

Методические рекомендации по изучению темы «Инспекция программного кода»

Настоящие методические рекомендации предназначены для студентов государственного профессионального образовательного учреждения «Печорский промышленно-экономический техникум» очной формы обучения специальности *09.02.07 Информационные системы и программирование*.

Методические рекомендации представляют собой сборник материалов, для изучения темы «Инспекция программного кода» и состоят из двух частей: теоретический материал и серия практических работ по инспекции кода.

Теоретический материал представляет собой обзор существующих методов инспекции программного кода и описание общих подходов к инспекции, а также описание существующих правил написания программного кода. Серия практических работ позволяет студентам провести инспекцию собственного программного кода, разработанного на занятиях по *ОП.05 Основы алгоритмизации и программирования* - программа вычисления площади простых геометрических фигур – **Вычисление площади фигуры**.

Анализ программного кода, выполненный студентами в рамках практических работ не является законченным и может быть продолжен на дополнительных занятиях или самостоятельно.

Теоретический материал по теме Инспекция кода

Цель: изучение понятия инспекции программного кода.

Задачи:

- расширить знания о существующих стандартах кодирования;
- изучить понятие инспекции кода и основные подходы к проведению инспекции;
- изучить правила именования объектов кода согласно различных нотаций;
- совершенствовать и развивать коммуникативно-познавательные умения, направленные на систематизацию и углубление знаний.

После изучения материала студент:

должен знать:

- понятие инспекции программного кода;
- подходы к проведению инспекции и ее виды;
- правила именования объектов кода согласно различных нотаций.

Должен уметь:

- различать виды инспекции программного кода;
- определять способ проведения инспекции исходя из конкретной ситуации;
- различать нотации именования объектов кода.

Понятие инспекции

Инспекция кода (ревизия кода, рецензия кода, Code review) – это систематическая проверка исходного кода программы. Цель проверки – обнаружение и исправление ошибок, которые были пропущены, остались незамечены при разработке. Результат инспекции как правило – улучшение качество ПО и навыки разработчика.

В ходе инспекции, когда могут быть найдены, с последующим устранением, такие уязвимости, как ошибки в форматировании строк, утечка памяти, переполнение буфера, что улучшает безопасность ПО. Системы контроля версий также дают возможность проведения совместной инспекции исходного кода. Помимо этого, есть инструменты, которые предназначены именно для совместной инспекции.

Обзор кода является одним из наиболее эффективных методов поиска и устранения дефектов программы. Обзоры проводятся человеком, что позволяет находить широкий класс ошибок, в том числе с трудом детектируемых или вообще не детектируемых автоматическими средствами.

Безусловно, обзор кода, не отменяет использование анализаторов кода или других методик обнаружения ошибок, например, unit-тестирования. К сожалению, не существует метода, который один обеспечил бы обнаружение всех дефектов программы (в исследованиях

эффективность обзора кода обычно оценивается как 30-50% обнаруженных ошибок в приложении).

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.

За все это приходится платить временем, которое могло быть потрачено на кодирование. Тем не менее, на проектах, рассчитанных хоть на какую-либо перспективу, затраченное время в будущем вернется сторицей за счет создания изначально качественного продукта.

Виды инспекции кода

Можно выделить следующие виды обзоров кода:

- Формальная инспекция кода.
- Неформальная инспекция кода (другое название – анализ кода).
- Чтение кода.
- Парное программирование.

Формальная инспекция кода

Формальная инспекция кода представляет собой как следует из названия формализованную процедуру просмотра кода. По МакКоннеллу она выглядит примерно так.

Координатор инспекции назначает дату инспекционного собрания и инспекторов, которые должны до собрания самостоятельно изучить инспектируемый фрагмент кода. В назначенное время собираются координатор инспекции, секретарь, автор кода и инспекторы. Кто-то из инспекторов или руководитель начинают читать код строчку за строчкой (для удобства желательно его заранее распечатать вместе с номерами строк). Инспектора указывают на найденные ими проблемы, автор кода отвечает на вопросы инспекторов – если все соглашаются, что ошибка действительно имеется, то секретарь ее записывает. Координатор инспекции следит за процессом в целом, например, за тем, чтобы обсуждения одной ошибки не затягивались. По окончании инспекции составляется отчет с ее результатами.

Достоинства:

- Очень высокая эффективность.

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

Недостатки:

- 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 в отличие от формальной не имеет четких правил. Например, это может происходить так: автор кода, перед его публикацией, подзывает первого попавшегося разработчика за свой компьютер, где показывает и рассказывает то, что написал. Проверяющий пытается вникнуть в написанное, задает вопросы и высказывает свои соображения. Безусловно, при таком способе эффективность будет не сильно высокая, зато такая инспекция отнимает мало времени.

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 незначительных 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 из описанных здесь – думаю, любому разработчику так или иначе приходило читать чужой код. Не говоря уж про мир Open Source, где это зачастую единственный доступный метод обзора кода.

Достоинства:

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

Недостатки:

- 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

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. Широкое распространение парное программирование получило после появления методологии экстремального программирования, хотя активно использовалось и до этого. Часто парное программирование используется спонтанно: думаю, многим приходилось подходить к другому разработчику за компьютер, чтобы помочь решить сложную задачу.

Достоинства:

- Высокая эффективность, особенно в плане обмена опытом и распространения знания о проекте.
- Высокая концентрация на работе – работая в паре, разработчики гораздо меньше отвлекаются на посторонние вещи.
- Естественное ограничение количества одновременно разрабатываемых командой задач – сконцентрированность на меньшем количестве задач обеспечивает более качественную и быструю их реализацию, что позволяет непрерывно поставлять новые версии продукта.
- Нет психологических вопросов присущих инспекциям – оба автора кода, и оба одновременно его же инспекторы, предложения по улучшению воспринимаются именно как предложения, а не критика.
- Повышение командного духа – успехи, достигнутые в паре, больше объединяют людей, чем индивидуальные достижения.
- Отлично подходит для обучения новичков.

Недостатки:

- Падение общей производительности, два программиста заняты одной задачей, вместо разработки двух задач – данное утверждение достаточно спорное, согласно ряду исследований, программисты, работающие в паре, имеют всего на 15% процентов меньшую производительность, чем два программиста работающих по отдельности.
- Требуется синхронизация рабочего графика – трудно работать в паре, когда партнеры в разное время ходят на обед или приходят на работу.
- Повешенная утомляемость за счет постоянной высокой концентрации на работе – для программистов, работающих в паре, даже может иметь смысл делать рабочий день меньше стандартных 8-ми часов.
- Не все люди совместимы, а некоторые даже вообще не способны работать с кем-то вместе. На

самом деле, таких людей достаточно немного и большую часть проблем взаимодействия в

Специальность: 09.02.07 Информационные системы и программирование

Дисциплина: ОП.05 Основы алгоритмизации и программирования

ОП.08 Основы проектирования баз данных

МДК 01.01 Технология разработки программного обеспечения

Преподаватель: Кулик Е.В., Михеева Т.А.

паре можно преодолеть, выполняя ряд правил (*например*), а также с накоплением опыта работы в паре.

- Неэффективно для выполнения рутинных задач – в этом случае разработчик, не владеющий клавиатурой, будет просто скучать.
- Трудно синхронизировать темп разработчиков уровень опыта и знаний которых сильно различается – для эффективной работы от более опытного разработчика требуются терпение и некоторые наставнические навыки.

Как можно заметить некоторые недостатки вытекают из достоинств. В этом нет ничего удивительного – парное программирование методика достаточно не простая, как может показаться с первого взгляда, и не все ее свойства очевидны. Поэтому, начав работать в парах, не стоит ожидать моментальных результатов. Только после получения некоторого опыта парное программирование начнет приносить плоды, и более того – можно будет добиться минимизации некоторых негативных эффектов.

Для распределенной команды практически единственный доступный метод обзора это **чтение кода**. Несмотря на то, что существуют системы для удаленного парного программирования, чисто по психологическим причинам, они не так комфортны как программирование за одним компьютером. Кроме того, чтение кода можно практиковать и в нераспределенных командах, но когда стоит выбор гадать над назначением какого-то куска кода или поинтересоваться об этом у автора, я бы выбрал последнее.

Формальные инспекции отлично подходят для обзора сложных или критичных участков кода – в этом случае временные затраты с лихвой окупятся результатом. Некоторые практикуют постоянное использование формальных инспекций, но мне трудно представить, как можно формальными инспекциями провести обзор всего имеющегося кода.

Парное программирование не зря является одним из принципов экстремального программирования. Его высокая эффективность и дополнительные бонусы, присущие только этому виду обзора кода, действительно позволяют его рекомендовать как основной способ разработки в команде (за исключением простейших или рутинных задач). Оптимизма добавляет и то, что с большей частью недостатков можно успешно бороться. Самой большой проблемой может стать попытка уговорить менеджмент использовать такой расточительный, по их мнению, метод – тут вам в помощь книги и статьи практиков экстремального программирования.

Вариантов достаточно – можно выбрать наиболее подходящий для себя способ. Кроме того, никто не запрещает комбинировать методики (например, одна часть команды работает в парах, другая часть – в одиночку, но код, написанный ими, обязательно инспектируется). Можно начинать с более простых методик и постепенно переходить к сложным – главное начать, а положительные эффекты, думаю, не заставят себя ждать.

На что смотреть во время инспекции

Архитектура / Дизайн

Принцип «одной ответственности». Идея в том, что у каждого класса должно быть только одно назначение. На самом деле реализовать это труднее, чем кажется. Если возникает нужда в союзе «и» при описании того, что делает метод, то это знак, что стоит разделить его на несколько более простых.

Принцип «Открыт/Закрыт». Если язык объектно-ориентированный, то открыты ли ваши объекты для расширения, но закрыты для модификации? Что произойдет, если нам нужно будет добавить еще один экземпляр класса *x*?

Дубликация кода. Если код повторяется три раза или больше, то необходимо вынести его в отдельный метод.

Слепой тест. Если расфокусировать зрение, то выглядит ли форма кода, на который вы смотрите, идентичной другим кускам кода? Если нет, то это может быть сигналом к тому, что код нуждается в рефакторинге.

При изменении кода всегда пытайтесь его улучшить. Обычно, если я исправляю какую-то часть кода, которая работает неправильно или выглядит некрасиво, у меня всегда возникает искушение просто исправить несколько строчек и на этом закончить. Я рекомендую не останавливаться на этом и сделать код лучше, чем он был.

Потенциальные баги. Просматривайте код на наличие ошибок-на-единицу, нарушений условий циклов и т.д.

Обработка ошибок. Достаточно ли хороша обработка ошибок? Введены ли кастомные исключения? Если да, то полезны ли они?

Эффективность. Если используется какой-либо алгоритм, эффективна ли его реализация? Например, пробег по всем ключам словаря — не лучший способ найти нужное значение.

Стиль

Имена методов. Давать имена различным вещам — одна из самых сложных задач в программировании. Если метод называется `get_message_queue_name()`, но делает что-то кроме этого, например, убирает HTML из входных данных, тогда это имя не подходит ему.

Имена переменных. `foo` и `bar` — не самые лучшие имена для структур данных. Переменная `ex` также не настолько красноречива, как `exception`. Будьте как можно лаконичнее. Говорящие имена переменных облегчат чтение кода в будущем.

Длина функций. Функция не должна быть длиннее 20 строк. Если я вижу метод больше 50 строк, я пытаюсь разбить его на два.

Длина классов. Классы должны быть меньше 300 строк, а в идеале — меньше 100. Скорее всего, если в вашем коде есть длинные классы, то их можно разбить на несколько, что облегчит понимание их предназначения.

Длина файла. Для Python 1000 строк в одном файле — предел. Если их становится больше, то, возможно, стоит разбить файл на несколько, с более специфичным предназначением. Чем больше файл, тем меньше читабельность кода в нем.

Документация. Сложные методы лучше задокументировать так, чтобы было понятно, за что отвечает каждый аргумент.

Закомментированный код. Стоит удалить закомментированные строки кода, чтобы не было ничего лишнего.

Количество аргументов в методе. Посмотрите, сколько аргументов передается в ваш метод. Больше трех? Это знак того, что они могут быть сгруппированы по-другому.

Читабельность. Легко ли разобраться в вашем коде? Часто ли вы делаете паузы во время чтения, чтобы разобраться в нем?

Тестирование

Полнота тестов. Насколько тесты продуманы? Могут ли они заставить ваш код упасть? Легко ли они читаются? Насколько они хрупки? Насколько они большие? Медленные ли они?

Тестирование на правильном уровне. Тестируется ли тот уровень приложения, который нужно тестировать для проверки функциональности? Гарри Бернардт рекомендует такое соотношение — 95% юнит-тестов и 5% интеграционных тестов.

Количество объектов-имитаций. Имитационные объекты хороши, но не стоит пихать их везде. Если в тесте их более трех штук, нужно его переписать. Либо тест, либо сама функция, для которой он предназначен, охватывают слишком большую часть кода. В обоих случаях это стоит обсудить.

Соответствие требованиям. Обычно в конце инспектирования смотрим на задачу или баг, для которого был предназначен тест. Если он не соответствует каким-то критериям, то лучше провести тестирование заново.

Стандарт оформления кода

Стандарт оформления кода (стандарт кодирования, стиль программирования) — набор правил и соглашений, используемых при написании исходного кода на некотором языке программирования. Наличие общего стиля программирования облегчает понимание и поддержание исходного кода, написанного более чем одним программистом, а также упрощает взаимодействие нескольких человек при разработке программного обеспечения.

Стандарт оформления кода обычно принимается и используется некоторой группой разработчиков программного обеспечения для единообразного оформления совместно используемого кода. Целью принятия и использования стандарта является упрощение восприятия программного кода человеком, минимизация нагрузки на память и зрение при чтении программы.

Образцом для стандарта кодирования может стать набор соглашений, принятых в какой-либо распространённой печатной работе по языку (например, стандарт кодирования на языке Си, получивший сокращённое наименование K&R, происходит из классического описания Си его авторами — Керниганом и Ритчи), широко применяемая библиотека или API (так, на распространение венгерской нотации явно повлияло её использование в MS-DOS и Windows API, а большинство стандартов кодирования для Delphi используют, в той или иной мере, манеру кодирования библиотеки VCL).

Реже разработчик языка выпускает подробные рекомендации по кодированию. Например, выпущены стандарты кодирования на C# от Microsoft[2] и на Java от Sun. Предложенная разработчиком или принятая в общеизвестных источниках манера кодирования в большей или меньшей степени дополняется и уточняется в корпоративных стандартах.

Стандарт сильно зависит от используемого языка программирования. Например, стандарт оформления кода для языка Си будет серьёзно отличаться от стандарта для языка BASIC. В целом, исходя из назначения стандарта, обычно он имеет целью добиться такого положения, когда программист достаточной квалификации мог бы дать заключение о функции, выполняемой конкретным участком кода, а в идеале — также определить его корректность, изучив только сам этот участок кода или, во всяком случае, минимально изучив другие части программы.

Иными словами, смысл кода должен быть виден из самого кода, без необходимости изучать контекст. Поэтому стандарт кодирования обычно строится так, чтобы за счёт определённого визуального оформления элементов программы повысить информативность кода для человека.

Обычно, стандарт оформления кода описывает:

- способы выбора названий и используемый регистр символов для имён переменных и других идентификаторов;
- запись типа переменной в её идентификаторе (венгерская нотация) и
- регистр символов (нижний, верхний, «верблюжий», «верблюжий» с малой буквы), использование знаков подчёркивания для разделения слов;
- стиль отступов при оформлении логических блоков — используются ли символы табуляции, ширина отступа;
- способ расстановки скобок, ограничивающих логические блоки;
- использование пробелов при оформлении логических и арифметических выражений;
- стиль комментариев и использование документирующих комментариев.

Основные принципы распространённых стандартов кодирования в последнее время оказывают влияние на синтаксис вновь создаваемых языков программирования. В некоторых из них соглашения, ранее применявшиеся только в стандартах кодирования, стали обязательными

элементами синтаксиса. Так, в некоторых современных языках (Python[3], Nemerle) отступы влияют на логику исполнения (то есть блоки кода выделяются не ключевыми словами, а размером отступов), в других (Ruby) — частью языка стали соглашения о регистре букв и префиксах для типов, констант, переменных и полей классов. В результате, если ранее недисциплинированный программист мог игнорировать стандарты кодирования из личных соображений, ради удобства или скорости написания кода, то теперь, при работе на новых языках, соблюдение стандартов в определённой мере контролируется транслятором.

Венгерская нотация

Своё название венгерская нотация получила благодаря программисту компании Microsoft венгерского происхождения Чарльзу Симони, предложившему её ещё во времена разработки первых версий MS-DOS. Эта система стала внутренним стандартом Майкрософт.

Суть венгерской нотации сводится к тому, что имена идентификаторов предваряются заранее оговорёнными префиксами, состоящими из одного или нескольких символов. При этом, как правило, ни само наличие префиксов, ни их написание не являются требованием языков программирования, и у каждого программиста (или коллектива программистов) они могут быть своими.

Применяемая система префиксов зависит от многих факторов:

- языка программирования (чем более «либеральный» синтаксис, тем больше контроля требуется со стороны программиста, а значит, тем более развита система префиксов. К тому же использование в каждом из языков программирования своей терминологии также вносит особенности в выбор префиксов);
- стиля программирования (объектно-ориентированный код может вообще не требовать префиксов, в то время как в «монолитном» для разборчивости они зачастую нужны);
- предметной области (например, префиксы могут применяться для записи единиц измерения);
- доступных средств автоматизации (генератор документации, навигация по коду, предиктивный ввод текста, автоматизированный рефакторинг и т. д.).

Таблица 1. Пример системы префиксов

Префикс	Объект
btn	Кнопка
chk	Флажок
cmd	Командная кнопка
frm	Форма
gbx	Рамка

lbl	Метка
opt	Переключатель
txt	Текстовое поле

Практическая работа по теме Инспекция кода

Цель работы: проведение инспекции программного кода на соответствие его стандартам кодирования.

Задачи работы:

- совершенствовать и развивать коммуникативно-познавательные умения, направленные на систематизацию и углубление знаний;
- закрепить навыки написания и чтения программного кода в среде VBA;
- приобрести навыки анализа программного кода;
- проанализировать существующий программный код на соответствие стандартам кодирования;
- выполнить инспекцию и ревьюирование программного кода.

Для работы над практической работой студент:

должен знать:

- понятие инспекции программного кода;
- подходы к проведению инспекции и ее виды;
- правила именования объектов кода согласно различных нотаций;
- способы редактирования изображений в графических редакторах.

Должен уметь:

- различать виды инспекции программного кода;
- определять способ проведения инспекции исходя из конкретной ситуации;
- различать нотации именования объектов кода;
- выполнять простейшие действия по обработке изображений в графических редакторах.

По итогам выполнения практической работы студент:

должен уметь:

- проводить анализ программного кода и выявлять его недостатки;
- выполнять переименование объектов кода с сохранением его работоспособности ;
- редактировать графические изображения, используемые при работе программы;

иметь практический опыт:

- проведения инспекции программного кода;
- анализа программного кода на соответствие стандартам кодирования;
- приведения программного кода в соответствие со стандартами кодирования.

Внимание: Некоторые решения при инспекции кода программы не являются оптимальными и выполнены исключительно в учебных целях с целью обеспечения наглядности, максимального соответствия учебному плану и демонстрации возможностей инспекции. Кроме того, работа выполняется с целью интеграции дисциплин *ОП.05 Основы алгоритмизации и программирования* и *МДК 01.01. Технология*

Специальность: 09.02.07 Информационные системы и программирование

Дисциплина: ОП.05 Основы алгоритмизации и программирования

ОП.08 Основы проектирования баз данных

МДК 01.01 Технология разработки программного обеспечения

Преподаватель: Кулик Е.В., Михеева Т.А.

разработки программного обеспечения специальности 09.02.07 Информационные системы и программирование. Инспекция проводится на примере программы вычисления площади фигур.

Этапы работы:

1. Анализ исходного программного кода и выявление его недостатков.
2. Определение путей устранения недостатков.
3. Устранение выявленных недостатков.
4. Проведение повторного анализа программного кода.
5. В случае выявления недостатков выполнить п.п.2-4.

Ход работы:

1. Анализ исходного программного кода выявил следующие недостатки:

- Объекты интерфейса формы именованы без учета требований Венгерской нотации.
- Программный код не содержит комментарии и требуемые отступы.
- Рисунки изображения фигуры и соответствующей ей формулы размещены в разных графических файлах, оптимальнее совместить изображения в один файл.
- Математическая константа числа π вводится в текстовое поле.

Инспекция кода, устранение выявленных недостатков:

- Переименовать объекты интерфейса согласно требований Венгерской нотации, соответственно внести изменения в программный код модуля формы (рис.1).
- Написать комментарии к коду. Расставить отступы (рис.2).
- Удалить текстовое поле для ввода числа π . Создать константу (переменную) числа π в коде программы.
- Объединить в один графический файл изображения фигуры и соответствующей ей формулы (рис.3-4).

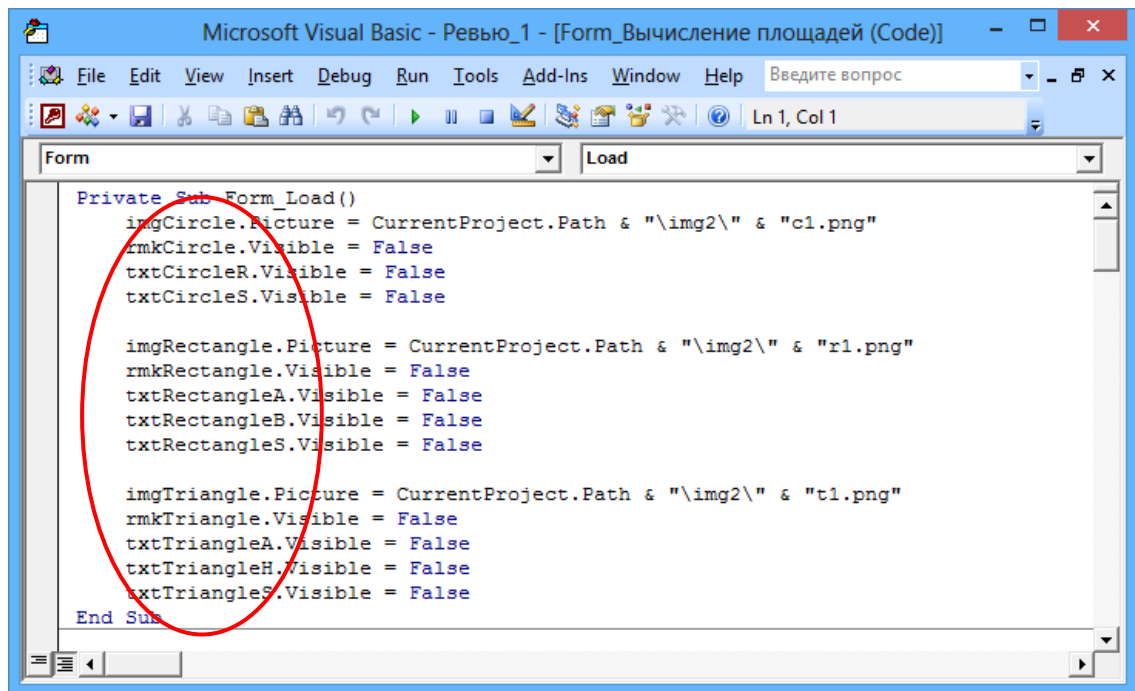


Рисунок 1. Правильное именование объектов интерфейса

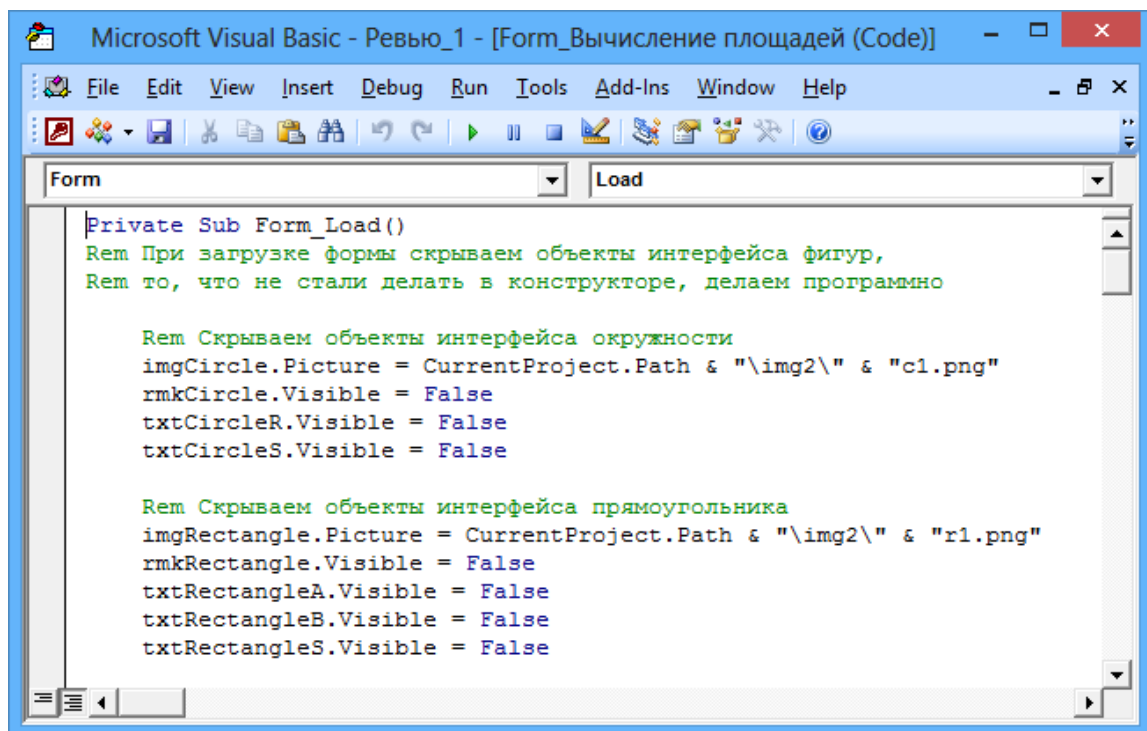


Рисунок 2. Фрагмент кода с комментариями и отступами

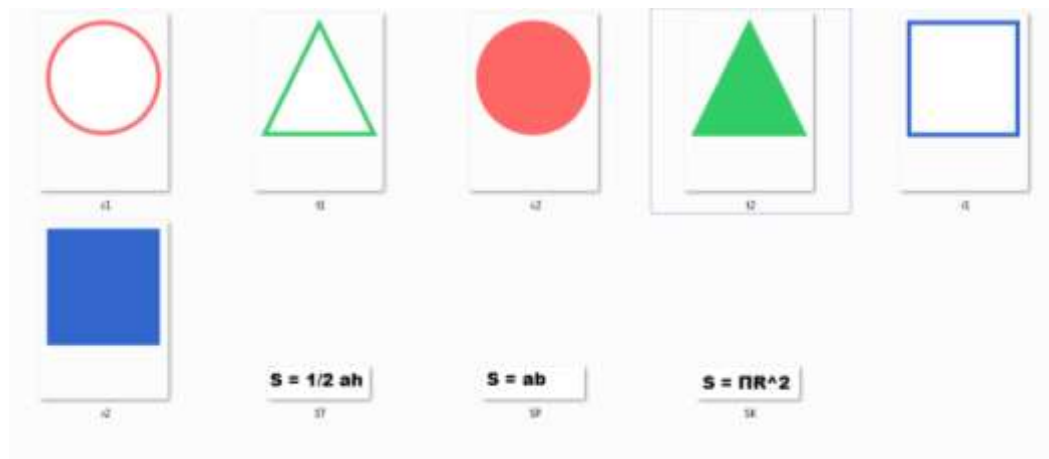


Рисунок 3. Набор изображений до инспекции

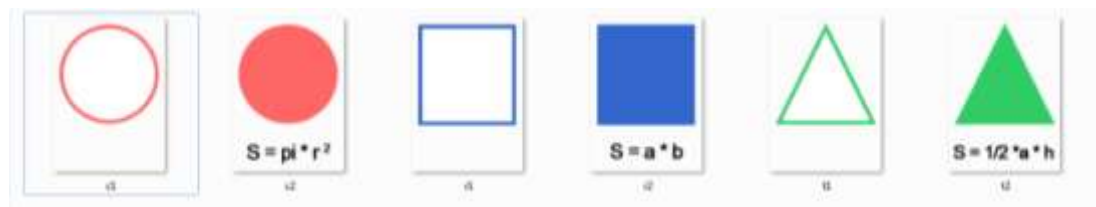


Рисунок 4. Набор изображений после инспекции

2. Повторный анализ программного кода выявил следующие недостатки:

- Программный код расчета площади фигуры повторяется во всех обработчиках событий текстовых полей, содержащих геометрические размеры фигуры.
- Программный код скрытия объектов интерфейса повторяется в нескольких обработчиках событий.

Инспекция кода, устранение выявленных недостатков:

- Написать *пользовательскую функцию* расчета площади геометрической фигуры (лист.1) (рис.5)
- Написать *пользовательские процедуры* скрытия объектов интерфейса фигуры (лист.2) (рис.6).

Листинг 1. Функция расчета площади геометрической фигуры

```
'arg1 - аргумент 1, определяем вид фигуры. Возможные значения аргумента
'   "с" - окружность
'   "r" - прямоугольник
'   "t" - треугольник
'arg2 - аргумент 2, первый числовой параметр фигуры
'arg3 - аргумент 3, второй числовой параметр фигуры
Function area_calculation(arg1 As String, arg2 As Byte, arg3 As Byte) As
```

```

Integer
'объявление переменной
Dim bytS As Integer
'объявление константы числа pi
Const PI = 3.14
If (arg1 = "c") Then
    bytS = PI * (arg2 * arg2) 'площадь окружности
ElseIf (arg1 = "r") Then
    bytS = arg2 * arg3 'площадь прямоугольника
ElseIf (arg1 = "t") Then
    bytS = (arg2 * arg3) / 2 'площадь треугольника
End If
area_calculation = bytS
End Function

```

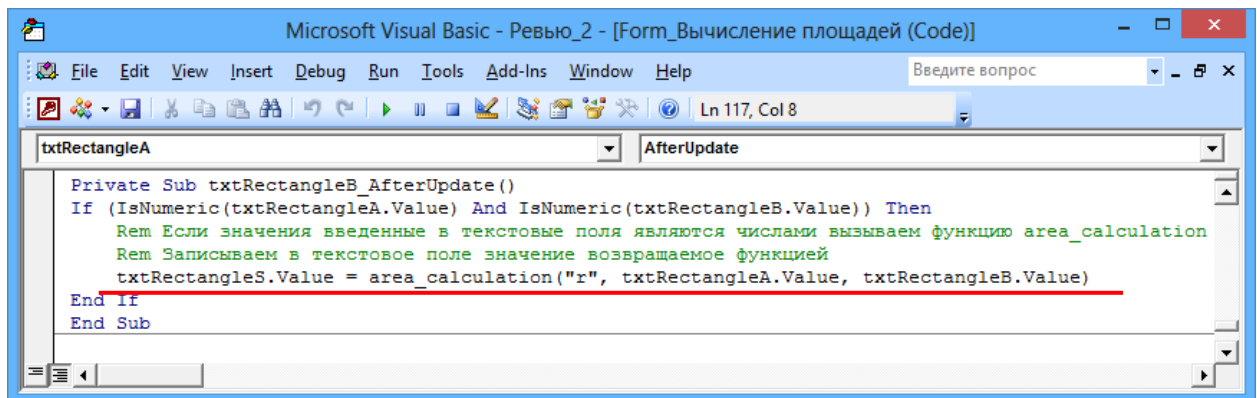


Рисунок 5. Вызов функции для расчета площади прямоугольника

Листинг 2. Процедура скрытия объектов интерфейса треугольник

```

Rem Пользовательская процедура скрытия объектов интерфейса треугольник
Public Sub Triangle_Hide()
    imgTriangle.Picture = CurrentProject.Path & "\img2\" & "t1.png"
    rmkTriangle.Visible = False
    txtTriangleA.Visible = False
    txtTriangleH.Visible = False
    txtTriangleS.Visible = False
End Sub

```

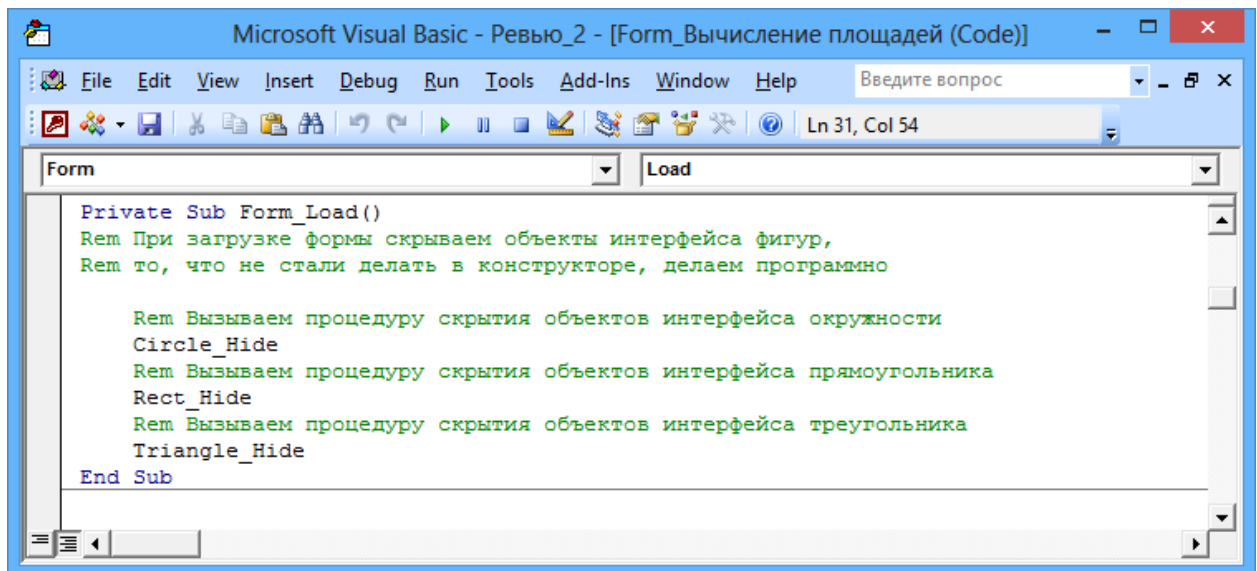



Рисунок 6. Вызов пользовательских процедур скртия объектов

3. Анализ программного кода выявил новые недостатки:

Обработчики событий *После обновления* для текстовых полей:

- txtTriangleA_AfterUpdate() и txtTriangleH_AfterUpdate().
- txtRectangleB_AfterUpdate() и txtRectangleA_AfterUpdate().

содержат одинаковый код. Следовательно необходимо оптимизировать программный код данных обработчиков.

Инспекция кода, устранение выявленных недостатков:

- Универсализировать функцию *area_calculation()*. Проверку на соответствие значений текстовых полей числу вынести из обработчика события в тело функции (лист.3) (рис.7).
- В обработчики событий добавить инструкцию подавления ошибки *On Error Resume Next*.

Листинг 3. Функция с проверкой значений текстовых полей

```

Function area_calculation(arg1 As String, arg2 As Byte, arg3 As Byte) As Integer
    'объявление переменной
    Dim bytS As Integer
    'объявление константы числа pi
    Const PI = 3.14

    'если значения введенные в текстовые поля являются числами
    If (IsNumeric(arg2) And IsNumeric(arg3)) Then
        If (arg1 = "c") Then
            bytS = PI * (arg2 * arg2)
        End If
    End If
End Function
  
```

Специальность: 09.02.07 Информационные системы и программирование

Дисциплина: ОП.05 Основы алгоритмизации и программирования

ОП.08 Основы проектирования баз данных

МДК 01.01 Технология разработки программного обеспечения

Преподаватель: Кулик Е.В., Михеева Т.А.

```

ElseIf (arg1 = "r") Then
    bytS = arg2 * arg3
ElseIf (arg1 = "t") Then
    bytS = (arg2 * arg3) / 2
End If
End If
area_calculation = bytS
End Function

```

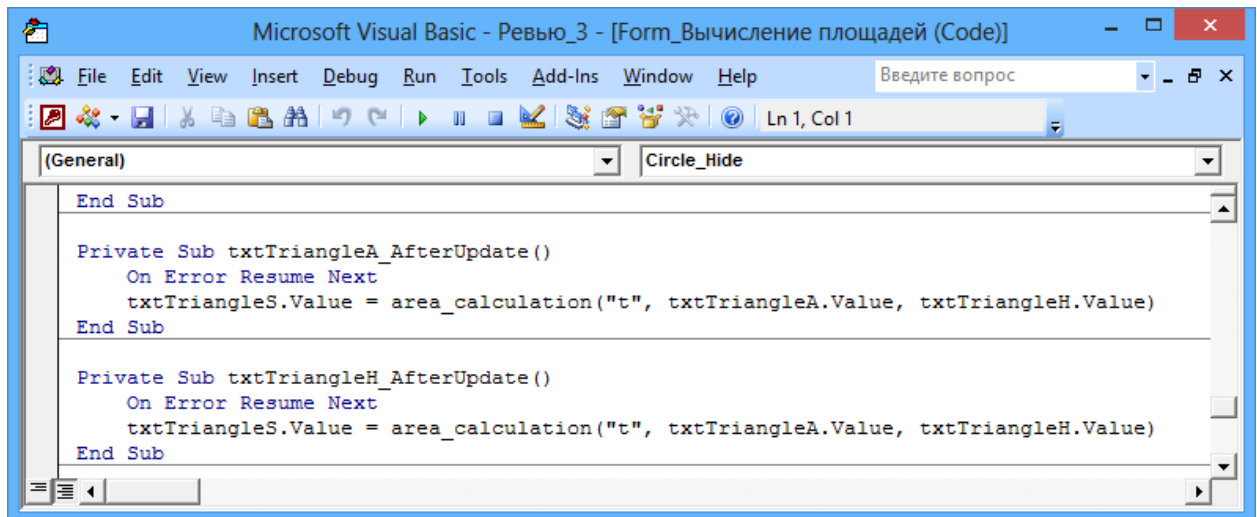


Рисунок 7. Вызов пользовательской функции

Заключение

В методических рекомендациях по выполнению практической работы поставлены и решены следующие задачи:

- совершенствованы умения, систематизированы и углублены знания проектирования приложений;
- закреплены навыки написания и чтения программного кода в среде VBA;
- приобретены навыки анализа программного кода;
- проанализирован существующий программный код на соответствие стандартам кодирования;
- выполнена инспекция и ревьюирование программного кода программы *Вычисление площади фигур*.

Анализ программного кода направленный на поиск необнаруженных на ранних стадиях разработки программного продукта ошибок, привел к возможности выявления некачественных решений и критических мест в программе.

Результатом выполнения практической работы стало ревьюирование программы, позволяющей рассчитать площадь геометрической фигуры.

Созданное приложение как в области логики работы, так и в области программного исполнения может быть модернизировано согласно дальнейших требований разработчика.

В качестве дальнейшего ревьюирования приложения можно предложить:

1. процесс вычислений площади "привязать" к событию *LostFocus*;
2. создание общей процедуры в обработчиках событий текстовых полей фигур;
3. написание единой функции скрытия пользовательских объектов интерфейса геометрических фигур;
4. инструкцию подавления ошибок *On Error Resume Next* заменить инструкцией обработки ошибок *On Error Go To*;
5. с помощью командной кнопки *Очистить* не просто чистить текстовые поля, но возвращать программу в исходное состояние (на момент загрузки пользовательской формы);
6. расширить список геометрических фигур. Названия фигур можно выводить в элементе управления *Поле со списком*. При выборе элемента из списка загружать графический файл соответствующей фигуры. Если расчет площади фигуры требует более двух параметров – делать видимыми предварительно созданные текстовые поля. Менять значения надписей к текстовым полям в соответствии выбранной фигуры.

Список использованных источников

1. Статья. Сравнение методик обзора кода. [Режим доступа <https://habr.com/ru/post/151450/>] Дата обращения: 16.07.2019 года
2. Статья. Стандарт оформления кода [Режим доступа https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82_%D0%BE%D1%84%D0%BE%D1%80%D0%BC%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%BA%D0%BE%D0%B4%D0%B0] Дата обращения: 16.07.2019 года.