме или через командную строку.

Благодаря тому, что стандарт HTA позволяет исполнять сценарии Java и VB, в этом формате можно создать много полезных приложений, от простого календаря и конвертора валют до ньюсридера и менеджера онлайн-доступа к блогам, сайтам и форумам. Поэтому, обладая даже небольшим опытом создания веб-страниц и написания простых Java-скриптов, пользователь может создавать HTA-файлы, которые ни по функциональности, ни по дизайну не уступают небольшим программам, написанным на более серьезных языках.

К сожалению, HTA-файлы нельзя просматривать в браузере, а значит, все средства защиты веб-браузеров оказываются бесполезны для них. Именно этот факт чаще всего приводится в качестве аргумента против HTA. Ведь в обход защиты с помощью VBscript и Jscript вполне можно организовать запись данных в реестр в фоновом режиме, запускать сторонние программы и скрипты, выполнять чтение и запись на диск. Поэтому хакеры достаточно хорошо освоили этот стандарт и с успехом применяют его для своих целей.

Атрибуты HTA-документов	
Атрибут	**Описание**
applicationName	имя приложения, использующегося для идентификации при исполнении HTA-документа
border	вид обрамления окна; среди возможных значений - thin, dialog, none, thick
borderStyle	стиль обрамления окна; среди возможных значений - complex, normal, raised, static, sunken
caption	указывает, нужно ли показывать заголовок окна (yes/no)
icon	путь к пиктограмме, которая будет показана в заголовке; допустимые параметры пиктограммы - 32x32 пикселей, формат. ICO
maximizeButton	указывает, будет ли активизирована кнопка "восстановить" в заголовке документа (yes/no)
minimizeButton	указывает, будет ли активизирована кнопка "свернуть" в заголовке документа (yes/no)
showInTaskbar	указывает, будет ли показан документ в панели инструментов Windows (yes/no)
windowState	первоначальный размер окна; среди возможных значений - normal, minimize, и maximize
innerBorder	указывает, будет ли окно иметь внутреннюю границу (yes/no)
navigable	указывает, будут ли ссылки открываться в отдельных окнах или в одном (yes/no)
scroll	указывает, будет ли окно иметь полосу прокрутки (yes/no)
scrollFlat	указывает вид полосы прокрутки: 3D или нет (yes/no)
singleInstance	определяет, возможно ли одновременное открытие нескольких окон для HTA-документов (yes/no)
sysMenu	указывает, будут ли показаны составляющие заголовка окна (yes/no)
contextMenu	указывает на доступность контекстного меню, вызываемого правой кнопкой мыши (yes/no)

Задание к лабораторной работе

Разработать документацию в форматах CHM и HTA на основе ранее подготовленного описания модулей, в которых учесть несколько сценариев использования вашего проекта.

Руководство пользователя в файле форматом HTA, руководство программиста – CHM.

Прядок выполнения работы

1. Изучить ранее созданное описание модулей разрабатываемого проекта.
2. Составить руководство пользователя в формате hta (не менее 10 пунктов).
3. Составить руководство программиста в формате chm (не менее 10 пунктов).
4. Составить отчет и закоммитить в репозиторий.

Содержание отчёта

1. Титульный лист.
2. Демонстрация справочных руководств в файлах chm, hta.
3. Руководство программиста в файле chm.
4. Руководство пользователя в файле hta.

Лабораторная работа 6

Веб-сайт из репозитория

Цель работы: получить практические навыки в создании веб-сайтов на основе репозитория.

Контрольные вопросы

1. Веб-сайт структура и правила оформления.
2. Виды веб-сайтов.
3. Доменное имя, виды.
4. Доменная зона.
5. Понятие корневого домена.
6. Структура изменения доменного имени на хостингах.
7. Доменный бизнес.
8. Юридический аспект использования доменных имен.

Методические указания

Разработать одностраничный веб-сайт презентацию.

Загрузить файлы веб-сайта в репозиторий, предварительно сместив данные предыдущих лабораторных работ, так чтобы они не мешали работе сайта.

Чтобы проверить работу сайта достаточно перейти по адресу username.github.io.

Гитхаб позволяет использовать собственное доменное имя вместо стандартного username.github.io. При наличии своего домена перейти во вкладку «Settings» в интерфейсе репозитория, и в разделе «GitHub Pages» в поле «Custom domain» ввести название домена (например: student-donntu-pi.ru) и нажать кнопку «Save».

Теперь Гитхаб знает о новом домене, однако, этого недостаточно — нужно изменить информацию о DNS-записях самого домена, для этого необходимо перейти на сайт доменного регистратора, где домен был куплен. Интерфейс работы с DNS-записями разный у каждого регистратора, но суть примерно одинакова.

Нужно настроить A-запись домена, для этого перейти в панель управления DNS-записями, найти (или добавить) A-запись и указать, например, «192.30.252.153» в качестве её значения.

Всё готово, в течение нескольких часов можно открыть свой сайт, используя свой домен.

Задание к лабораторной работе

Разработать одностраничный веб-сайт презентацию своего проекта, сместить данные предыдущих лабораторных работ во внутренний каталог, закоммитить в репозиторий файлы веб-сайта и отобразить его в браузере средствами веб-сервиса github.com.

Прядок выполнения работы

1. Разработать веб-сайт.
2. Сместить данные предыдущих лабораторных работ во внутренние каталоги.
3. Закоммитить файлы сайта в главный репозиторий.
4. Проверить работу сайта.

Содержание отчёта

1. Титульный лист.
2. Описание структуры репозитория после фиксации коммита файлов веб-сайта.
3. Демонстрация работы веб-сайта.

СПИСОК РЕКОМЕНДОВАННОЙ ЛИТЕРАТУРЫ

1. Scott Chacon, Pro Git: Пер.с англ. Без выходных данных.-М.:Apress,2012.-245 с.
2. Д. В. Ефанов, П. Г. Рощин, Система управления версиями GIT: учебное пособие.-М.: М-во образования и науки Российской Федерации, Нац. исслед. ядерный ун-т "МИФИ", 2014. -88 с.
3. Л.В. Фишерман, Git. Практическое руководство. Управление и контроль версий в разработке программного обеспечения.-СПб.: Наука и Техника, 2021. -304 с., ил.
4. Скотт Чакон, Бен Штрауб, Git для профессионального программиста.-СПб.: Apress, 2016. -496 с.
5. Бен Линн, Волшебство Git.-СПб.: Apress, 2011. -64 с.
6. С.В. Одиночкина, Основы технологий XML. -СПб: НИУ ИТМО, 2013. - 56с.
7. Основы XML/ - М.: Национальный Открытый Университет «ИНТУИТ», 2016. -437 с.
8. Г.В. Попов, Технология разработки стандартов и нормативной документации. Практикум [Текст] : учеб. пособие / Г.В. Попов, Н.Л. Клеменова, А.Н. Пегина, О.А. Орловцева; Воронеж. гос. ун-т инж. технол. – Воронеж : ВГУИТ, 2015. – 52 с.
9. Никсон Р., Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5. 4-е изд. – СПб.: Питер, 2016. – 768с.Ж ил. – (Серия «Бестселлеры O`Reilly»).
10. В.А. Дронова, Django 3.0 Практика создания веб-сайтов на Python. – СПб.: БХВ-Петербург, 2021. – 704 с.: ил. – (Профессиональное программирование).

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторным занятиям по дисциплине

«Профессиональная практика программной инженерии»

для обучающихся по

направлению подготовки 09.03.04 «Программная инженерия»

профиль «Инженерия программного обеспечения» всех форм обучения

Составители:

Филипишин Дмитрий Александрович – ассистент кафедры программной инженерии им. Л.П. Фельдмана ГОУВПО «ДОННТУ».

Ответственный за выпуск:

Зори Сергей Анатольевич – доктор технических наук, доцент, заведующий кафедрой «Программная инженерия» им. Л.П. Фельдмана ГОУВПО «ДОННТУ»