Лекция 1.

Методы организации работы в команде разработчиков.

Системы контроля версий.

Е.В. Поколодина, Н.А. Долгова, Д.В Ананьев

«Ревьюирование программных модулей», глава 1, п.1.1

Методы организации работы в команде разработчиков

Современные средства разработки программного обеспечения частично позволяют решить данные проблемы:

- Microsoft Visual Studio Team System (набор инструментов от Microsoft для разработки программных приложений, упрощения совместной работы над проектами, инструментов для тестирования и отладки разрабатываемых программ, а также построения отчетов.

Visual Studio Team System состоит из 5 основных продуктов, которые можно разделить на серверные и клиентские приложения

При использовании Visual Studio Team System решаются следующие задачи:

- Повышение предсказуемости успеха проекта
- Рост производительности труда команды разработчиков за счет понижения сложности процессов проектирования и реализации современных сервисно-ориентированных решений
- Обеспечение сотрудничества команды путем интеграции коммуникационных и прочих средств, используемых ее членами
- Расширение возможностей командной работы за счет обеспечения удаленным пользователям доступа к надежному, защищенному и масштабируемому окружению
- Предоставление команде разработчиков гибкого и расширяемого коммуникационного решения с настраиваемыми сервисами

Ролевые кластеры модели проектной группы MSF (*Microsoft Solution Framework*)

Основные роли ИТ-проекта:

- руководитель проекта
- архитектор
- проектировщик
- разработчик
- тестировщик
- эксперт в ПО
- специалист по интерфейсу
- библиотекарь

Ключевые члены проектной команды:

- руководитель проекта
- архитектор
- разработчик
- тестировщик

- *Роль в проекте* (*проектная роль*) - определенный набор функций и *полномочий* в проекте, созданный с целью *распределения обязанностей* между членами команды проекта.
- *Полномочия* - право задействовать ресурсы проекта, принимать решения и утверждать одобрение действий или результатов.
- *Ответственность* - работа, которую член команды проекта должен выполнить для завершения операций проекта.
- *Квалификация* - навыки и способности, необходимые для выполнения операций проекта.

Принципы объединения ролей

- Во-первых, роль команды разработчиков не может быть объединена ни с какой другой *ролью.*

  Разработчики – это создатели проекта. Наделение разработчиков дополнительными обязанностями приводит к нарушению календарного *графика* проекта.

- Второй принцип – это избежание *сочетания* ролей, имеющих предопределенные *конфликты интересов*.

	Управление продуктом	Управление программой	Разработка	Тестирование	Удовлетворение потребителя	Управление выпуском
Управление продуктом		-	-	+	+	±
Управление программой	-		-	±	±	+
Разработка	-	-		-	-	-
Тестирование	+	±	-		+	+
Удовлетворение потребителя	+	±	-	+		±
Управление выпуском	±	+	-	+	±	

+ Допустимо ± Нежелательно - Нельзя

Роли действующих лиц проекта и совмещение ролей	
Роли	**Характеристика совмещения ролей**
Менеджер и архитектор	Желательно
Менеджер и руководитель команды	Противоречиво
Руководитель команды и архитектор	Возможно
Руководитель команды и проектировщик подсистемы	Нежелательно
Менеджер и разработчик	Не допускается
Для различных разработчиков	Эффективно с ограничениями(обычное дело)
Создание документации (все сотрудники)	Успешно распределяется
Специалист по интерфейсу и менеджер	Разумно
Эксперт предметной области и менеджер	Зачастую разумно
Специалист по интерфейсу и эксперт предметной области	Редко бывает эффективно
Эксперт предметной области и разработчик	Бывает полезно
Специалист по интерфейсу и разработчик	Часто полезно
Библиотекарь и один из разработчиков	Допустимо
Тестировщики и другие члены команды	Перекрестно
Эксперт предметной области, тестировщик	Оправданно

Системы контроля версий

Системы контроля версий

Система управления (контроля) версиями (*Version Control System*) — программное обеспечение для облегчения работы с изменяющейся информацией.

Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Системы контроля версий

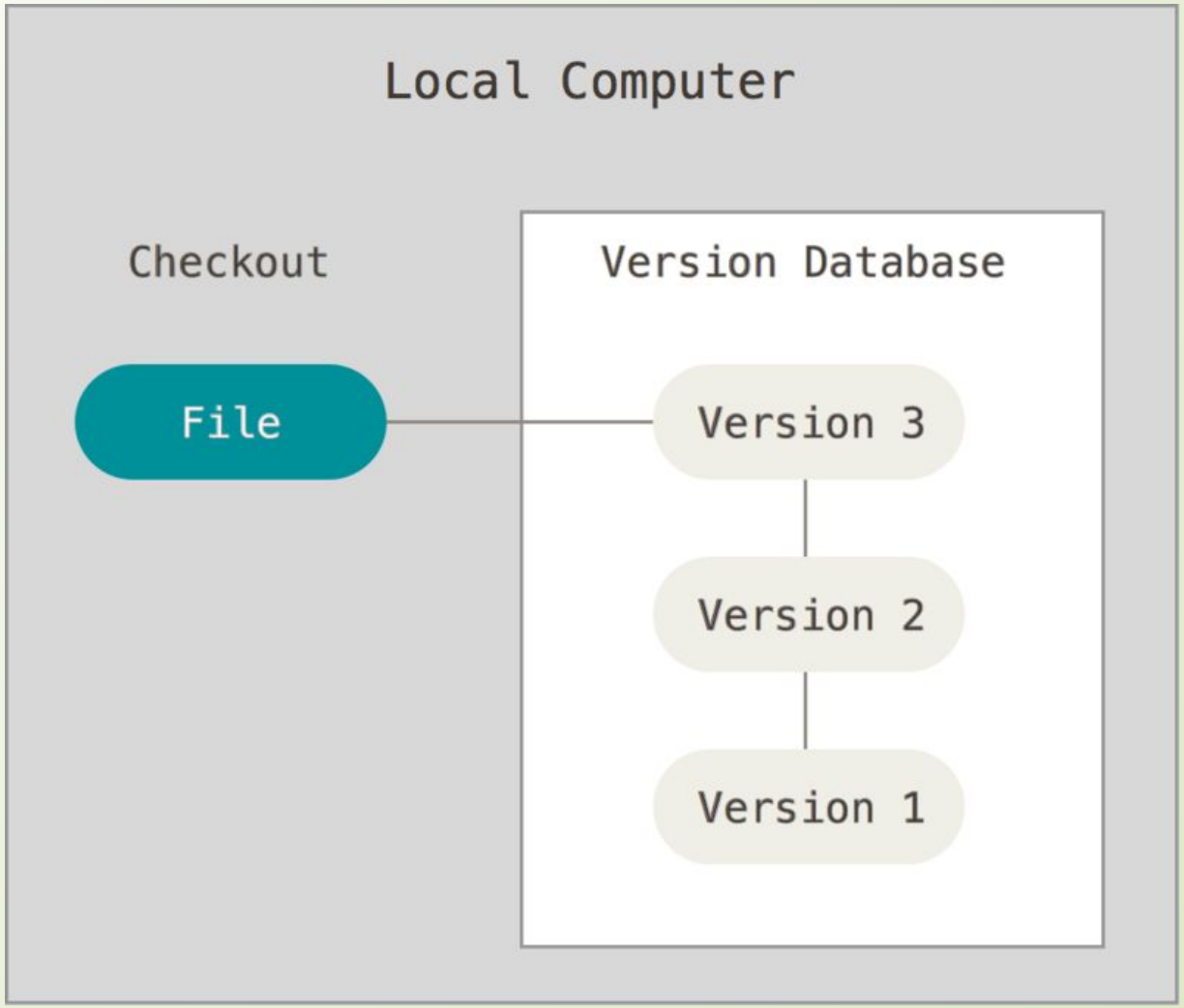
Системы контроля версий решают следующие проблемы:

- - хранение версий файлов;
- - возможность получить любые предыдущие версии хранимых файлов;
- - просмотр изменений внесенных между заданными в запросе версиями;
- - сохранение и просмотр комментариев и авторов к внесенным изменениям.

Типы систем контроля версий

- Локальные системы контроля версий
  - Пример: rcs.
- Централизованные системы контроля версий
  - Пример: CVS, Subversion и Perforce.
- Децентрализованные (распределенные) системы контроля версий
  - Пример: Git, Mercurial, Bazaar, Darcs.

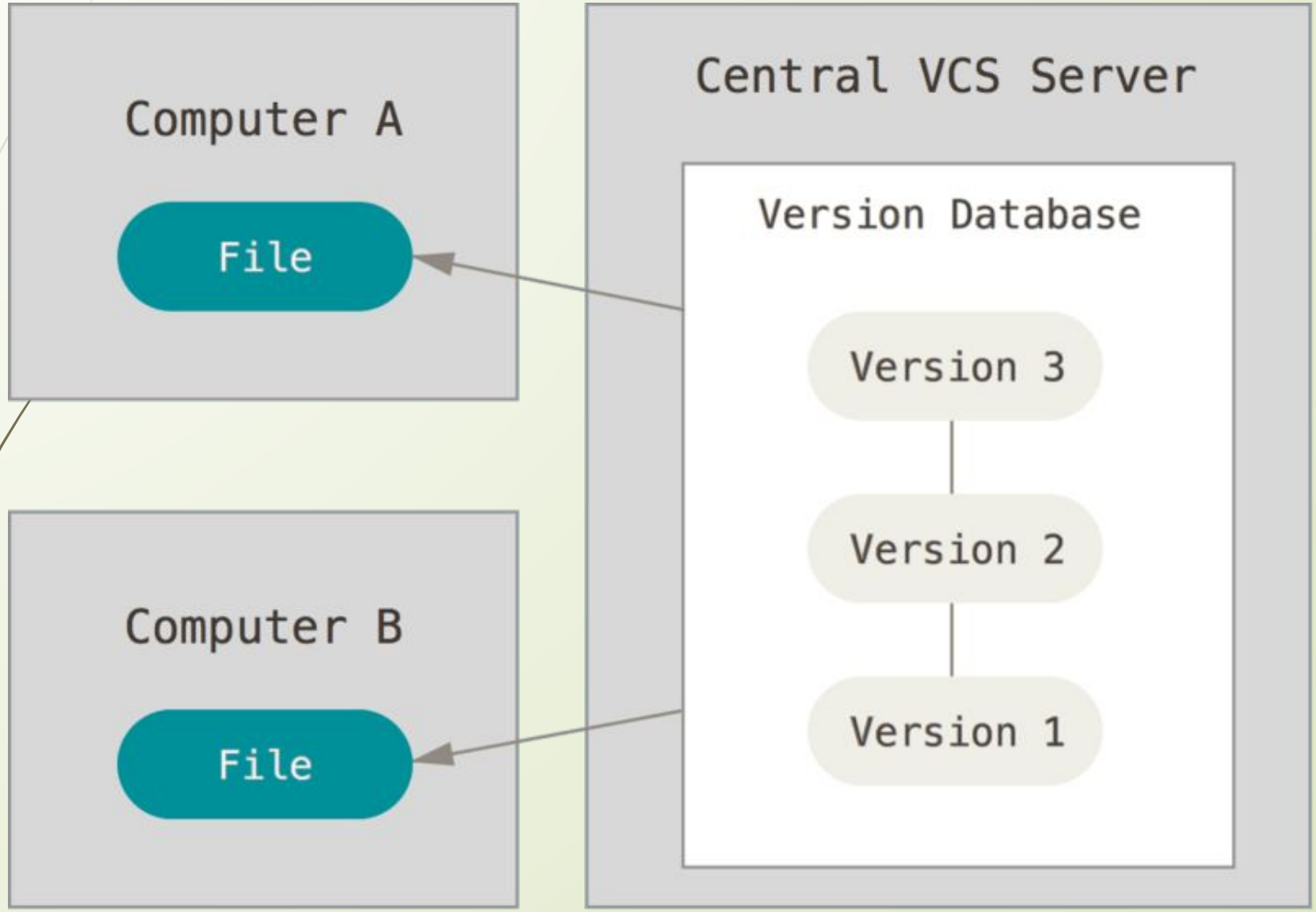
Локальные системы контроля версий



Достоинства и недостатки

Достоинства	Недостатки
Возможность восстановления данных из определенной версии;	Возможность потери данных вследствие возникновения физических поломок оборудования;
Высокая скорость выполнения восстановления.	Отсутствие возможности совместной разработки.

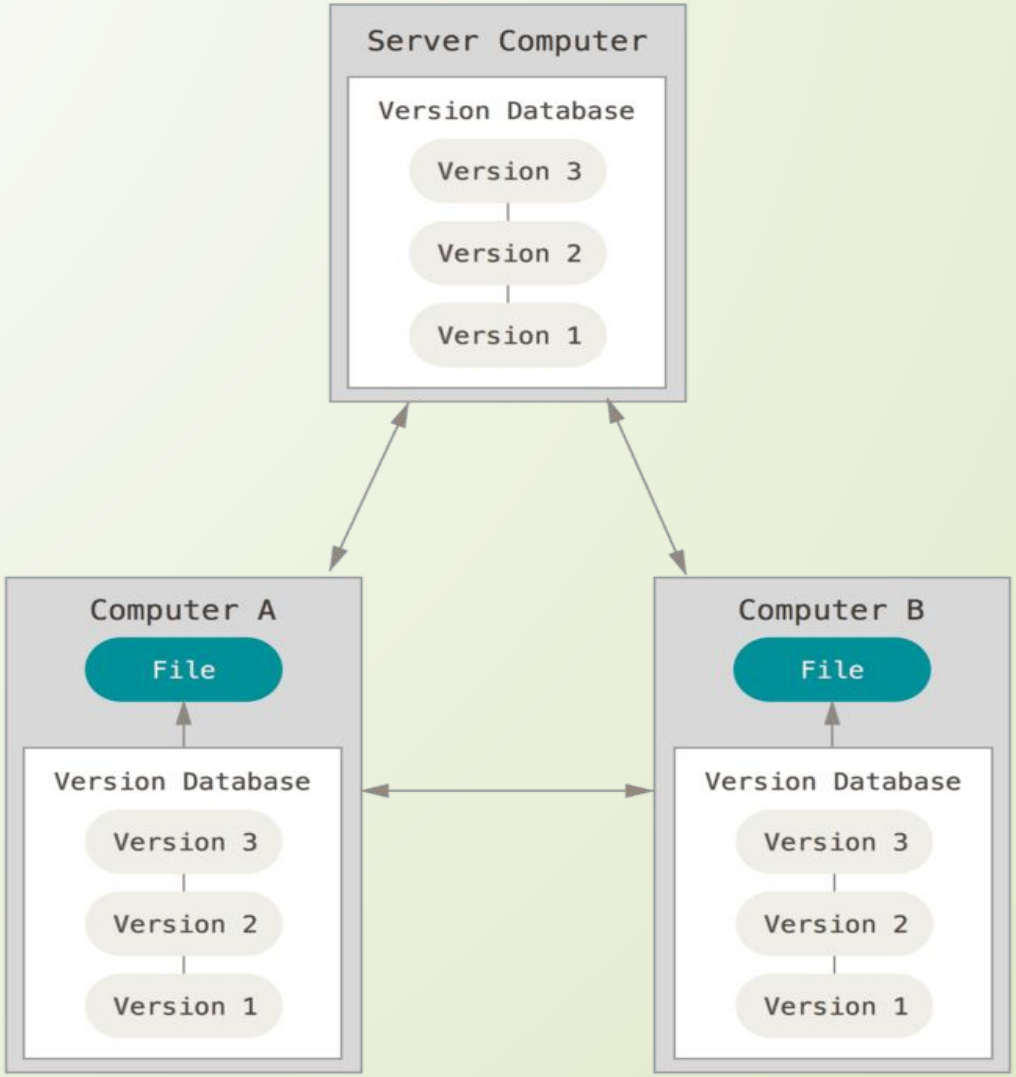
Централизованные системы контроля версий



Достоинства и недостатки

Достоинства	Недостатки
Возможность восстановления данных из определенной версии;	Отсутствие доступа к данным при сбое работы сервера;
Возможность ведения командной разработки проекта.	довольно низкая скорость работы.

Децентрализованные системы контроля версий



Достоинства и недостатки

Достоинства
Возможность восстановления данных из определенной версии
Возможность ведения командной разработки проекта
При сбое работы сервера система сохраняет данные в локальном репозитории, что позволяет эффективно вести процесс разработки, а после восстановления работы сервера, передать все изменения в удаленный репозиторий
При физической поломке сервера данные можно легко перенести в новый удалённый репозиторий с любого локального репозитория
Высокая скорость работы

Современные системы контроля версий

- Существует много систем контроля версий (Git, Darcs, Mercurial, Bazaar, Monotone и т.д), сходных по принципу работы и конечным задачам.
- Самая популярная на сегодняшний день система контроля версий – Git

Лекция 2.

Анализ программных продуктов

Е.В. Поколодина, Н.А. Долгова, Д.В Ананьев

«Ревьюирование программных модулей», глава 1, п.1.3

- **Анализ ПО** — это часть процесса разработки программного обеспечения, включающая в себя сбор требований к программному обеспечению, их систематизацию, выявление взаимосвязей, а также документирование.

- **Шаг 1.** Комиссия экспертов формирует таблицу критериев оценки, являющихся самыми важными для потребителя.

- **Шаг 2.** Та же комиссия для каждого критерия определяет методику оценки выполнения критерия таким образом, чтобы "обезразмерить" исходные показатели (шкалирование

)

- **Шаг 3.** Для каждого критерия эксперты выставляют коэффициенты значимости критерия для оценки продукта. Коэффициенты распределяются на отрезке от 0 до 1.

- **Шаг 4.** Производится расчет аддитивной суммы интегральной оценки для каждого сравниваемого продукта по следующей формуле

$$O_{зп}=\sum_{i=1}^{n}(Z_i \bullet K_i) \quad ,$$

- **Шаг 5.** Значения интегральных оценок для каждого сравниваемого продукта ранжируются по убыванию

Пример критериев сравнения ПП

№ п/п	Наименование критерия	Методика оценки критерия	Коэффициент значимости критерия
Общие критерии			
1	Наличие сертификата соответствия, выданного в российской системе сертификации	Определяется по наличию копий сертификатов, предоставляемых участником. Критерий считается выполненным, если имеется 1 сертификат или более	1
2	Наличие сертификата соответствия, выданного в системах сертификации других государств	Определяется по наличию копий сертификатов, предоставляемых участником. Критерий считается выполненным, если имеется 1 сертификат или более	0,5
3	Наличие подтверждения отдельных свойств продукта, выданного производителями программных и/или аппаратных платформ	Определяется по сведениям, предоставляемым участником. Критерий считается выполненным, если имеется 1 подтверждение или более	0,5
4	Наличие охранных документов	Определяется по сведениям, предоставляемым участником. Критерий считается выполненным, если имеется свидетельство о регистрации программы в Роспатенте	1
Технические критерии			
5	Наличие API и его описания для интеграции в другие программные продукты	Определяется по сведениям, предоставляемым участником. Критерий считается выполненным, если имеется описание API-продукта	1
6	Документационное обеспечение продукта	Определяется по сведениям, предоставляемым участником. Критерий считается выполненным, если эксперты оценили достаточную полноту эксплуатационной и пользовательской документации	1
7	Использование протоколов и форматов данных, определенных международными рекомендациями	Определяется по сведениям, предоставляемым участником. Критерий считается выполненным, если эксперты определили факт использования в продукте международных стандартов	1

Технические критерии			
8	Наличие GUI-интерфейса пользователя	Определяется по сведениям, предоставляемым участником. Критерий считается выполненным, если эксперты определили факт использования в продукте GUI-интерфейса пользователя (администратора)	0,5
9	Наличие встроенной системы помощи в интерфейсе пользователя	Определяется по сведениям, предоставляемым участником. Критерий считается выполненным, если эксперты определили факт наличия встроенной системы помощи в GUI-интерфейсе пользователя (администратора)	0,25
10	Наличие программы установки компонентов продукта	Определяется по сведениям, предоставляемым участником. Критерий считается выполненным, если эксперты определили факт наличия программы установки (инсталлятора) программных компонент продукта	0,5
11	Функциональность	Определяется по сведениям, предоставляемым участником. Критерий считается выполненным, если эксперты оценили факт достижения реализованной функциональности, заявленной в документации	1
Маркетинговые критерии			
12	Тиражируемость продукта среди потребителей, осуществляющих деятельность в различных сферах хозяйственной деятельности	Определяется по списку потребителей, предоставляемому участником. Критерий считается выполненным, если имеется 3 сферы деятельности или более	1
13	Тиражируемость продукта среди потребителей, осуществляющих деятельность в сфере государственного управления	Определяется по списку потребителей, предоставляемому участником. Критерий считается выполненным, если имеется 1 потребитель или более из числа органов государственной власти (федерального или регионального уровня)	0,5

Критерии оценки ПО

Сравнительный подход

В ходе сравнительного оценивания определяются:

- элементы сравнения (по каким элементам будет осуществляться сравнение программного обеспечения с аналогами);
- степень и характер различий объекта сравнения от аналогичного программного обеспечения;
- рыночная стоимость программного обеспечения с обоснованным обобщением стоимости аналогов.

Элементами для сравнительного анализа при оценке разработанного программного обеспечения могут выступать характеристики программного обеспечения, спрос на продукцию, производство и реализация которой может осуществляться с использованием ПО, срок возможного использования оцениваемого объекта и т.д

Доходный подход

Главные методы доходного подхода:

- дисконтирование денежных потоков;
- прямая капитализация;
- освобождение от роялти;
- избыточной прибыли;
- дробления прибыли.

Выгоды от использования программного обеспечения могут иметь следующие формы:

- снижение расходов на производство товаров (услуг, работ) или/и их реализацию;
- повышение стоимости продукции;
- увеличение объема выпуска продукции или реализуемых услуг и повышение эффективности;
- диверсификация и снижение рисков;
- сокращение расходов на налоги и другие обязательные платежи.

Затратный подход

При использовании данного подхода к оценке программного обеспечения применяются следующие методы:

- стоимости создания программного обеспечения
- выигрыша в себестоимости

Влияющие на стоимость ПО факторы

При оценке ПО учитываются также факторы, которые прямо или опосредованно могут повлиять на его стоимость. Это:

- уникальность программного обеспечения;
- широта возможных отраслей использования;
- объем и структура инвестиций для внедрения программного обеспечения;
- возможность и степень правовой защиты;
- наличие и характер рисков от использования программного обеспечения в разных отраслях;
- текущее использование программного обеспечения

Лекция 3.

Цели, задачи, этапы, объекты и планирование ревьюирования

Е.В. Поколодина, Н.А. Долгова, Д.В Ананьев

«Ревьюирование программных модулей», глава 1, п.1.2

- **Инспекция кода (Code review)** – систематический и периодический анализ программного кода, направленный на поиск необнаруженных на ранних стадиях разработки программного продукта ошибок, а также, на выявление некачественных архитектурных решений и критических мест в программе
- **Рецензирование кода**, **обзор кода** (англ. *code review*) или **инспекция кода** (англ. *code inspection*) — систематическая проверка исходного кода программы с целью обнаружения и исправления ошибок, которые остались незамеченными в начальной фазе разработки

Задачи и цели проведения инспекций программного кода

- Целью обзора является улучшение качества программного продукта и совершенствование навыков разработчика.

Поиск ошибок не единственная задача обзора кода. Помимо этого, обзор кода имеет еще несколько положительных свойств:

- Улучшается архитектура приложения за счет того, что каждую часть системы продумали как минимум 2 человека.
- Программист изначально мотивируется писать более качественный код, зная, что его будут просматривать.
- Распространяются знания о проекте среди команды.
- Происходит обмен программистским опытом.
- Вырабатывается единый стиль кодирования в команде.

Можно выделить следующие виды обзоров кода:

1. Формальная инспекция кода.
2. Неформальная инспекция кода (другое название – анализ кода).
3. Чтение кода.
4. Парное программирование.

Формальная инспекция кода

Формальная инспекция кода представляет собой формализированную процедуру просмотра кода

<u>Достоинства:</u>

- Очень высокая эффективность.
- Благодаря составляемому списку ошибок легко проверить их устранение.
- Инспекционные отчеты можно использовать в дальнейшем, например, для анализа характерных проблем.

<u>Недостатки:</u>

- Сложная формальная процедура, требующая времени.
- Отвлечение как минимум 3-х человек (координатор, автор кода и инспектор) от их основной работы.
- Большое психологическое давление на автора кода.

Неформальная инспекция кода

Неформальная инспекция не имеет четких правил.

Достоинства:

- Малые затраты времени.
- Простой процесс не требующий формальных процедур.

Недостатки:

- Невысокая эффективность за счет поверхностного знакомства проверяющего с кодом.
- Для проверки приходится отвлекать кого-нибудь от основной работы, что может сильно раздражать.
- Критика кода может плохо восприниматься автором, причем как обоснованно (например, из-за незначащих придирок проверяющего), так и необоснованно (например, автору трудно признавать свои ошибки).

Чтение кода

Чтение кода – это самостоятельное изучение разработчиком чужого кода без присутствия автора. Данная практика является самой простой и распространенной.

<u>Достоинства:</u>

- Простота.
- Высокая доступность – не требуется синхронизация во времени и пространстве.

<u>Недостатки:</u>

- Медленная обратная связь – могут потребоваться дополнительные комментарии к коду, которые нельзя быстро получить, а иногда даже быстрее самому исправить дефект, чем сообщить об этом автору.

Парное программирование

Является экстремальным методом обзора кода – обзор, осуществляемый постоянно: два разработчика за одним компьютером, за одним комплектом мыши и клавиатуры, вместе решают одну задачу

Достоинства:

- Высокая эффективность
- Высокая концентрация на работе
- Естественное ограничение количества одновременно разрабатываемых командой задач
- Повышение «командного духа»
- Отлично подходит для обучения новичков

Недостатки:

- Падение общей производительности,
- Требуется синхронизация рабочего графика
- Повышенная утомляемость за счет постоянной высокой концентрации на работе
- Неэффективно для выполнения рутинных задач
- Трудно синхронизировать темп разработчиков

Формальная инспекция кода

Процесс формальной инспекции состоит из пяти фаз:

1. инициализация
2. планирование
3. подготовка (экспертиза)
4. обсуждение
5. завершение

Формальная инспекция кода

1. Инициализация

Руководитель проекта или его заместитель запрашивает из базы, хранящей все данные проекта (например, из системы конфигурационного управления), список объектов, готовых к инспекции, выбирает объект инспекции, затем назначает участников формальной инспекции: автора, ведущего и одного или нескольких инспекторов. Ведущий также может выполнять роль инспектора; остальные участники выполняют только одну роль. На роль ведущего или инспектора не допускается назначать сотрудников, участвовавших в разработке объекта инспекции.

Формальная инспекция кода

2. Планирование

Ведущий меняет статус инспектируемых документов, чтобы отметить факт начала инспекции и ограничить доступ к инспектируемой документации. Во время инспекции изменение документов невозможно, а соответствующий статус сохраняется до конца инспекции. Далее будем называть этот статус Review.

Время, отводимое на этап подготовки, не может быть менее одного часа.

Ведущий должен определить дату, время и место обсуждения, если оно будет проходить в форме собрания.

Формальная инспекция кода

3. Подготовка

Инспекторы должны извлечь из базы данных проекта исходные и инспектируемые документы, используя указанные в бланке идентификаторы и номера версий. При этом инспекторы должны убедиться, что все документы находятся в соответствующем состоянии.

В ходе подготовки инспекторы детально изучают инспектируемые документы, руководствуясь списком контрольных вопросов. Обнаруженные несоответствия должны быть точно локализованы, сформулированы и записаны.

Формальная инспекция кода

4. Обсуждение

Обсуждение проводится в форме одного или нескольких собраний, каждое из которых продолжается не более двух часов.

Для проведения собрания необходимо присутствие ведущего, хотя бы одного из инспекторов и автора.

В ходе обсуждения ведущий синхронизирует работу участников.

В ходе обсуждения необходимо в бланке инспекции проставить ответы на контрольные вопросы и зафиксировать замечания.

Формальная инспекция кода

5. Завершение

По окончании обсуждения, инспекторы сдают ведущему свои рабочие материалы, которые включают в себя распечатки инспектируемых документов с пометками и бланки инспекции. Ведущий складывает эти материалы в прозрачную папку вместе с экземпляром бланка инспекции, заполненным в ходе обсуждения, причем титульный лист бланка инспекции должен лежать сверху, чтобы можно было по нему идентифицировать папки.

После собрания ведущий изменяет статус инспектируемых документов в базе данных проекта в соответствии с принятым решением - либо им присваивается статус *Принят* либо *Переработать* (в последнем случае необходима повторная инспекция)

Лекция 4.

Исследования программного кода

Е.В. Поколодина, Н.А. Долгова, Д.В Ананьев

«Ревьюирование программных модулей», глава 1, п.1.5

Методы анализа программного кода

Выделяют несколько методов анализа кода для его проверки:

- статический
- динамический
- гибридный

Статический анализ (статистические анализаторы)

решает ряд важных задач:

- выявление различных типов ошибок и возможных уязвимостей программного кода;
- подсчет метрик. Метрика ПО - мера, позволяющая получить численное значение некоторого свойства ПО или его спецификаций;
- формирование рекомендаций по оформлению кода.

Достоинства методов статического анализа кода:

- полное покрытие кода
- не зависит от используемого компилятора и среды
- можно легко и быстро обнаруживать опечатки

Недостатки методов статического анализа кода:

- недостаточно хорошая диагностика утечек памяти и параллельных ошибок
- корректный код при анализе тоже может попасть в список недостатков (ложнопозитивные события)

Динамический анализ

Динамический анализ кода решает ряд важных задач, в частности позволяет:

- имитировать поведение пользователя и рассматривать работу программы в различных ситуациях и с различными наборами данных, корректными и некорректными;
- находить программные ошибки, к которым можно отнести не только ошибки кода, но и ошибочные результаты и вычисления, полученные при выполнении программы или ее модулей;
- обнаружить наличие уязвимостей в программе;
- оценить используемые ресурсы, время выполнения программы в целом, время выполнения отдельных модулей программы, количество внешних запросов, количество используемой оперативной памяти и других ресурсов;
- получить определенные наборы метрик.

Достоинства динамического анализа кода:

- обычно не присутствует появление ложных срабатываний;
- обнаруженная ошибка является фактической, а не возможной;
- позволяет протестировать программы с закрытым кодом.

Недостатки динамического анализа кода:

- обнаруживает дефекты только при тестировании с определенными наборами данных, в частях программы, которые тестирование не охватывает, ошибки могут быть не обнаружены;
- требуются значительные вычислительные ресурсы для проведения тестирования;
- только один путь выполнения может быть проверен в каждый конкретный момент времени. Требуется большое количество тестовых запусков для большей полноты тестирования;
- при тестировании на реальном процессоре исполнение некорректного кода может привести к непредсказуемым последствиям.

Методы исследования кода

- Отладка
- Трассировка
- Ревьюирование
- Тестирование
- Профилирование
- Обратное проектирование
- Дизассемблирование
- Использование анализаторов трафика (снифферов)
- Использование утилит для динамического анализа кода
- Экспертиза ПО

Механизмы и контроль внесения изменений в код

Е.В. Поколодина, Н.А. Долгова, Д.В Ананьев

«Ревьюирование программных модулей», глава 1, п.1.6

Система контроля версий позволяет:

- команде программистов работать с одним и тем же хранилищем исходного кода, называемым репозиторием проекта;
- хранить несколько версий одних и тех же файлов проекта;
- вести учет и контроль версий;
- получать программисту личную копию хранилища и работать с этой копией на своем локальном компьютере;
- возвращаться к более ранним версиям файлов кода, получать актуальные версии файлов и решать ряд других важных задач.

Существуют разные типы СКВ со своими алгоритмами и возможностями работы:

- локальная;
- централизованная;
- децентрализованная (распределенная).

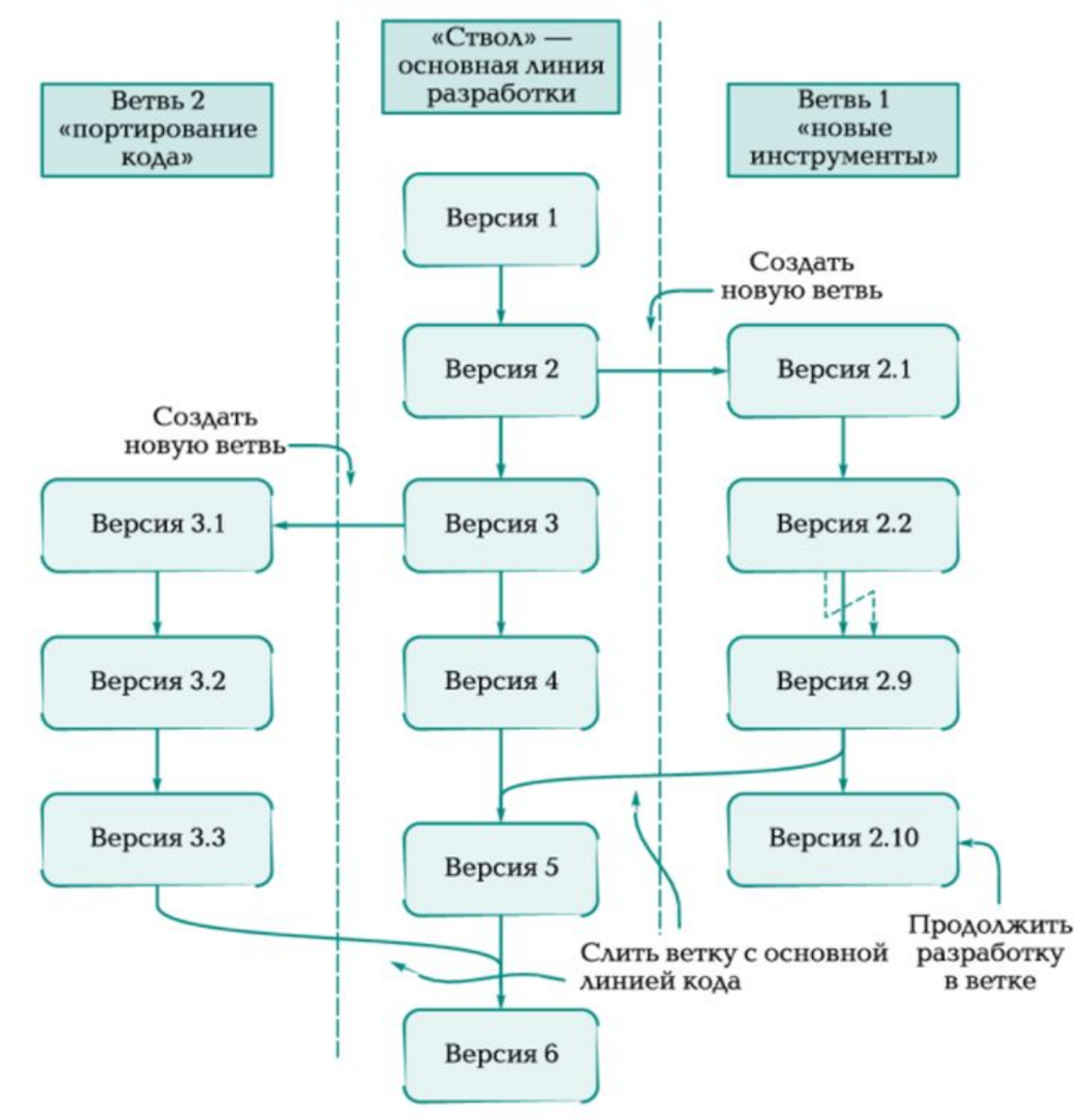
При организации репозиториев соблюдаются определенные принципы:

- репозиторий содержит дерево всех файлов и директорий проекта, хранит главную копию всех файлов исходного кода и вспомогательные файлы, и документы;
- при записи файла в репозиторий сохраняется его предыдущее состояние, и только после этого происходит обновление файла;
- внесенные изменения становятся доступными для других разработчиков команды;
- при чтении файла из репозитория по умолчанию предоставляется последняя версия файла со всеми сделанными изменениями.

При работе с репозиторием СКВ реализует ряд механизмов:

- контроль за исходным кодом
- управление версиями
- управление конфигурациями
- контроль доступа
- ветвление

Механизм ветвления проекта в СКВ



Лекция 5.

Обзор утилит для review

Е.В. Поколодина, Н.А. Долгова, Д.В Ананьев

«Ревьюирование программных модулей», глава 2, п.2.1

Рецензирование кода, **обзор кода** (англ. *code review*) или **инспекция кода** (англ. *code inspection*) — систематическая проверка исходного кода программы с целью обнаружения и исправления ошибок, которые остались незамеченными в начальной фазе разработки

Утилита - компьютерная программа, расширяющая стандартные возможности оборудования и операционных систем и выполняющая узкий круг специфических задач

- **DeepCode** - программа применяется для обнаружения и исправления ошибок в коде
- **Node.js** - предназначена для поиска уязвимостей в модулях
- **PVS-Studio** - инструмент для выявления ошибок и потенциальных уязвимостей в исходном коде программ, написанных на языках C, C++ и C#
- **BlameNotifier** - инструмент позволяет рассылать письма разработчикам об ошибках, которые PVS-Studio нашел во время ночного прогона

!!!!Самостоятельно учебник, глава 2, п.2.1, стр.82-87

1) коллаборационист

ключевая характеристика:

- Установите правила проверки и автоматические уведомления, чтобы гарантировать, что проверки будут завершены вовремя.
- Пользовательские шаблоны проверки уникальны для Collaborator. Установите настраиваемые поля, контрольные списки и группы участников, чтобы адаптировать экспертные оценки к идеальному рабочему процессу вашей команды.
- Легко интегрируется с 11 различными SCMs, а также IDE, такими как Eclipse & Visual Studio
- Создавайте пользовательские отчеты о проверке, чтобы повысить эффективность процесса и упростить аудит.
- Проводите экспертные обзоры документов в одном и том же инструменте, чтобы команды могли легко согласовываться с требованиями, изменениями дизайна и нагрузками соответствия.

Официальный сайт: https://smartbear.com/product/collaborator/free-trial/

2) CodeScene

ключевая характеристика:

- Автоматический Просмотр кода комментарии на запросы вытягивания.
- Качественные ворота для CI / CD.
- А целенаправленный рабочий процесс для улучшения планирования.
- Контролируйте технический долг и код здоровья.
- Работает с любым git хостингом.
- Интегрируется с Jira для отслеживания тенденций в производительности доставки.
- CodeScene доступен как в локальной, так и в размещенной версии.

Официальный сайт: https://empear.com/

CodeScene™

Powered by Empear

3) Визуальный Эксперт

ключевая характеристика:

- Находите неиспользуемые объекты, индексы или таблицы
- Определите отсутствующие индексы, ухудшающие время выполнения запроса.
- Проверьте соглашения об именовании.
- Генерируйте метрики кода: строки кода, количество объектов, переменные и т.д.
- Находите негабаритные объекты.
- Найдите пустые функции, не имеющие активного кода.

Visual Expert toolbox также включает в себя CRUD matrix generation, автоматическую документацию по коду, электронные диаграммы, синхронизированные с кодом, анализ производительности кода и многое другое.

Официальный сайт:

https://www.visual-expert.com/EN/lp-ve-download-source_adv914ve.html?single

VISUAL EXPERT

4) Codebrag

ключевая характеристика:

- Codebrag-это простой, легкий, бесплатный и **открытый инструмент для просмотра кода**, который делает обзор интересным и структурированным.
- Codebrag используется для решения таких проблем, как неблокирующий обзор кода, встроенные комментарии и лайки, интеллектуальные уведомления по электронной почте и т. д.
- С Codebrag можно сосредоточиться на рабочем процессе, чтобы выяснить и устранить проблемы наряду с совместным обучением и совместной работой.
- Codebrag помогает в поставке расширенного программного обеспечения, используя его гибкую проверку кода.
- Лицензия для Codebrag с открытым исходным кодом поддерживается **AGPL** .

Официальный сайт: http://codebrag.com/

5) Геррит

Gerrit Code Review

ключевая характеристика:

- Gerrit-это **бесплатный веб-инструмент для проверки кода**, используемый разработчиками программного обеспечения для проверки своего кода в веб-браузере и отклонения или утверждения изменений.
- Gerrit может быть интегрирован с Git, который является распределенной системой управления версиями.
- Gerrit обеспечивает управление репозиторием для Git.
- Используя Gerrit, участники проекта могут использовать рационализированный процесс просмотра кода, а также чрезвычайно настраиваемую иерархию.
- Gerrit также используется при обсуждении нескольких подробных сегментов кода и повышении правильности вносимых изменений.

Официальный сайт: https://www.gerritcodereview.com/

6) Codestriker

ключевая характеристика:

- Codestriker-это открытое и бесплатное онлайн-приложение для просмотра кода, которое помогает в совместном просмотре кода.
- С помощью Codestriker можно записывать вопросы, комментарии и решения в базу данных, которая в дальнейшем может быть использована для проверок кода.
- Codestriker поддерживает традиционный просмотр документов. Его можно интегрировать с ClearCase, Bugzilla, CVS, etc.
- Codestriker имеет лицензию под GPL.

Официальный сайт: http://codestriker.sourceforge.net/

Codestriker
Web-based code reviewing

7) Родекод

ключевая характеристика:

- Rhodecode-это инструмент управления открытым исходным кодом, защищенным и инкорпорированным корпоративным исходным кодом.
- Rhodecode служит интегрированным инструментом для Git, Subversion и Mercurial.
- Основными функциями Rhodecode являются совместная работа с командой, управление репозиторием и безопасность и аутентификация кода.
- Rhodecode имеет 2 выпуска, Community Edition (CE) который является бесплатным и открытым исходным кодом и Enterprise Edition (EE) лицензируется на одного пользователя.
- Rhodecode автоматизирует рабочие процессы для быстрого выполнения.

Официальный сайт: https://rhodecode.com/

RhodeCode

8) Phabricator

ключевая характеристика:

- Инструмент проверки кода от Phabricator suite называется “дифференциальный". Он используется для минимизации усилий, необходимых для создания кода наилучшего качества.
- Phabricator имеет два типа рабочих процессов проверки кода, а именно “предварительный толчок”, также называемый “обзором”, и “пост-толчок”, называемый “аудит”.
- Phabricator может быть интегрирован с Git, Subversion и Mercurial.

Официальный сайт: https://www.phacility.com/phabricator/

Phabricator

9) Тигель

ключевая характеристика:

- Тигель гибкое применение которое приспосабливает обширный ряд подходов к работы и размеров команды.
- Crucible-это легкий инструмент для проверки однорангового кода, который используется в обзорах до и после фиксации.
- Обзор кода стал легким для SVN, Perforce и CVS и т. д. С помощью Crucible.

Официальный сайт: https://www.atlassian.com/software/crucible

Crucible

10) Veracode

ключевая характеристика:

- Veracode используется разработчиками при создании защищенного программного обеспечения путем сканирования двоичного кода или байтового кода вместо исходного кода.
- С помощью Veracode можно идентифицировать неправильные зашифрованные функциональные возможности, вредоносный код и бэкдоры из исходного кода.
- Veracode может просмотреть большое количество кода и сразу же возвращает результаты.
- Для использования Veracode нет необходимости покупать какое-либо программное или аппаратное обеспечение, вам просто нужно оплатить необходимые службы analysis services.

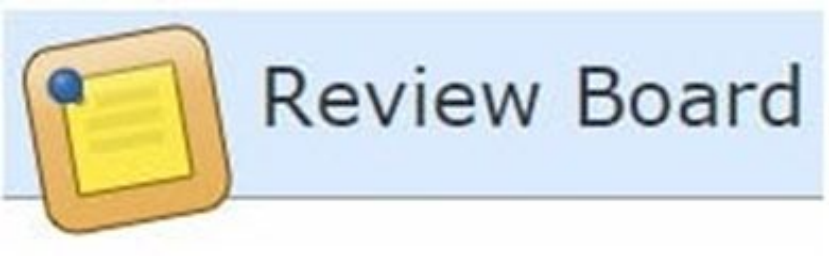
Официальный сайт: https://www.veracode.com/

11) Наблюдательный Совет

ключевая характеристика:

- Используя обзорную доску для обзора кода можно сэкономить деньги и время. Сэкономленное время можно использовать для концентрации на создании отличного программного обеспечения.
- Обзорная доска может быть интегрирована с ClearCase, CVS, Perforce, пластиком и т. д.
- В коде review by Review Board tool, код является синтаксис выделен, что делает его чтение быстрее.
- Обзорная доска поддерживает обзоры до фиксации и обзоры после фиксации.

Официальный сайт: https://www.reviewboard.org/



ДОПОЛНИТЕЛЬНЫЕ ИНСТРУМЕНТЫ ДЛЯ РАССМОТРЕНИЯ

- 12) бармен http://getbarkeep.org/
- 13) JArchitect https://www.jarchitect.com/
- 14) Инструмент Проверки Кода http://codereviewtool.com/
- 15) рецензируемый https://reviewable.io/
- 16) Ритвельд https://codereview.appspot.com/
- 17) Плагин Peer Review https://trac-hacks.org/wiki/PeerReviewPlugin

Лекция 6.

Предпроцессинг кода

Е.В. Поколодина, Н.А. Долгова, Д.В Ананьев

«Ревьюирование программных модулей», глава 2, п.2.2

Предпроцессинг (препроцессинг)– это самая первая стадия компиляции программы

<u>Препроцессор</u> — это специальная программа, являющаяся частью компилятора языка Си. Она предназначена для предварительной обработки текста программы. Препроцессор позволяет включать в текст программы файлы и вводить макроопределения.

Работа препроцессора осуществляется с помощью специальных директив (указаний). Они отмечаются знаком решетка #. По окончании строк, обозначающих директивы в языке Си, точку с запятой можно не ставить.

Основные функции препроцессора

- замена соответствующих диграфов и триграфов на эквивалентные символы «#» и «\»;
- удаление экранированных символов перевода строки;
- замена строчных и блочных комментариев пустыми строками (с удалением окружающих пробелов и символов табуляции);
- вставка (включение) содержимого произвольного файла (#include);
- макроподстановки (#define);
- Условная компиляция (#if, #ifdef, #elif, #else, #endif);
- вывод сообщений (#warning, #error)

Условная компиляция позволяет выбрать код для компиляции в зависимости от:

- модели процессора (платформы);
- разрядности адресов;
- размерности типов;
- наличия/отсутствия поддержки расширений языка;
- наличия/отсутствия библиотек и/или функций;
- особенностей поведения конкретных функций;
- и другого.

Этапы работы препроцессора:

- лексический анализ кода (синтаксический анализ не выполняется);
- обработка директив;
- выполнение подстановок:
- диграфов и триграфов;
- комментариев;
- директив;
- лексем, заданных директивами

Директивой (командной строкой) препроцессора называется строка в исходном коде, имеющая следующий формат: #ключевое_слово параметры:

- символ #;
- ноль или более символов пробелов и/или табуляции;
- одно из предопределённых ключевых слов;
- параметры, зависимые от ключевого слова

- если ключевое слово не указано, директива игнорируется;
- если указано несуществующее ключевое слово, выводится сообщение об ошибке и компиляция прерывается

Список ключевых слов:

- define — создание константы или макроса;
- undef — удаление константы или макроса;
- include — вставка содержимого указанного файла;
- if — проверка истинности выражения;
- ifdef — проверка существования константы или макроса;
- ifndef — проверка не существования константы или макроса;
- else — ветка условной компиляции при ложности выражения if;
- elif — проверка истинности другого выражения; краткая форма записи для комбинации else и if;
- endif — конец ветки условной компиляции;
- line — указание имени файла и номера текущей строки для компилятора;
- error — вывод сообщения и остановка компиляции;
- warning — вывод сообщения без остановки компиляции;
- pragma — указание действия, зависящего от реализации, для препроцессора или компилятора;

Интеграция в IDE

Е.В. Поколодина, Н.А. Долгова, Д.В Ананьев

«Ревьюирование программных модулей», глава 2, п.2.2

Интегри́рованная среда́ разрабо́тки, ИСР (англ. Integrated development environment — IDE), **Единая среда разработки, ЕСР** — это комплекс программных средств, используемый программистами для разработки программного обеспечения (ПО).

<u>Среда разработки включает в себя:</u>

- текстовый редактор,
- транслятор (компилятор и/или интерпретатор),
- средства автоматизации сборки,
- отладчик

Лекция 7.

Ревью кода системы средствами Git

Процессы ревью в Github и аналогах построены вокруг вносимых изменений, а в нашем случае комментарии нужно дать к состоянию всего кода системы на момент комментирования.

Как это сделать средствами самого Git: зафиксировать состояние в ветке для ревью, затем в merge request к этой ветке оставить свои замечания.

Проблематика

Представьте ситуацию: вам передают репозиторий с кодом и просят вынести свое мнение о нем. Обычно в подобных случаях замечания составляются в отдельном документе/таске/страничке в конфлюенс *(**Confluence** - место для совместной работы команды и обмена знаниями, для создания и обсуждения Ваших файлов, идей, набросков, спецификаций, макетов, диаграмм и проектов)* и т.п., что не очень удобно так как:

1. Замечания могут устареть уже в процессе написания, так как разработка может продолжаться.
2. Сложно ссылаться на отдельные участки кода, так как приходится постоянно переключаться между документом и кодом.
3. В отрыве от кода документ затеряется с довольно высокой вероятностью.

Метод ревью кода системы

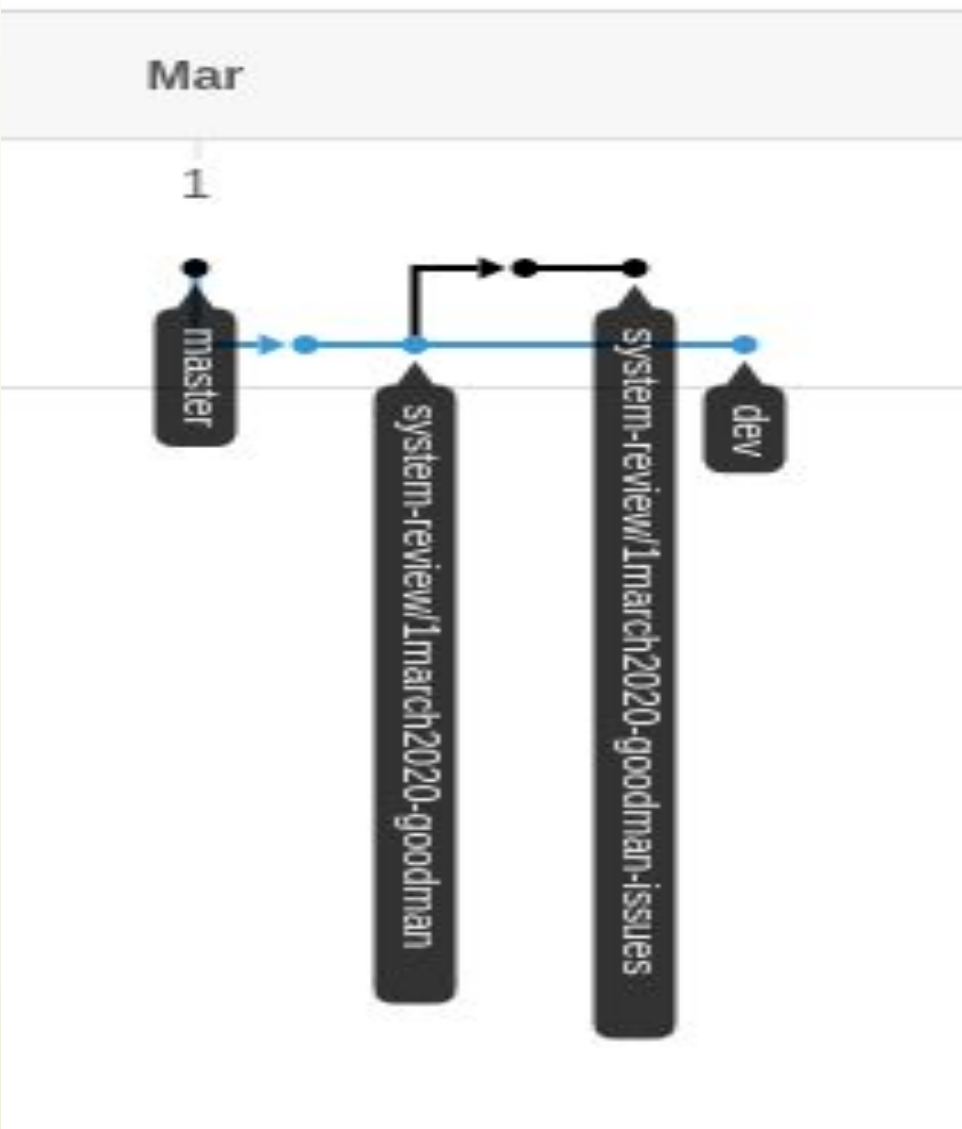
Итак, нам нужно проделать следующее: зафиксировать состояние в ветке для ревью, затем в merge request к этой ветке оставить свои замечания.

На примере подготовленного для заметки репозитория https://github.com/oktend/system-review-example проделаем эти шаги:

1. Найдем состояние в репозитории для ревью (на момент ревью это был последний коммит в dev): https://github.com/oktend/system-review-example/commit/0514531a35edf19e7032eb49f45a98d019f83efe
2. Ветвим от выбранного состояния ветку для нашего системного ревью, например "system-review/1march2020-goodman": https://github.com/oktend/system-review-example/tree/system-review/1march2020-goodman

4. Создаем от вновь созданной ветки еще одну ветку, в которой будем собирать замечания, например "1march2020-goodman-issues": https://github.com/oktend/system-review-example/tree/system-review/1march2020-goodman-issues
5. Вносим в эту ветку удобным нам способом наши замечания, как прямо в код, так и в отдельные документы.
6. Создаем merge request (может называться pull request) к ветке для ревью "system-review/1march2020-goodman-issues" -> "system-review/1march2020-goodman": https://github.com/oktend/system-review-example/pull/1/files

Теперь наши ветки выглядят примерно так:



https://github.com/oktend/system-review-example/network

Результат

В созданном merge request можно увидеть все собранные в ходе ревью замечания, даже обсудить их.

Состояние, для которого были выдвинуты замечания будет зафиксировано пока ветку явно не удалят.

Замечания можно делать как в отрыве от кода:
https://github.com/oktend/system-review-example/blob/c80b03710059b235347ec781bf08dca9c0e68f7d/review-1march2020-goodman.md

так и в контексте кода:
https://github.com/oktend/system-review-example/blob/c80b03710059b235347ec781bf08dca9c0e68f7d/foo.js

Замечания можно просматривать в веб-интерфейсе github (или аналогов), в IDE, или средствами самого git.

К ревью можно будет вернуться в будущем сохранив замечания и контекст, в котором они были выдвинуты.

Лекция 8.

Как правильно делать код-ревью

Терминология:

- **CL**: «changelist» — список изменений кода, отправленный в систему контроля версий на ревью.

Аналоги:

- PR: «Pull Request» в GitHub
- MR: «Merge Request» в GitLab

1. Стандарты код-ревью

1.1. Менторство

1.2. Принципы

1.3. Разрешение конфиликтов

2. На что обращать внимание в ревью

2.1. Дизайн (структура

2.2. Функциональность)

2.3. Сложность

2.4. Тесты

2.5. Наименования

2.6. Комментарии

2.7. Стиль

2.8. Документация

2.9. Каждая строчка

2.10. Контекст

2.11. Позитивные моменты

Итоги

Самое важное при проведении ревью:

- Код хорошо спроектирован,
- Функционал работает правильно,
- Изменения в пользовательском интерфейсе требуют особого внимания: помимо чистоты кода необходимо убедиться в том, что для пользователя изменения выглядят как задумано,
- Параллельно выполняемый код исполняется безопасно,
- Код не переусложнен,
- Нет надуманного и ненужного на данный момент функционала,
- На код написаны тесты
- Тесты хорошо спроектированы,
- Использованы корректные имена для файлов, классов, методов, переменных и тд
- Оставлены понятные и нужные комментарии, по большей части поясняющие причины решений, а не суть (суть должна быть понятна из самого кода),
- Код задокументирован (g3doc),
- Код соответствует нашим стайл-гайдам.