МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ
И ПРОГРАММНОЙ ИНЖЕНЕРИИ

МЕТРОЛОГИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Методические указания к выполнению лабораторных работ (draft)

Санкт-Петербург 2018

Составители: Павлов Е.В., Копкин Е.В.

Рецензент:

Метрология программного обеспечения. Методические указания к выполнению лабораторных работ – СПб ГУАП, 2018. – 42 с.

В методические указания включены краткие теоретические сведения, необходимые для выполнения лабораторных работ, требования к содержанию отчетов и порядку выполнения работ, а также приведены примеры выполнения.

Методические указания предназначены для выполнения лабораторных работ по дисциплине «Метрология программного обеспечения» студентами различных форм обучения по направлениям 09.03.04 «Программная инженерия» и 01.03.02 «Прикладная математика и информатика».

Подготовлены кафедрой компьютерных технологий и программной инженерии.

СОДЕРЖАНИЕ

1. ЛАБОРАТОРНАЯ РАБОТА
«ОЦЕНКА КАЧЕСТВА ПРОГРАММНЫХ СРЕДСТВ»

1.1 Цель работы

Целью данной работы является изучение оценочных показателей качества и получение навыков описания модели качества программного обеспечения.

1.2 Задание на лабораторную работу

Описать модель качества программного обеспечения при помощи оценочных показателей качества в соответствии со стандартом ГОСТ 28195-89.

Выбрать исходный текст программы в соответствии с заданием объемом: количество страниц А4 шрифт Cambria 12 кегль (без учета комментариев). Программа должна быть нелинейной (обязательное наличие операторов ветвления и циклов).

- для оценки «хорошо» объем исходного листинга программы: 1-2 стр;
- для оценки «отлично» минимальный объем исходного листинга программы: 3 стр.

1.3 Порядок выполнения работы

1. Выбрать исходный текст программы с любого Open Source источника. Языки реализации: MATLAB, C, C++, Java, C#. Настоятельно не рекомендуется брать мобильные или веб-приложения в виду специфики ЛР. Выбранный текст программы, закрепляется за студентом как индивидуальный вариант задания для данной и последующих ЛР.
2. Описать программу по показателям качества, изложенным в подразделе 1.4;
3. Ознакомиться с требованиями по содержанию отчета в подразделе 1.5;
4. Написать отчет о работе.

1.4 Теоретический материал

ГОСТ 28195-89 устанавливает общие положения по оценке качества программных средств [1]. Ниже представлены показатели качества по данному стандарту:

1. Надежность
- Возможность обработки ошибочных ситуаций;
- Полнота обработки ошибочных ситуаций;
- Наличие проверки допустимых значений входных данных;
- Наличие системы контроля полноты входных данных;
- Наличие средств контроля корректности входных данных;
- Наличие средств восстановления процесса в случае сбоев оборудования;
- Наличие возможности разделения по времени выполнения отдельных функций программ;
- Наличие возможности повторного старта с точки останова;
- Наличие обработки неопределенностей;
- Наличие централизованного управления процессами, конкурирующими из-за ресурсов;
- Наличие возможности автоматически обходить ошибочные ситуации в процессе вычисления;
- Наличие средств, обеспечивающих завершение процесса в случае ошибок;
- Наличие средств, обеспечивающих выполнение программы в сокращенном объеме в случае ошибок.

2. Сопровождаемость

- Наличие комментариев в точках входа и выхода программы;
- Оценка простоты программы по числу точек входа и выхода:

$$W = 1 / ((D +1) \times (F +1)),$$

где D – общее число точек входа в программу, F – общее число точек выхода из программы;

- Оценка простоты программы по числу переходов по условию:

$$U = (1 – A / B),$$

где A – общее число переходов по условию, B – общее число исполняемых операторов.

- Оценка программы по числу циклов;
- Используется ли язык высокого уровня;
- Наличие заголовочных комментариев программы с указанием ее структурных и функциональных характеристик;
- Использование при построении программы метода структурного программирования;
- Использование при построении программы метода объектно-ориентированного программирования;
- Наличие ограничений на размеры модуля;
- Наличие модульной схемы программы и поддержка оверлейной структуры;
- Оценка программы по числу уникальных модулей;
- Оценка программы по числу циклов с одним входом и одним выходом.

3. Удобство применения

- Возможность освоения программных средств по документации;
- Возможность освоения программы в обучающем режиме;
- Полнота и понятность документации для освоения;
- Легкость и быстрота загрузки и запуска программы;
- Легкость и быстрота завершения работы программы;
- Возможность распечатки содержимого программы;
- Возможность приостанова и повторного запуска работы без потерь информации;
- Соответствие меню требованиям пользователя;
- Возможность прямого перехода вверх и вниз по многоуровневому меню (пропуск уровней);
- Возможность управления подробностью получаемых выходных данных;
- Обеспечение удобства ввода данных;
- Интуитивно понятный интерфейс;
- Легкость восприятия оперируемой информацией и данными.

4. Эффективность

- Функции ввода/вывода;
- Функции защиты и проверки данных;
- Функции защиты от несанкционированного доступа;
- Функции контроля доступа;
- Число знаков после запятой в результатах вычислений;
- Требуемый объем внутренней памяти (оперативная память);
- Требуемый объем внешней памяти (дисковое пространство);
- Оценка числа потенциальных пользователей;
- Оценка числа функций программного обеспечения;

- Насколько набор функций удовлетворяет требованиям пользователя;
- Насколько возможности программ охватывают область решаемых пользователем задач;
- Возможность настройки формата выходных данных для конкретных пользователей;
- Оценка независимости модулей;
- Оценка программ по числу переходов и точек ветвления;
- Оценка зависимости программы от программ операционной системы;
- Зависимость от других программных средств;
- Оценка программы по использованию условных переходов;
- Оценка программы по использованию безусловных переходов;
- Оценка программы по использованию локальных переменных;
- Оценка программы по числу комментариев;
- Комментарии к точкам ветвлений;
- Комментарии к операторам объявления переменных;
- Оценка семантики операторов;
- Семантика имен используемых переменных;
- Использование отступов, сдвигов и пропусков при формировании текста.

5. Корректность
- Наличие интерфейса с пользователем;
- Отсутствие противоречий в настройке системы;
- Единообразие организации списков передаваемых параметров;
- Единообразие наименования каждой переменной и константы;
- Используются ли разные идентификаторы для разных по смыслу переменных;
- Все ли общие переменные объявлены как глобальные переменные.

1.5 Содержание отчета

Отчет о работе должен содержать:
1. Титульный лист
2. Цель работы
3. Задание на лабораторную работу и вариант
4. Оценка показателей качества программного обеспечения
5. Выводы по работе
6. Использованные источники (2-4 источника)
7. Приложение (листинг программы)

1.6 Пример выполнения работы

В качестве варианта задания выбрана программа[1], написанная на Node.JS (серверный JavaScript), реализующая «общение» с LongPoll-сервером ВКонтакте.

Таблица 4.1 — Оценка показателей качества ПО по ГОСТ 28195-89

	Наименование показателя качества	Оценка, пояснение
1.	Надежность	
1.1	Возможность обработки ошибочных ситуаций	присутствует (обработка исключений)
1.2	Полнота обработки ошибочных ситуаций	отсутствует
1.3	Наличие проверки допустимых значений входных данных	отсутствует

1 Листинг данной программы приведен в ПРИЛОЖЕНИИ.

1.4	Наличие системы контроля полноты входных данных	присутствует
1.5	Наличие средств контроля корректности входных данных	отсутствует
1.6	Наличие средств восстановления процесса в случае сбоев оборудования	отсутствует
1.7	Наличие возможности разделения по времени выполнения отдельных функций программ	присутствует
1.8	Наличие возможности повторного старта с точки останова	отсутствует
1.9	Наличие обработки неопределенностей (деление на 0, квадратный корень из отрицательного числа etc.)	отсутствует (не встречается)
1.10	Наличие централизованного управления процессами, конкурирующими из-за ресурсов	отсутствует (не встречается)
1.11	Наличие возможности автоматически обходить ошибочные ситуации в процессе вычисления	присутствует (обработка исключений)
1.12	Наличие средств, обеспечивающих завершение процесса в случае ошибок	присутствует
1.13	Наличие средств, обеспечивающих выполнение программы в сокращенном объеме в случае ошибок	присутствует
2.	Сопровождаемость	
2.1	Наличие комментариев в точках входа и выхода программы	присутствует
2.2	Оценка простоты программы по числу точек входа и выхода: $W = 1 / ((D +1) \times (F + 1))$, где D – общее число точек входа в программу, F – общее число точек выхода из программы	$1 / ((1 + 1) \times (12 + 1)) = 0.04$
2.3	Оценка простоты программы по числу переходов по условию: $U = (1 – A / B)$, где A – общее число переходов по условию, B – общее число исполняемых операторов в программе	$1 – (11 / 52) = 0.79$
2.4	Оценка программы по числу циклов (количество циклов в программе)	в программе отсутствуют циклы, но есть 2 асинхронные конструкции
2.5	Используется ли язык высокого уровня	да (JavaScript)
2.6	Наличие заголовочных комментариев программы с указанием ее структурных и функциональных характеристик	отсутствуют
2.7	Использование при построении программы метода структурного программирования	да
2.8	Использование при построении программы метода объектно-ориентированного программирования	нет
2.9	Наличие ограничений на размеры модуля	отсутствует (ограничений нет)
2.10	Наличие модульной схемы программы и поддержка оверлейной структуры	отсутствует (оверлейная структура не поддерживается)
2.11	Оценка программы по числу уникальных модулей	программа представляет собой 1 модуль
2.12	Оценка программы по числу циклов с одним входом и одним выходом	циклы отсутствуют
3.	Удобство применения	
3.1	Возможность освоения программных средств по документации	отсутствует
3.2	Возможность освоения программы в обучающем режиме	отсутствует

3.3	Полнота и понятность документации для освоения	отсутствует
3.4	Легкость и быстрота загрузки и запуска программы	присутствует
3.5	Легкость и быстрота завершения работы программы	отсутствует (прекращение работы программы возможно только при ручном завершении процесса)
3.6	Возможность распечатки содержимого программы	отсутствует
3.7	Возможность приостанова и повторного запуска работы без потерь информации	отсутствует
3.8	Соответствие меню требованиям пользователя	меню отсутствует
3.9	Возможность прямого перехода вверх и вниз по многоуровневому меню (пропуск уровней)	меню отсутствует
3.10	Возможность управления подробностью получаемых выходных данных	отсутствует
3.11	Обеспечение удобства ввода данных	отсутствует
3.12	Интуитивно понятный интерфейс	отсутствует
3.13	Легкость восприятия оперируемой информацией и данными	отсутствует
4.	Эффективность	
4.1	Функции ввода/вывода	реализованы
4.2	Функции защиты и проверки данных	реализованы частично, присутствует только проверка от пустых полей
4.3	Функции защиты от несанкционированного доступа	отсутствуют
4.4	Функции контроля доступа	отсутствуют
4.5	Число знаков после запятой в результатах вычислений	отсутствуют (не требуется)
4.6	Требуемый объем внутренней памяти (оперативная память)	32 MB
4.7	Требуемый объем внешней памяти (дисковое пространство)	не требуется
4.8	Оценка числа потенциальных пользователей	до 300 одновременно
4.9	Оценка числа функций программного обеспечения	одна
4.10	Насколько набор функций удовлетворяет требованиям пользователя	полностью
4.11	Насколько возможности программ охватывают область решаемых пользователем задач	полностью
4.12	Возможность настройки формата выходных данных для конкретных пользователей	отсутствует
4.13	Оценка независимости модулей	подключаемые модули независимы, ПО не может использоваться как библиотека или модуль для другого проекта
4.14	Оценка программ по числу переходов и точек ветвления	в программе 8 точек ветвления
4.15	Оценка зависимости программы от программ операционной системы	не зависит
4.16	Зависимость от других программных средств	зависит от наличия программ nginx/apache, node и npm, а также npm-пакетов http, https, url, querystring
4.17	Оценка программы по использованию условных переходов	8 условных переходов

4.18	Оценка программы по использованию безусловных переходов	отсутствуют
4.19	Оценка программы по использованию локальных переменных	12 локальных переменных
4.20	Оценка программы по числу комментариев	4 комментария к каждой функции
4.21	Комментарии к точкам ветвлений	отсутствуют
4.22	Комментарии к операторам объявления переменных	отсутствуют;
4.23	Оценка семантики операторов	написаны в едином стиле
4.24	Семантика имен используемых переменных	написаны в едином стиле
4.25	Использование отступов, сдвигов и пропусков при формировании текста	присутствует
5.	Корректность	
5.1	Наличие интерфейса с пользователем	отсутствует
5.2	Отсутствие противоречий в настройке системы	противоречия отсутствуют
5.3	Единообразие организации списков передаваемых параметров	все параметры передаются единообразно
5.4	Единообразие наименования каждой переменной и константы	все переменные и константы названы единообразно; наименования некоторых переменных повторяются в разных блоках кода (в разных функциях)
5.5	Используются ли разные идентификаторы для разных по смыслу переменных	используются
5.6	Все ли общие переменные объявлены как глобальные переменные	не все, переменная response (http.ClientRequest) передается в качестве аргумента и не может использоваться как глобальная в связи с техническими особенностями системы

К сожалению, показатели по данному ГОСТу закончились, но не расстраивайтесь, в следующей ЛР вас ждет очередная порция увлекательных показателей и характеристик, я бы даже сказал еще более увлекательных.

Пример вывода:

В результате выполнения данной лабораторной работы были изучены оценочные показатели качества программного обеспечения в соответствии с ГОСТ 28195-89. Составлена модель качества на основе таких показателей как надежность, сопровождаемость, удобство применения, эффективность и корректность.

Некоторые из показателей качества не были учтены должным образом или их оценка была произведена некорректно ввиду специфики оцениваемого программного обеспечения. Настоящий стандарт не включает в себя характеристики для полноценной оценки мобильных и web-приложений. В частности, возникли трудности при оценки таких параметров как:

- *перечислите параметры, которые вызвали затруднения.*

1.7 Контрольные вопросы

1) Назначение ГОСТ 28195-89? На секунду представил, как я буду это спрашивать с серьезным лицом. Пожалуй, не стоит так себя смешить, это опасно для жизни.

2) Что представляет собой надежность?

3) Что представляет собой сопровождаемость?

4) Что представляет собой удобство применения?

5) Что представляет собой эффективность?

6) Что представляет собой корректность?

7) В чем заключаются методы структурного программирования?

8) В чем заключаются методы объектно-ориентированного программирования?

9) Приведите примеры операторов условного и безусловного перехода.

10) Чем характеризуются локальные и глобальные переменные?

11) Что подразумевается под семантикой операторов и имен переменных?

12) Поясните: единообразие организации списков передаваемых параметров.

2. ЛАБОРАТОРНАЯ РАБОТА
«ПОСТРОЕНИЕ МОДЕЛИ КАЧЕСТВА ПРИ ИСПОЛЬЗОВАНИИ»

2.1 Цель работы

Целью данной работы является изучение способов описания модели качества программных продуктов при использовании.

2.2 Задание на лабораторную работу

Описать модель качества программного продукта при использовании для программы из ЛР №1 в соответствии со стандартом ГОСТ Р ИСО/МЭК 25010-2015.

- для оценки «хорошо» объем исходного листинга программы: 1-2 стр;
- для оценки «отлично» минимальный объем исходного листинга программы: 3 стр.

2.3 Порядок выполнения работы

1. Изучить теоретический материал в подразделе 2.4 (удачи);
2. Построить модель качества при использовании для программы из ЛР №1;
3. Ознакомиться с требованиями по содержанию отчета в подразделе 2.5;
4. Написать отчет о работе.

2.4 Теоретический материал

Качество при использовании — это степень, в которой продукт или система могут использоваться конкретными пользователями для достижения определенных целей с эффективностью, производительностью, свободой от риска и удовлетворенностью в конкретных условиях использования для удовлетворения их потребностей [2].

Свойства качества при использовании представляют собой пять характеристик, которыми являются: эффективность, производительность, удовлетворенность, свобода от риска и покрытие контекста (табл. 1).

Таблица 2.4.1 — Характеристики и подхарактеристики качества при использовании

1. Эффективность
2. Производительность
3. Удовлетворенность
3.1 Полноценность
3.2 Доверие
3.3 Удовольствие
3.4 Комфорт
4. Свобода от риска
4.1 Смягчение отрицательных последствий экономического риска
4.2 Смягчение отрицательных последствий риска здоровья и безопасности
4.3 Смягчение отрицательных последствий экологического риска

5. Покрытие контекста
5.1 Полнота контекста
5.2 Гибкость

1. Эффективность – точность и полнота, с которой пользователи достигают определенных целей.

2. Производительность – связь точности и полноты достижения пользователями целей с израсходованными ресурсами.

3. Удовлетворенность – способность продукта или системы удовлетворить требованиям пользователя в заданном контексте использования.

3.1 Полноценность – степень удовлетворенности пользователя достижением прагматических целей, включая результаты использования и последствия использования.

3.2 Доверие – степень уверенности пользователя или другого заинтересованного лица в том, что продукт или система будут выполнять свои функции так, как это предполагалось.

3.3 Удовольствие – степень удовольствия пользователя от удовлетворения персональных требований. В число персональных требований могут входить потребности получения новых знаний и навыков, личное общение и ассоциации с приятными воспоминаниями.

3.4 Комфорт – степень удовлетворенности пользователя физическим комфортом.

4. Свобода от риска – способность продукта или системы смягчать потенциальный риск для экономического положения, жизни, здоровья или окружающей среды. Риск является функцией вероятности возникновения такой угрозы и потенциальных неблагоприятных последствий этой угрозы.

4.1 Смягчение отрицательных последствий экономического риска – способность продукта или системы смягчать потенциальный риск для финансового положения и эффективной работы, коммерческой недвижимости, репутации или других ресурсов в предполагаемых условиях использования.

4.2 Смягчение отрицательных последствий риска для здоровья и безопасности – способность продукта или системы смягчать потенциальный риск для людей в предполагаемых условиях использования.

4.3 Смягчение отрицательных последствий экологического риска – способность продукта или системы смягчать потенциальный риск для имущества или окружающей среды в предполагаемых условиях использования.

5. Покрытие контекста – степень, в которой продукт или система могут быть использованы с эффективностью, результативностью, свободой от риска и в соответствии с требованиями как в первоначально определенных условиях использования, так и в условиях, выходящих за спецификации.

5.1 Полнота контекста – степень, в которой продукт или система могут быть использованы с эффективностью, результативностью, свободой от риска и в соответствии с требованиями при всех указанных условиях использования. Например, степень, в которой программное обеспечение применимо при использовании маленького экрана, с низкой сетевой пропускной способностью, неквалифицированными пользователями и в отказоустойчивом режиме (например, при отсутствии сети).

5.2 Гибкость – степень, в которой продукт или система могут быть использованы с эффективностью, результативностью, свободой от риска и в соответствии с требованиями в условиях, выходящих за рамки первоначально определенных в требованиях. Гибкость может быть достигнута путем адаптации продукта для дополнительных групп пользователей, задач и культур. Гибкость позволяет использовать продукт в условиях обстоятельств, возможностей и индивидуальных настроек, которые не были предусмотрены заранее.

Если продукт не обладает гибкостью, то он не может быть безопасно использован в непредусмотренных условиях.

Гибкость может быть определена либо как степень, до которой продукт может быть использован пользователями непредусмотренного типа для достижения дополнительных целей с эффективностью, результативностью, свободой от риска и в соответствии с требованиями при дополнительных условиях использования, либо как возможность изменения для поддержки адаптации к новым типам пользователей, задач и сред, а также пригодности для индивидуализации. Пресвятая Дева Мария, что я только что прочитал...

Модель качества продукта.

Модель качества продукта разделяет свойства качества продукта на восемь характеристик, которыми являются: функциональная пригодность, надежность, уровень производительности, удобство использования, защищенность, совместимость, сопровождаемость и переносимость. Каждая характеристика состоит из нескольких связанных подхарактеристик (табл. 2).

Таблица 2.4.2 — Модель качества продукта

1. Функциональная пригодность
1.1 Функциональная полнота
1.2 Функциональная корректность
1.3 Функциональная целесообразность
2. Уровень производительности
2.1 Временные характеристики
2.2 Использование ресурсов
2.3 Потенциальные возможности
3. Совместимость
3.1 Сосуществование
3.2 Интероперабельность
4. Удобство использования
4.1 Определимость пригодности
4.2 Изучаемость
4.3 Управляемость
4.4 Защищенность от ошибки пользователя
4.5 Эстетика пользовательского интерфейса
4.6 Доступность
5. Надежность
5.1 Завершенность
5.2 Готовность
5.2 Отказоустойчивость
5.2 Восстанавливаемость

6. Защищенность
6.1 Конфиденциальность
6.2 Целостность
7. Сопровождаемость
7.1 Модульность
7.2 Возможность многократного использования
7.3 Анализируемость
7.4 Модифицируемость
7.5 Тестируемость
8. Переносимость
8.1 Адаптируемость
8.2 Устанавливаемость
8.3 Взаимозаменяемость

1. Функциональная пригодность – степень, в которой продукт или система обеспечивают выполнение функции в соответствии с заявленными и подразумеваемыми потребностями при использовании в указанных условиях.

1.1 Функциональная полнота – степень покрытия совокупностью функций всех определенных задач и целей пользователя.

1.2 Функциональная корректность – степень обеспечения продуктом или системой необходимой степени точности корректных результатов.

1.3 Функциональная целесообразность – степень функционального упрощения выполнения определенных задач и достижения целей. Например, для решения задачи пользователю предоставляется возможность выполнять только необходимые шаги, исключая любые ненужные.

2. Уровень производительности – производительность относительно суммы использованных при определенных условиях ресурсов. Ресурсы могут включать в себя другие программные продукты, конфигурацию программного и аппаратного обеспечения системы и материалы (например, бумагу для печати, носители).

2.1 Временные характеристики – степень соответствия требованиям по времени отклика, времени обработки и показателей пропускной способности продукта или системы.

2.2 Использование ресурсов – степень удовлетворения требований по потреблению объемов и видов ресурсов продуктом или системой при выполнении их функций.

2.3 Потенциальные возможности – степень соответствия требованиям предельных значений параметров продукта или системы. В качестве параметров могут быть: возможное количество сохраняемых элементов, количество параллельно работающих пользователей, емкость канала, пропускная способность по транзакциям и размер базы данных.

3. Совместимость – способность продукта, системы или компонента обмениваться информацией с другими продуктами, системами или компонентами, и/или выполнять требуемые функции при совместном использовании одних и тех же аппаратных средств или программной среды.

3.1 Сосуществование – способность продукта совместно функционировать с другими независимыми продуктами в общей среде с разделением общих ресурсов и без отрицательного влияния на любой другой продукт.

3.2 Интероперабельность – способность двух или более систем, продуктов или компонент обмениваться информацией и использовать такую информацию.

4. Удобство использования – степень, в которой продукт или система могут быть использованы определенными пользователями для достижения конкретных целей с эффективностью, результативностью и удовлетворенностью в заданном контексте использования. Удобство использования может быть либо задано или измерено как характеристика качества продукта в терминах ее подхарактеристик, либо задано или измерено непосредственно показателями, которые составляют подмножество качества при использовании.

4.1 Определимость пригодности – возможность пользователей понять, подходит ли продукт или система для их потребностей, сравним ли с функциональной целесообразностью. Определимость пригодности зависит от возможности распознать уместность продукта или функций системы от первоначальных впечатлений о продукте или системе и/или от какой-либо связанной с ними документации. Информация, предоставляющая продукт или систему, может включать в себя демонстрации, обучающие программы, документацию, а для веб-сайта – информацию на домашней странице.

4.2 Изучаемость – возможность использования продукта или системы определенными пользователями для достижения конкретных целей обучения для эксплуатации продукта или системы с эффективностью, результативностью, свободой от риска и в соответствии с требованиями в указанном контексте использования. Изучаемость может быть задана или измерена либо как степень возможности использования продукта или системы определенными пользователями для достижения конкретных целей обучения, для эксплуатации продукта или системы с эффективностью, результативностью, свободой от риска и в соответствии с требованиями в указанном контексте использования, либо как свойство продукта, соответствующего пригодности для обучения.

4.3 Управляемость – наличие в продукте или системе атрибутов, обеспечивающих простое управление и контроль. Управляемость соответствует управляемости, устойчивости к ошибкам (оператора) и согласованности с ожиданиями пользователей.

4.4 Защищенность от ошибки пользователя – уровень системной защиты пользователей от ошибок.

4.5 Эстетика пользовательского интерфейса – степень «приятности» и «удовлетворенности» пользователя интерфейсом взаимодействия с пользователем. Это свойство относится к тем свойствам продукта или системы, которые повышают привлекательность интерфейса для пользователя, таким как использование цвета и естественного графического дизайна.

4.6 Доступность – возможность использования продукта или системы для достижения определенной цели в указанном контексте использования широким кругом людей с самыми разными возможностями. В диапазон возможностей входят ограничения возможностей, связанные с возрастом. Доступность для людей с ограниченными возможностями может быть задана или измерена либо как степень, в которой продукт или система могут быть применены пользователями с указанными ограниченными возможностями для достижения определенных целей с эффективностью, результативностью, свободой от риска и в соответствии с требованиями в указанном контексте использования, либо как наличие свойств продукта для поддержки доступности.

5. Надежность – степень выполнения системой, продуктом или компонентом определенных функций при указанных условиях в течение установленного периода времени. В программном обеспечении износа не происходит. Проблемы с надежностью возникают из-за недостатков в требованиях, при разработке и реализации или из-за изменений условий использования. Характеристики функциональной надежности программного обеспечения включают в себя готовность и либо присущие ей, либо внешние влияющие факторы, такие

как надежность и доступность (включая отказоустойчивость и восстанавливаемость), безопасность (включая обеспечение конфиденциальности и целостность), пригодность для обслуживания, долговечность и техническую поддержку.

5.1 Завершенность – степень соответствия системы, продукта или компонента при нормальной работе требованиям надежности.

5.2 Готовность – степень работоспособности и доступности системы, продукта или компонента. В общем, готовность можно оценить как долю общего времени, в течение которого система, продукт или компонент находятся в работающем состоянии. Готовность, таким образом, определяется сочетанием завершенности, которая определяет частоту отказов, отказоустойчивости и восстанавливаемости, которая, в свою очередь, определяет продолжительность времени бездействия после каждого отказа.

5.3 Отказоустойчивость – способность системы, продукта или компонента работать как предназначено, несмотря на наличие дефектов программного обеспечения или аппаратных средств.

5.4 Восстанавливаемость – способность продукта или системы восстановить данные и требуемое состояние системы в случае прерывания или сбоя. В некоторых случаях после сбоя вычислительная система находится в нерабочем состоянии некоторое время, продолжительность которого определяется ее восстанавливаемостью.

6. Защита, защищенность – степень защищенности информации и данных, обеспечиваемая продуктом или системой путем ограничения доступа людей, других продуктов или систем к данным в соответствии с типами и уровнями авторизации. Защищенность вносит свой вклад в доверие.

Защищенность подразумевает главным образом защиту данных путем разграничения прав. Например, пользователи продукта должны обладать разными правами на изменение сущностей приложения, например, администратор имеет возможность редактировать все сущности системы, оператор (менеджер) только сущности, создаваемые клиентами, клиенты могут создавать и редактировать только свои сущности. В равной степени это относится к оборудованию и стороннему ПО, чтобы защитить данные в системе от предумышленной или случайной (ошибочной) порчи.

6.1 Конфиденциальность – обеспечение продуктом или системой ограничения доступа к данным только для тех, кому доступ разрешен.

6.2 Целостность – степень предотвращения системой, продуктом или компонентом несанкционированного доступа или модификации компьютерных программ или данных.

7. Сопровождаемость, модифицируемость – результативность и эффективность, с которыми продукт или система могут быть модифицированы предполагаемыми специалистами по обслуживанию. Модификация может включать в себя исправления, улучшения или адаптацию программного обеспечения к изменениям в условиях использования, в требованиях и функциональных спецификациях. Модификации могут быть выполнены как специализированным техническим персоналом, так и рабочим или операционным персоналом и конечными пользователями. Сопровождаемость включает в себя установку разного рода обновлений. Сопровождаемость можно интерпретировать либо как присущее продукту или системе свойство, упрощающее процесс обслуживания, либо как качество при использовании, проверенное на практике специалистами по обслуживанию в целях поддержки продукта или системы.

На показатель сопровождаемости влияют: качество кода, документируемость кода (наличие комментариев), реализация шаблонов проектирования, наличие различной сопровождающей документации, насколько взаимозависимы элементы системы и т. п.

7.1 Модульность – степень представления системы или компьютерной программы в виде отдельных блоков таким образом, чтобы изменение одного компонента оказывало минимальное воздействие на другие компоненты.

7.2 Возможность многократного использования – степень, в которой актив может быть использован в нескольких системах или в создании других активов. Под активом подразумевается что-либо, имеющее ценность для человека или организации, т.е. такие продукты деятельности, как документы требований, модули исходного кода и т.д.

Иными словами, можно ли использовать приложение или его компонент (библиотеку, модуль, документацию) для разработки других приложений.

7.3 Анализируемость – степень простоты оценки влияния изменений одной или более частей на продукт или систему или простоты диагностики продукта для выявления недостатков и причин отказов, или простоты идентификации частей, подлежащих изменению. Конкретная реализация продукта или системы может включать в себя механизмы анализа собственных дефектов и формирования отчетов об отказах и других событиях.

Эти определения в ГОСТе даже моего кота пугают, а он прошел через многое. Если приложение разбито на отдельные независимые модули (например, чтобы составить наборы тестов для каждого отдельного модуля, не затрагивая работу всей системы в целом), то его проще анализировать, тестировать и сопровождать. Если осуществляется фиксация ошибок (через лог) и ведение статистики, то это также влияет на анализируемость.

7.4 Модифицируемость – степень простоты эффективного и рационального изменения продукта или системы без добавления дефектов и снижения качества продукта. Модифицируемость – это сочетание изменяемости и устойчивости. Реализация модификации включает в себя кодирование, разработку, документирование и проверку изменений. На модифицируемость могут оказывать влияние модульность и анализируемость.

7.5 Тестируемость – степень простоты эффективного и рационального определения для системы, продукта или компонента критериев тестирования, а также простоты выполнения тестирования с целью определения соответствия этим критериям.

8. Переносимость, мобильность – степень простоты эффективного и рационального переноса системы, продукта или компонента из одной среды (аппаратных средств, программного обеспечения, операционных условий или условий использования) в другую.

Переносимость можно интерпретировать либо как присущее продукту или системе свойство продукта или системы, упрощающее процесс переноса, либо как качество при использовании, предназначенное для переноса продукта или системы.

8.1 Адаптируемость – степень простоты эффективной и рациональной адаптации для отличающихся или усовершенствованных аппаратных средств, программного обеспечения, других операционных сред или условий использования. В адаптируемость входит и масштабируемость внутренних потенциальных возможностей (например, экранных полей, таблиц, объемов транзакции, форматов отчетов и т.д.). Адаптация может быть выполнена как специализированным техническим персоналом, так и рабочим или операционным персоналом и конечными пользователями. Если система должна быть адаптирована конечным пользователем, то адаптируемость соответствует пригодности для индивидуализации.

8.2 Устанавливаемость – степень простоты эффективной и рациональной, успешной установки и/или удаления продукта или системы в заданной среде. Если продукт или система должны устанавливаться конечным пользователем, устанавливаемость может повлиять на результирующие функциональную целесообразность и управляемость.

8.3 Взаимозаменяемость – способность продукта заменить другой конкретный программный продукт для достижения тех же целей в тех же условиях. Взаимозаменяемость новой версии программного продукта важна для пользователя при обновлении продукта. Во взаимозаменяемость могут быть включены атрибуты как устанавливаемости, так и адаптируемости.

Всё прочитали? А сын маминой подруги прочитал всё!

2.5 Содержание отчета

Отчет о работе должен содержать:

1. Титульный лист
2. Цель работы
3. Задание на лабораторную работу и вариант
4. Модель качества при использовании
5. Выводы по работе
6. Использованные источники (2-4 источника)
7. Приложение (листинг программы)

2.6 Пример выполнения работы

Модель должна включать описание степени характеристики (низкая, средняя, высокая или отсутсвует) и обстоятельства, которые позволили сделать такое заключение.

Таблица 2.6.1 — Характеристики и подхарактеристики качества при использовании.

	Характеристика	Оценка	Пояснение
1.	Эффективность	Высокая	Приложение в полной мере отвечает критериям точности и полноте получаемых данных результатов при использовании.
2.	Производительность	Средняя	Приложение не предполагает контроль израсходованных ресурсов, например, при недостаточном количестве ОЗУ может произойти «обрушение» процесса (исключение uncaughtException).
3.	Удовлетворенность	Высокая	Приложение отвечает всем требованиям в контексте использования.
3.1	Полноценность	Высокая	Приложение обеспечивает в полной мере выполнение всех требуемых функций, что выражается в абсолютную полезность для пользователя при использовании.
3.2	Доверие	Среднее	Данное ПО является «прослойкой» для клиента приложения (сайта). Поскольку приложение выполняется на сервере, клиент не знает, что может произойти с его данными при их отправке их приложению.
3.3	Удовольствие	Среднее	Приложение не предполагает GUI, однако, его назначение облегчает использование сайта.
3.4	Комфорт	Низкое	При большой нагрузке (от 300 одновременных подключениях) приложение «отказывается» работать и происходит периодическое «падение» процессов.
4.	Свобода от риска	Отсутствует	В приложении не реализованы механизмы для снижения различного рода рисков.
4.1	Смягчение отрицательных последствий экономического риска	Отсутствует	В приложении не реализованы механизмы для снижения различного рода рисков.
4.2	Смягчение отрицательных последствий риска здоровья и безопасности	Отсутствует	В приложении не реализованы механизмы для снижения различного рода рисков.
4.3	Смягчение отрицательных последствий экологического риска	Отсутствует	В приложении не реализованы механизмы для снижения различного рода рисков.

5.	Покрытие контекста	Среднее	Приложение в полной мере выполняет свои функции в рамках изначально определенных задач и целей, однако выход за спецификации использования не предусмотрен.
5.1	Полнота контекста	Высокое	Приложение в полной мере выполняет свои функции в рамках указанных условий использования.
5.2	Гибкость	Средняя	При необходимости приложение может быть расширено за рамки изначально предусмотренных спецификаций использования.

Таблица 2.6.2 — Модель качества продукта

	Характеристика	Оценка	Пояснение
1.	Функциональная пригодность	Высокая	Приложение в полной мере отвечает всем заявленным и предполагаемым требованиям.
1.1	Функциональная полнота	Высокая	Приложение охватывает все цели и задачи пользователя при использовании (в рамках контекста).
1.2	Функциональная корректность	Высокая	Приложение обеспечивает необходимую степень точности результатов использования.
1.3	Функциональная целесообразность	Высокая	Приложение выполняет все необходимые действия вместо клиента.
2.	Уровень производительности	Средний	Приложение не требует много ресурсов, однако при высокой нагрузке производительность снижается в разы.
2.1	Временные характеристики	Низкие	Отклик приложения при высокой нагрузке может достигать до нескольких секунд (при обычной нагрузке около 20-50мс).
2.2	Использование ресурсов	Низкое	При высокой нагрузке все пользовательские запросы отправляются в очередь, из-за чего node.js может исчерпать весь объем ОЗУ. После определенного порога происходит обрушение процесса и работа приложения прекращается.
2.3	Потенциальные возможности	Средние	Стабильная работа приложения гарантируется при 150 одновременных подключениях клиентов. Абсолютный зарегистрированный максимум за (примерно) год использования – 471.
3.	Совместимость	Отсутствует	У приложения отсутствует возможность обмениваться данными с другими приложениями или системами.
3.1	Сосуществование	Высокая	Приложение не влияет на другие процессы при небольшом количестве пользователей.
3.2	Интероперабельность	Отсутствует	Приложение не имеет возможности обмениваться и использовать информацию от других приложений/систем/компонентов.
4.	Удобство использования	Среднее	В приложении отсутствуют понятные пользователю параметры и способы использования.
4.1	Определимость пригодности	Низкая	Пользователь с трудом поймет назначение приложения.
4.2	Изучаемость	Высокая	Приложение может быть использовано только через GUI сайта, поэтому пользователь непосредственно не контактирует с приложением. Запуск приложения осуществляется в соответствии со спецификой работы сайта.

4.3	Управляемость	Средняя	Имеются параметры передачи ключей капчи, однако отсутствует настройка получения событий или изменение параметра ожидания запроса от LongPoll-сервера.
4.4	Защищенность от ошибки пользователя	Высокая	При неверных параметрах приложение выдаст ошибку (работа с неверными параметрами будет прекращена).
4.5	Эстетика пользовательского интерфейса	Отсутствует	Приложение не имеет пользовательского интерфейса.
4.6	Доступность	Низкая	Приложение может работать строго в одном контексте (получение и обработка данных от LongPoll-сервера социальной сети ВКонтакте).
5.	Надежность	Средняя	Uptime (время непрерывной работы) приложения высокий, однако, при больших нагрузках или неисправностей оборудования приложение может не функционировать.
5.1	Завершенность	Высокая	Приложение полностью соответствует требованиям надежности при нормальной работе.
5.2	Готовность	Средняя	Примерно за год использования приложение находилась в рабочем состоянии ~ 99.8% времени. Частота падений зависит от количества одновременных пользователей.
5.3	Отказоустойчивость	Низкая	При нарушении работы сервера и/или отсутствия необходимого запаса ОЗУ может произойти «падение» приложения.
5.4	Восстанавливаемость	Низкая	Восстановить данные, с которыми приложение работало в момент отказа, невозможно. Однако, если иметь в виду данные, которые запрашивает приложение от LongPoll-сервера, то их можно получить, передав точно такие же параметры, какие были переданы при первом запросе.
6.	Защищенность	Средняя	Все пользователи приложения имеют одинаковые права.
6.1	Конфиденциальность	Высокая	Приложение не будет работать без передачи специального параметра userAccessToken. В противном случае, приложение выдаст ошибку.
6.2	Целостность	Высокая	Приложение не имеет доступа к модификации данных в памяти.
7.	Сопровождаемость	Средняя	Приложение разбито на отдельные функции/методы, выполняющие поставленные задачи, которые понятны по контексту и комментариям в коде, однако, использованный подход (при реализации) не удобен для сопровождения приложения.
7.1	Модульность	Высокая	Приложение использует несколько отдельных независимых модулей npm. Однако, само приложение не может выступать как модуль.
7.2	Возможность многократного использования	Отсутствует	Данное приложение нельзя использовать как модуль в другом приложении или системе.
7.3	Анализируемость	Низкая	Отсутствует лог, покрытие тестами не выполнено.
7.4	Модифицируемость	Средняя	Есть зависимости от пакетов npm, но внесение изменений в код осуществляется без контроля целостности и нарушения логики работы приложения.
7.5	Тестируемость	Низкая	Тестирование приложения является в достаточной степени трудоемкой задачей ввиду технических особенностей и контекста использования.

8.	Переносимость	Высокая	Приложение может работать в любой ОС, при наличии установленных программ node.js и npm.
8.1	Адаптируемость	Высокая	Приложение не требует изменений при переносе в другую среду.
8.2	Устанавливаемость	Высокая	Установка приложения не требуется. Приложение требует наличие программ node.js и npm.
8.3	Взаимозаменяемость	Низкая	Приложение выполняет специфические функции, поэтому оно не может выступать как альтернатива для иных приложений.

Пример вывода:

В результате выполнения данной лабораторной работы были получены навыки описания модели качества программного продукта при использовании в соответствии со стандартом ГОСТ Р ИСО/МЭК 25010-2015.

Составлена модель качества программного продукта при использовании, включающая в себя пять характеристик: эффективность, производительность, удовлетворенность, свободу от риска и покрытие контекста.

В ходе выполнения ЛР некоторые характеристики (или подхарактеристики) не были корректно оценены в виду отсутствия полной информации о приложении или статистических данных его работы. В частности:

- перечислите характеристики или подхарактеристики, которые не удалось корректно оценить и кратко опишите причину.

2.7 Контрольные вопросы

1) Назначение ГОСТ Р ИСО/МЭК 25010-2015?

2) Какие показатели вносят вклад в характеристику «доверие»?

3) Что подразумевает собой защищенность?

4) Что влияет на степень тестируемости ПП?

5) Что подразумевает собой переносимость?

6) Чем адаптируемость отличается от переносимости?

3. ЛАБОРАТОРНАЯ РАБОТА «МЕТРИКИ ОЦЕНКИ КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ И СЛОЖНОСТИ ПОТОКА ДАННЫХ»

3.1 Цель работы

Целью данной работы является изучение методик оценки качества и сложности потока данных программы на основе лексического анализа исходного кода.

3.2 Задание на лабораторную работу

Составить в виде таблицы перечень операторов и операндов, которые используются в программе из ЛР №1. Произвести расчет по метрике Холстеда.

Произвести оценку сложности программного обеспечения на основе метрики Джилба и определить информационную прочность программного модуля по метрике Чепина.

Выполнить расчет среднего числа строк для функций и произвести оценку уровня комментируемости кода.

- для оценки «хорошо» объем исходного листинга программы: 1-2 стр;
- для оценки «отлично» минимальный объем исходного листинга программы: 3 стр.

3.3 Порядок выполнения работы

1. Составить перечень операторов и операндов в программе;
2. Подсчитать общее количество;
3. Произвести расчет по упрощенной метрике Холстеда (без теоретического словаря), представленной в подразделе 3.4;
4. Произвести расчет по метрике Джилба;
5. Произвести расчет по метрике Чепина;
6. Произвести расчет среднего арифметического количества строк для функций (или методов) и выполнить оценку уровня комментируемости кода;
7. Ознакомиться с требованиями по содержанию отчета в подразделе 3.5;
8. Написать отчет о работе.

3.4 Теоретический материал

Метрика ПО — мера, позволяющая получить численное значение некоторого свойства ПО или его спецификаций [3].

Качество ПО — способность программного продукта при заданных условиях удовлетворять установленным или предполагаемым потребностям.

Критерий качества ПО — это нефункциональное требование к программе, которое обычно не описывается в договоре с заказчиком, но, тем не менее, является желательным требованием, повышающим качество программы.

Выделяют следующие критерии качества:

1. Понятность: Назначение ПО должно быть понятным, из самой программы и документации.
2. Полнота: Все необходимые части программы должны быть представлены и полностью реализованы.

3. Краткость: Отсутствие лишней, дублирующейся информации. Повторяющиеся части кода должны быть преобразованы в вызов общей процедуры. То же касается и документации.

4. Портируемость: Лёгкость в адаптации программы к другому окружению: другой архитектуре, платформе, операционной системе или её версии.

5. Согласованность: По всей программе и в документации должны использоваться одни и те же соглашения, форматы и обозначения.

6. Сопровождаемость: Насколько сложно изменить программу для удовлетворения новых требований. Это требование также указывает, что программа должна быть хорошо документирована, не слишком запутана, и иметь резерв роста по использованию ресурсов (память, процессор).

Сопровождение (поддержка) программного обеспечения — процесс улучшения, оптимизации и устранения дефектов программного обеспечения (ПО) после передачи в эксплуатацию. Сопровождение ПО — это одна из фаз жизненного цикла программного обеспечения, следующая за фазой передачи ПО в эксплуатацию. В ходе сопровождения в программу вносятся изменения, с тем, чтобы исправить обнаруженные в процессе использования дефекты и недоработки, а также для добавления новой функциональности, с целью повысить удобство использования (юзабилити) и применимость ПО.

7. Тестируемость: Позволяет ли программа выполнить проверку приёмочных характеристик, поддерживается ли возможность измерения производительности.

8. Удобство использования: Простота и удобство использования программы. Это требование относится прежде всего к интерфейсу пользователя.

9. Надёжность: Отсутствие отказов и сбоев в работе программ, а также простота исправления дефектов и ошибок.

Если говорить об определении надежности ПО, закрепленном в ГОСТ, то оно звучит следующим образом: Надежность программного обеспечения — способность программного продукта безотказно выполнять определенные функции при заданных условиях в течение заданного периода времени с достаточно большой вероятностью.

10. Структурированность.

11. Эффективность: Насколько рационально программа относится к ресурсам (память, процессор) при выполнении своих задач.

12. Безопасность.

В той или иной степени, перечисленные критерии качества, были описаны в ЛР №1-2.

Метрика Холстеда изначально разрабатывалась для оценки качества кода, написанного на FORTRAN, сейчас данную метрику используют скорее для оценки сложности ПО [4].

Метрики сложности программ принято разделять на три основные группы:

1. Метрики размера программ;
2. Метрики сложности потока управления программ (ЛР №4);
3 Метрики сложности потока данных программ.

Метрики первой группы базируются на определении количественных характеристик, связанных с размером программы, и отличаются относительной простотой. К наиболее известным метрикам данной группы относятся число операторов программы, количество строк исходного текста и набор метрик Холстеда. Метрики этой группы ориентированы на анализ исходного текста программ. Поэтому они могут использоваться для оценки сложности промежуточных продуктов разработки.

Метрики второй группы базируются на анализе управляющего графа программы. Представителем данной группы является метрика Маккейба. Управляющий граф программы, который используют метрики данной группы, может быть построен на основе алгоритмов модулей. Поэтому метрики второй группы могут также применяться для оценки сложности промежуточных продуктов разработки.

Метрики третьей группы базируются на оценке использования, конфигурации и размещения данных в программе. В первую очередь это касается глобальных переменных. К данной группе относятся метрики Чепина.

Метрики Холстеда позволяют частично учесть возможность записи одной и той же функциональности разным количеством строк и операторов. Они основаны на большом числе количественных показателей, таких, как число уникальных операторов программы, число уникальных операндов программы, общее число операторов, общее число операндов, теоретическое число уникальных операторов и теоретическое число уникальных операндов. Через эти показатели различными формулами определяются уровень качества программирования, сложность понимания программы, трудоемкость кодирования программы, уровень языка выражения, информационное содержание программы (данная характеристика позволяет определить умственные затраты на создание программы) и оценка интеллектуальных усилий при разработке программы, характеризующая число требуемых элементарных решений при написании программы [5].

Метрика Холстеда (без теоретического словаря):

n_1 — число уникальных операторов программы, включая символы разделители, имена процедур и знаки операций (словарь операторов);

n_2 — число уникальных операндов программы (словарь операндов);

N_1 — общее число операторов в программе;

N_2 — общее число операндов в программе;

$n = n_1 + n_2$ — словарь программы;

$N = N_1 + N_2$ — длина программы;

$V=N*\log_2(n)$ — объем программы.

Метрика Джилба показывает сложность программного модуля на основе насыщенности кода программы условными операторами:

$$cl = CL / n,$$

где CL – количество управляющих операторов, n – общее количество операторов в программе.

Метрика Чепина оценивает информационную прочность отдельно взятого модуля по характеру использования переменных из списка ввода-вывода. Всё множество переменных разбивается на 4 группы: P (вводимые для расчетов и для обеспечения вывода), M (модифицируемые, или создаваемые внутри программы), C (управляющие), T (паразитные, то есть не используемые). Переменные, выполняющие несколько функций, учитываются в каждой функциональной группе. Метрика Чепина выражается формулой

$$Q = a_1 \times P + a_2 \times M + a_3 \times C + a_4 \times T,$$

где a_1, a_2, a_3, a_4 — весовые коэффициенты. Данные коэффициенты следует принять равными 1, 2, 3 и 0.5 соответственно, учитывается, что переменные группы C влияют также на поток управления программы, поэтому у них самый высокий вес. Паразитные переменные не увеличивают сложность потока данных, но затрудняют понимание программы.

Оценка среднего числа строк для функций (методов) производится на основе среднего арифметического.

$$S = S_{сум} / S_{кол}$$

где $S_{сум}$ – суммарное количество срок в классах/функциях/модулях программы; $S_{кол}$ – количество классов/функций/модулей в программе.

Оценка уровня комментируемости: $F = (N_{ком} / N_{стр}) \times 100\%$,

где $N_{ком}$ – количество комментариев в программе; $N_{стр}$ – количество строк исходного текста (с учетом комментариев).

Потенциальные недостатки подхода, связанного с метриками ПО:

- неэтичность: утверждается, что неэтично судить о производительности программиста по метрикам, введенным для оценки эффективности программного кода. Такие известные метрики, как количество строк кода и цикломатическая сложность, часто дают поверхностное представление об «удачности» выбора того или иного подхода при решении поставленных задач, однако, нередко они рассматриваются, как инструмент оценки качества работы разработчика. Такой подход достаточно часто приводит к обратному эффекту, приводя к появлению в коде более длинных конструкций и избыточных необязательных методов.
- замещение «управления людьми» на «управление цифрами», которое не учитывает опыт сотрудников и их другие качества;
- искажение: процесс измерения может быть искажён за счёт того, что сотрудники знают об измеряемых показателях и стремятся оптимизировать эти показатели, а не свою работу. Например, если количество строк исходного кода является важным показателем, то программисты будут стремиться писать как можно больше строк и не будут использовать способы упрощения кода, сокращающие количество строк.
- неточность: нет метрик, которые были бы одновременно и значимы и достаточно точны. Количество строк кода — это просто количество строк, этот показатель не даёт представление о сложности решаемой проблемы. Анализ функциональных точек был разработан с целью лучшего измерения сложности кода и спецификации, но он использует личные оценки измеряющего, поэтому разные люди получат разные результаты.

3.5 Содержание отчета

Отчет о работе должен содержать:

1. Титульный лист
2. Цель работы
3. Задание на лабораторную работу и вариант
4. Метрика качества ПО (метрика Холстеда)
5. Метрики сложности потока данных
   5.1 Метрика Джилба
   5.2 Метрика Чепина
6. Количественные оценки;
   6.1 Оценка среднего числа строк для функций (методов, классов, модулей)
   6.2 Оценка уровня комментируемости кода
7. Выводы по работе
8. Использованные источники (2-4 источника)
9. Приложение (листинг программы)

3.6 Пример выполнения работы

Метрика качества ПО (метрика Холстеда)

Обратите внимание, что в качестве операторов в таблице указываются не только уникальные операторы программы и их количество, но и символы-разделители, имена процедур и знаки операций — всё это учитывается метрикой Холстеда.

Таблица 3.6 — Перечень операторов и операндов, используемых в программе

	Оператор	Операнд	Общее количество
1	=		35
2	==		1
3	;		60
4	,		121
5	var		6
6	if-else		8
7	""		64
8	:		36
9	||		2
10	&&		3
11	!		7
12	|		3
13	{ }		18
14	[ ]		14
15	+=		2
16	new		2
17	++		1
18	+		11
19	<		1
20	? :		1
21	. (разделитель дроби)		1
22	. (оператор доступа)		66
23	try-catch		3
24	return		10
25	function		16
			492
1		require	4
2		APIDOG_LONGPOLL_RESULT_CODE_OK	2
3		APIDOG_LONGPOLL_RESULT_CODE_CAPTCHA	2

4		APIDOG_LONGPOLL_RESULT_CODE_API_ERROR	2
5		APIDOG_LONGPOLL_RESULT_CODE_SERVER_ISSUE	4
6		APIDOG_LONGPOLL_RESULT_CODE_EMPTY_RESPONSE	2
7		APIDOG_LONGPOLL_RESULT_CODE_NOT_JSON	2
8		APIDOG_LONGPOLL_RESULT_CODE_FAILED	3
9		APIDOG_LONGPOLL_RESULT_CODE_API_REQUEST_UNKNOWN	3
10		LP_MODE_ATATCHS	2
11		LP_MODE_EXTENDS	2
12		LP_MODE_PTS	1
13		LP_MODE_EXTRA_FRIENDS	2
14		LP_MODE_RANDOM_ID	2
15		LP_VERSION_NORMAL	2
16		LP_VERSION_GROUP_NEGATIVE	2
17		config	5
18		version	5
19		mode	5
20		server	7
21		http	3
22		https	2
23		createServer	1
24		function	16
25		request	4
26		response	32
27		requestData	2
28		url	6
29		GET	11
30		userAccessToken	8
31		captchaId	3
32		captchaKey	3
33		outputJSON	13
34		null	12
35		writeHead	1
36		200	1
37		getLongPollServer	4
38		id	2

39		key	10
40		waitForLongPoll	3
41		ts	8
42		listen	1
43		4000	1
44		captcha	5
45		n	5
46		end	4
47		data	19
48		error	5
49		error_code	1
50		captchaImg	1
51		captcha_sid	2
52		captcha_img	1
53		source	1
54		host	4
55		querystring	5
56		port	2
57		80	1
58		443	1
59		path	4
60		result	4
61		String	2
62		setEncoding	2
63		on	6
64		chunk	4
65		json	10
66		JSON	2
67		parse	2
68		toString	1
69		failed	3
70		e	6
71		stringify	3
72		v	2
73		push	1
74		25000	1
75		write	1

76		API	2
77		method	2
78		params	2
79		callback	3
80		res	4
81		apiResponse	3
82		7500	1
			334

$n_1 = 25$ — число уникальных операторов программы;
$n_2 = 82$ — число уникальных операндов программы (словарь операндов);
$N_1 = 492$ — общее число операторов в программе;
$N_2 = 334$ — общее число операндов в программе;

$n = n_1 + n_2 = 25 + 82 = 107$ — словарь программы;
$N = N_1 + N_2 = 492 + 334 = 826$ — длина программы;
$V = N \times \log_2(n) = 826 \times \log_2(107) = 5568{,}451731 \approx 5568.45$ — объем программы.

Метрики сложности потока данных

Метрика Джилба:

$$cl = CL / n = 9 / 107 = 0{,}08411215 \approx 0.084$$

Метрика Чепина:

$$Q = a_1 \times P + a_2 \times M + a_3 \times C + a_4 \times T = 1 \times 5 + 2 \times 7 + 3 \times 18 + 0.5 \times 5 = 75.5$$

Количественные оценки.

Оценка среднего числа строк для функций (методов):

$$S = S_{сум} / S_{кол} = (20 + 5 + 32 + 31 + 24) / 5 = 22.4$$

Оценка уровня комментируемости:

$$F = (N_{ком} / N_{стр}) \times 100\% = (24 / 160) \times 100\% = 15\%$$

Пример вывода:

В результате выполнения данной лабораторной работы были изучены методы оценки качества и сложности потока данных программы. Выполнены расчеты соответствующих оценок на основе лексического анализа исходного кода.

Результатами работы являются следующие оценки:
- объем программы по метрике Холстеда: 5568,45
- сложность потока данных по метрике Джилба: 0,084
- сложность потока данных по метрике Чепина: 75,5
- среднее число строк для функций (методов): 22,4
- уровень комментируемости кода: 15%

Оценки, полученные по метрикам Холстеда, Джилба и Чепина несут скорее исследовательский характер, чем практический, и должны рассматриваться в сравнении с аналогичными значениями.

3.7 Контрольные вопросы

1) Дайте определение метрики.

2) Дайте определение качеству ПО.

3) Перечислите критерии качества ПО.

4) Какие выделяют группы метрик сложности ПО?

5) Приведите примеры метрик размера программ?

6) Приведите примеры метрик сложности потока данных.

7) Опишите метрику Холстеда.

8) Опишите метрику Джилба.

9) Опишите метрику Чепина.

10) Что представляет собой паразитная переменная?

11) Почему паразитные переменные необходимо учитывать, на что они влияют?

12) Как появляются паразитные переменные?

13) Что представляют собой управляющие переменные?

14) Почему управляющие переменные обладают наибольшим коэффициентом?

15) Перечислите потенциальные недостатки подхода, связанного с метриками?

4. ЛАБОРАТОРНАЯ РАБОТА
«МЕТРИКА ОЦЕНКИ СЛОЖНОСТИ ПОТОКА УПРАВЛЕНИЯ»

4.1 Цель работы

Целью данной работы является изучение способов оценки сложности программного обеспечения на основе метрики Маккейба.

4.2 Задание на лабораторную работу

Оценить структурную сложность программы по метрике Маккейба:
- Начертить блок-схему алгоритма программы;
- По блок-схеме построить граф потока управления;
- Выделить линейно-независимые маршруты и циклы;
- Вычислить цикломатическую сложность;
- Построить матрицы смежности и достижимости.

- для оценки «хорошо» объем исходного листинга программы: 1-2 стр;
- для оценки «отлично» минимальный объем исходного листинга программы: 3 стр.
Выполнить профилирование кода и выявить горячие точки программы.

4.3 Порядок выполнения работы

1. Ознакомиться с теоретическим материалом, изложенным в 4.4;
2. Начертить блок-схему алгоритма программы;
3. Построить граф потока управления программы;
4. Выделить линейно-независимые маршруты и циклы;
5. Произвести оценку цикломатической сложности (метрика Маккейба);
6. Построить матрицы смежности и достижимости;
7. Для оценки «отлично» выполнить профилирование кода и выявить горячие точки;
8. Ознакомиться с требованиями по содержанию отчета в подразделе 4.5;
9. Написать отчет о работе.

4.4 Теоретический материал

Цикломатическая сложность, или метрика Маккейба (McCabe) является наиболее известной и широко используемой при создании инструментария для оценки сложности программ на различных языках [5]. Цикломатическая сложность вычисляется для графа потока управления процедуры или функции по формуле:

$$M(G) = E – N + P,$$

где E – количество дуг, N – количество вершин, P – число компонент связности графа G, величина P измеряется количеством замыкающих дуг, необходимых для преобразования исходного графа в максимальный сильно связный граф (это такой граф, в котором любые две вершины взаимно достижимы).

Для корректных программ сильно связный граф потока управления получается замыканием дугами вершин, соответствующих точкам выхода, на вершины точек входа. Метрика Маккейба может вычисляться для всей системы, если построен общий граф потока управления на основе графа вызовов, либо для отдельных модулей, классов, методов и

других единиц. Для правильной и хорошо структурированной программы с одной точкой входа и одной точкой выхода P = 1.

Вычисление цикломатической сложности программного модуля осуществляется по величинам, определяемым по максимально связному графу – который получается из исходного графа замыканием дуги из конечной вершины в начальную.

В результате в максимально связном графе появляются маршруты между любыми двумя вершинами. Цикломатическое число в максимально связном графе равно максимальному числу его линейно-независимых циклов, то есть циклов, которые не содержат в себе другие циклы. Каждый линейно-независимый маршрут или цикл отличается от всех остальных хотя бы одной вершиной или дугой, т. е. его структура не может быть полностью образована компонентами других маршрутов.

Как будет ясно из примера расчета (на самом деле нет, но вы теперь будете об этом знать), оценка цикломатической сложности не различает циклические и условные конструкции. В процессе автоматизированного вычисления показателя цикломатической сложности, как правило, применяется упрощенный подход, в соответствии с которым построение графа не осуществляется, а вычисление показателя производится на основании подсчета числа операторов управляющей логики (if, switch etc.) и возможного количества путей исполнения программы [6]. Цикломатическое число Маккейба показывает требуемое количество проходов для покрытия всех контуров сильносвязанного графа или количества тестовых прогонов программы, необходимых для исчерпывающего тестирования по принципу «работает каждая ветвь». Данный показатель напрямую связан с профилированием.

Профилирование — сбор характеристик работы программы, таких как время выполнение отдельных фрагментов (обычно подпрограмм), число верно предсказуемых условных переходов, число кэш-промахов, etc. [7]. Инструмент, используемый для анализа работы, называют профилировщиком или профайлером. Обычно выполняется совместно с оптимизацией программы.

Горячая точка — участок кода в программе, на который приходится большая часть исполняемых инструкций процессора или на исполнение которого процессор затрачивает очень много времени. Критичным примером второго случая является взаимная блокировка (deadlock — тупик) — ситуация в многозадачной среде или СУБД, при которой несколько процессов находятся в состоянии бесконечного ожидания ресурсов, занятых самими этими процессами.

Покрытие кода — мера, используемая при тестировании ПО. Она показывает процент, насколько исходный код был протестирован.

Анализ покрытия — процесс выявления неиспользуемых участков кода при помощи, например, многократного запуска программы.

Выделяют следующие способы покрытия кода:

1. Покрытие операторов (каждая ли строка исходного кода была выполнена и протестирована);
2. Покрытие условий (каждая ли точка решения была выполнена и протестирована);
3. Покрытие путей (все ли возможные пути через заданную часть кода были выполнены и протестированы);
4. Покрытие функций (каждая ли функция программы была выполнена);
5. Покрытие вход/выход (все ли вызовы функций и возвраты из них были выполнены);
6. Покрытие значений параметров (все ли типовые и граничные значения параметров были проверены).

Тестирование ПО — процесс исследования, испытания программного продукта, имеющий две различные цели:

1. Продемонстрировать разработчикам и заказчикам, что программа соответствует требованиям;

2. Выявить ситуации, в которых поведение программы является неправильным, нежелательным или не соответствующим спецификации.

Граф потока управления — множество всех возможных путей исполнения программы, представленное в виде графа.

В графе потока управления каждый узел (вершина) графа соответствует базовому блоку — прямолинейному участку кода, не содержащего в себе ни операций передачи управления, ни точек, на которые управление передается из других частей программы.

Блок, не связанный со входным блоком, считается недостижимым («мёртвый» код). Достижимость — одно из свойств графа, используемое при оптимизациях. Недостижимый блок может быть удален из программы. Матрица достижимости позволяет сравнительно просто выделить циклы и некоторые структурные некорректности (лишние тупиковые участки графа программы). При построении матрицы достижимости на позиции (i;j) помещается 1, если из вершины i, можно перейти к вершине j за любое число шагов.

Матрица смежности — один из способов представления графа в виде матрицы, в которой единица располагается в позиции (i;j), если из вершины i, можно перейти к вершине j за один шаг.

Матрица смежности и списки смежности являются основными структурами данных, которые используются для программного представления графов.

Использование матрицы смежности предпочтительно только в случае неразреженных (плотных) графов, с большим числом рёбер, так как она требует хранения по одному биту данных для каждого элемента. Если граф разрежён, то большая часть памяти напрасно будет тратиться на хранение нулей, зато в случае неразреженных графов матрица смежности достаточно компактно представляет граф в памяти, что может быть на порядок лучше списков смежности.

В математике плотным графом называется граф, в котором число рёбер близко к максимальному. Граф с противоположным свойством, имеющий малое число рёбер, называется разреженным графом. Разница между разреженным и плотным графом расплывчата и зависит от контекста.

4.5 Содержание отчета

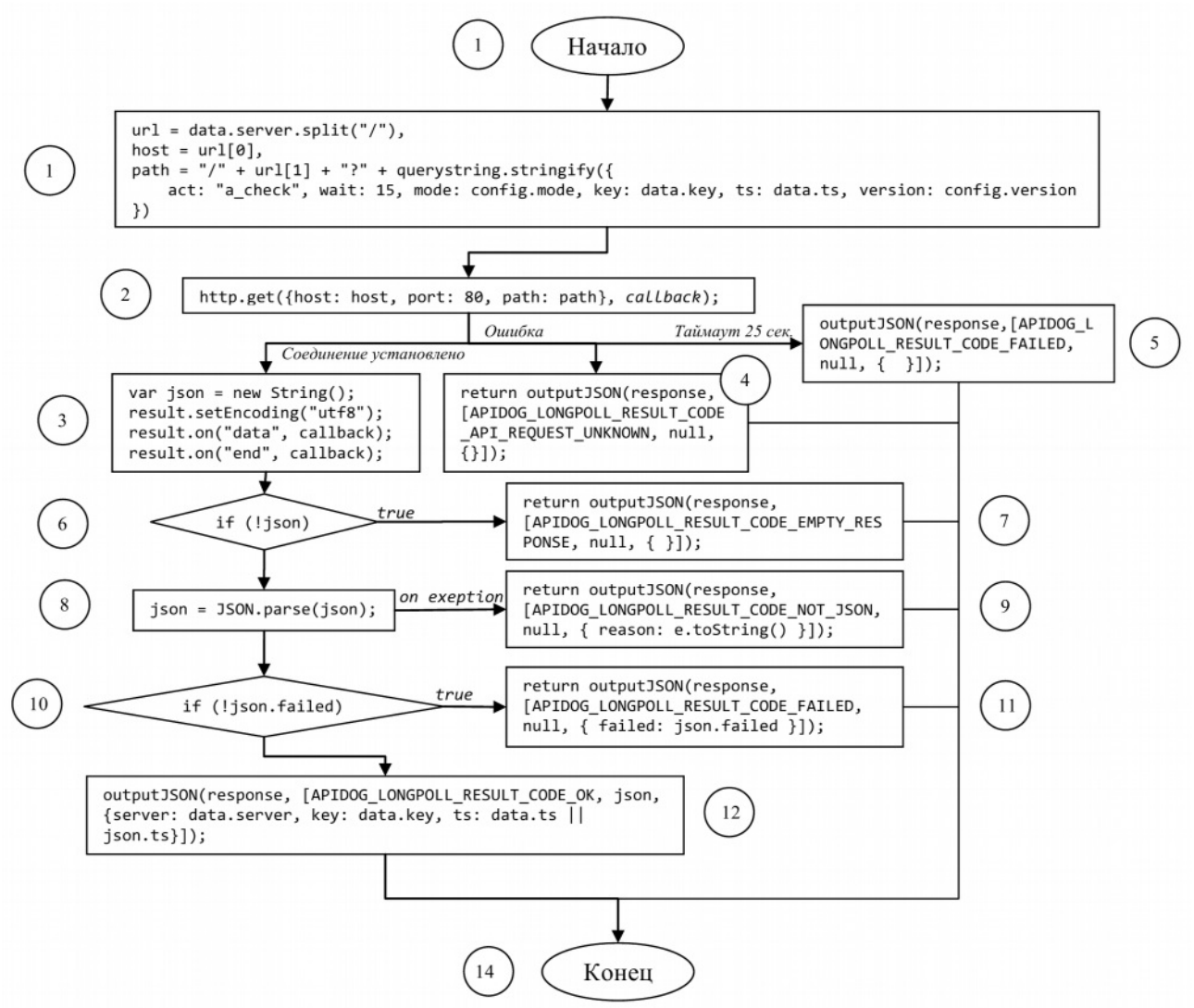
Отчет о работе должен содержать:

1. Титульный лист
2. Цель работы
3. Задание на лабораторную работу и вариант
4. Оценка сложности потока управления
   - 4.1 Блок-схема алгоритма
   - 4.2 Граф потока данных
   - 4.3 Линейно независимые маршруты и циклы
   - 4.4 Расчет цикломатической сложности (метрика Маккейба)
5. Матрицы смежности и достижимости;
6. Для оценки «отлично» результаты профилирования кода;
7. Выводы по работе
8. Использованные источники (2-4 источника)
9. Приложение (листинг программы)

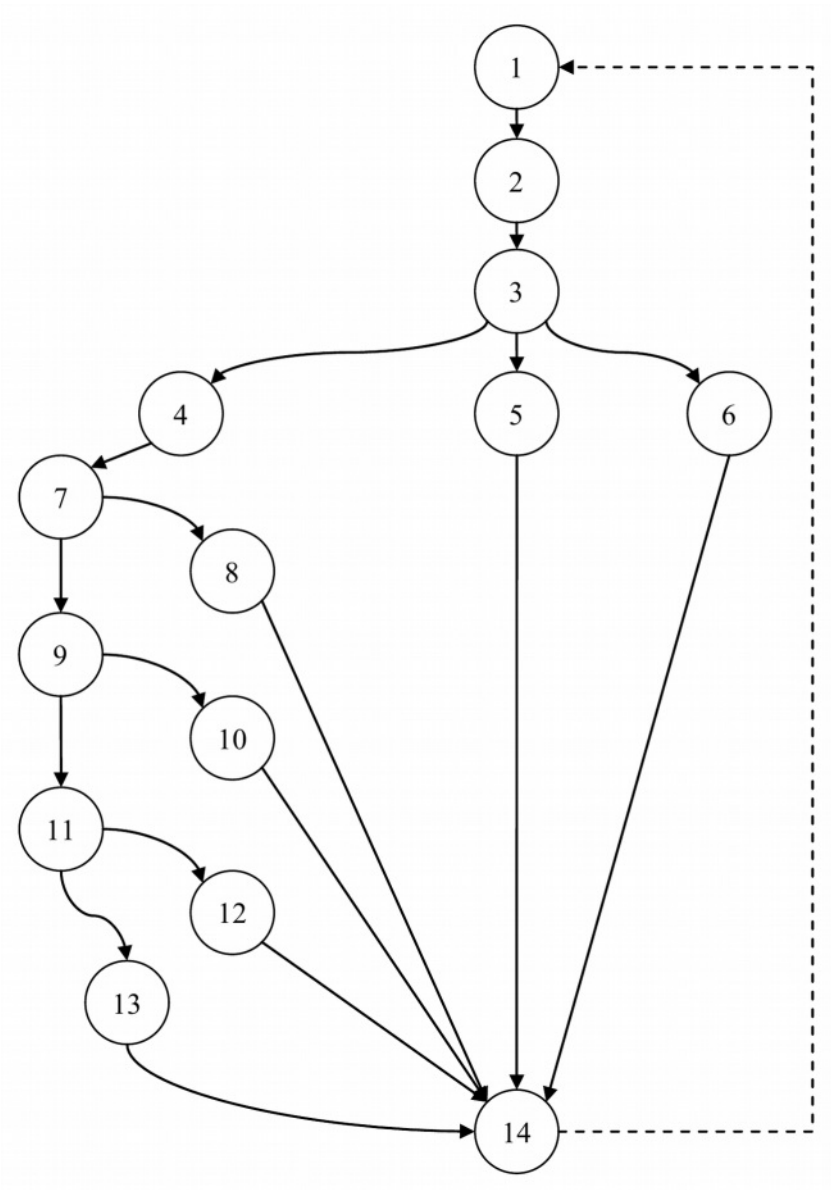
Возможно вам показалось, что в теоретическом материале информация представлена бессистемно, так как преподу было лень все расписывать. В общем-то так и есть.

4.6 Пример выполнения работы

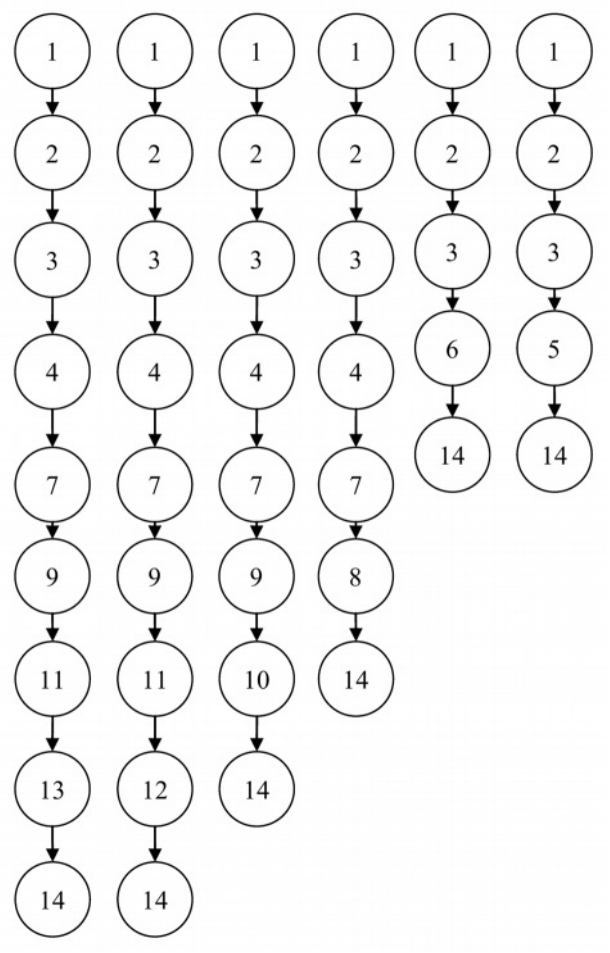
Блок-схема алгоритма



Граф потока данных



Линейно независимые маршруты и циклы



Расчет цикломатической сложности (метрика Маккейба)

$M(G) = E – N + P = 19 - 14 + 1 = 6$

Примеры матриц смежности и достижимости приводить не буду, думаю, разберетесь.

К методичке прилагается справка по профилированию в MATLAB, если вы выбрали программу на другом языке, то воспользуйтесь средствами анализа производительности Visual Studio или VTune (в этом случае разбираться с профилированием будете сами, считайте это частью задания на «отлично»).

Пример вывода:

В результате выполнения данной лабораторной работы были изучены методы оценки сложности потока управления на основе метрики Маккейба.

Произведен расчет цикломатической сложности программы и построены матрицы смежности и достижимости.

Цикломатическое число Маккейба совпадает с числом линейно-независимых маршрутов и циклов, что свидетельствует о правильно выполненных расчетах.

Произведено профилирование кода и выявлены следующие горячие точки:
- *указать на фрагментов кода, где выявлены горячие точки программы.*

4.7 Контрольные вопросы

1) Приведите пример метрик сложности потока данных.

2) Что показывает цикломатическая сложность?

3) Что представляет собой граф потока управления?

4) Чем сильносвязный (плотный) граф отличается от разряженного?

5) Какие операторы влияют на цикломатическую сложность?

6) Что представляет собой профилирование?

7) Что представляет собой «горячая» точка?

8) Что представляет собой покрытие кода?

9) Приведите способы покрытия кода.

10) Что представляет собой тестирование ПО?

11) Что представляет собой «мертвый» код?

12) Назначение матрицы смежности?

13) Назначение матрицы достижимости?

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 28195-89. Оценка качества программных средств. Общие положения. – М.: ИПК Издательство стандартов, 2001. – 30 с.

2. ГОСТ Р ИСО/МЭК 25-10-2015. Системная и программная инженерия. Требования и оценка качества систем и программного обеспечения (SQuaRE).Модели качества систем и программных продуктов. – М.: Стандартинформ, 2015. – 29 с.

3. Метрика программного обеспечения [Электронный ресурс]: Материал из Википедии – свободной энциклопедии. – Электронные данные – Wikipedia, 2017. – URL: https://ru.wikipedia.org/wiki/Метрика_программного_обеспечения

4. Черников Б.В. Оценка качества программного обеспечения: Практикум: учебное пособие / Б.В. Черников, Б.Е. Поклонов / Под ред. Б.В. Черникова. – М.: ИД "ФОРУМ": ИНФРА-М, 2012. – 400 с.: ил.

5. Метрики кода и их практическая реализация [Электронный ресурс]. – Электронные данные – ООО СМ-Консалт (СМК), 2004-2016. – URL: http://cmcons.com/articles/CC_CQ/dev_metrics/mertics_part_1/

6. Значения метрик кода [Электронный ресурс]: документация для Visual Studio 2015. – Электронные данные – Microsoft, 2017. – URL: https://msdn.microsoft.com/ru-ru/library/bb385914.aspx

7. Профилирование (информатика) [Электронный ресурс]: Материал из Википедии – свободной энциклопедии. – Электронные данные – Wikipedia, 2017. – URL: https://ru.wikipedia.org/wiki/Профилирование_(информатика)

ПРИЛОЖЕНИЕ

Листинг программы

```
	var http = require("http"),
	https = require("https"),
	url = require("url"),
	querystring = require("querystring"),
	APIDOG_LONGPOLL_RESULT_CODE_OK = 0,
	APIDOG_LONGPOLL_RESULT_CODE_CAPTCHA = 1,
	APIDOG_LONGPOLL_RESULT_CODE_API_ERROR = 2,
	APIDOG_LONGPOLL_RESULT_CODE_SERVER_ISSUE = 3,
	APIDOG_LONGPOLL_RESULT_CODE_EMPTY_RESPONSE = 4,
	APIDOG_LONGPOLL_RESULT_CODE_NOT_JSON = 5,
	APIDOG_LONGPOLL_RESULT_CODE_FAILED = 6,
	APIDOG_LONGPOLL_RESULT_CODE_API_REQUEST_UNKNOWN = 7,
	LP_MODE_ATATCHS = 2,
	LP_MODE_EXTENDS = 8,
	LP_MODE_PTS = 32,
	LP_MODE_EXTRA_FRIENDS = 64,
	LP_MODE_RANDOM_ID = 128,
	LP_VERSION_NORMAL = 0,
	LP_VERSION_GROUP_NEGATIVE = 1,
	config = {
		version: LP_VERSION_GROUP_NEGATIVE,
		mode: LP_MODE_ATATCHS | LP_MODE_EXTENDS | LP_MODE_EXTRA_FRIENDS |
LP_MODE_RANDOM_ID
	},
	server = http.createServer(function(request, response) {
		var requestData = url.parse(request.url),
			GET = querystring.parse(requestData.query),
			userAccessToken = GET.userAccessToken,
			captchaId = GET.captchaSid,
			captchaKey = GET.captchaKey;

		if (!userAccessToken) {
			outputJSON(response,
[APIDOG_LONGPOLL_RESULT_CODE_API_REQUEST_UNKNOWN, null, {} ]);
		};
		response.writeHead(200, {
			"Content-Type": "application/json; charset=utf-8",
			"Access-Control-Allow-Origin": "https://apidog.ru",
			"Access-Control-Allow-Credentials": true,
			"Access-Control-Allow-Methods": "HEAD, OPTIONS, GET, POST",
			"Access-Control-Allow-Headers": "Content-Type, User-Agent, X-
Requested-With, If-Modified-Since, Cache-Control",
			"X-APIdog-Config": "v: " + config.version + "; mode: " + config.mode
		});
		if (!GET.ts && !GET.key && !GET.server) {
```

```
                        getLongPollServer(response, userAccessToken, {id: captchaId, key:
captchaKey});
                } else {
                        waitForLongPoll(response, {ts: GET.ts, key: GET.key, server: GET.server});
                };
        }).listen(4000);
/**
 * Запрос адреса LongPoll-сервера
 * @param  {Object} response         Объект для ответа
 * @param  {String} userAccessToken    Пользовательский токен
 * @param  {Object} captcha          Данные капчи
 * @param  {Number} n                Номер попытки
 */
function getLongPollServer(response, userAccessToken, captcha, n) {
        n = n || 0;

        var params = { v: 4.104, access_token: userAccessToken };

        if (captcha) {
                params.captcha_sid = captcha.id;
                params.captcha_key = captcha.key;
        };

        try {
                API("messages.getLongPollServer", params, function(data) {
                        if (!data) {
                                return n < 2 ? getLongPollServer(response, userAccessToken,
captcha, ++n) : response.end();
                        };

                        if (!data.response) {
                                if (data.error && data.error.error_code == 14) {
                                        return outputJSON(response,
[APIDOG_LONGPOLL_RESULT_CODE_CAPTCHA, null, {
                                                captchaId: data.error.captcha_sid,
                                                captchaImg: data.error.captcha_img
                                        }]);
                                };
                                return outputJSON(response,
[APIDOG_LONGPOLL_RESULT_CODE_API_ERROR, null, {source: data.error}]);
                        };
                        waitForLongPoll(response, data.response);
                }, response);
        } catch (e) {
                return outputJSON(response,
[APIDOG_LONGPOLL_RESULT_CODE_SERVER_ISSUE, null, {  }]);
        };
};
```

```
/**
 * Висячий процесс запроса LongPoll
 * @param  {Object} response Объект для ответа
 * @param  {Object} data     Объект с данными для запроса
 */
function waitForLongPoll(response, data) {
	var url = data.server.split("/"), host = url[0],
		path = "/" + url[1] + "?" + querystring.stringify({
			act: "a_check", wait: 15, mode: config.mode, key: data.key, ts: data.ts,
version: config.version
		});

	http.get({ host: host, port: 80, path: path }, function(result) {
		var json = new String();
		result.setEncoding("utf8");
		result.on("data", function(chunk) { json += chunk; });
		result.on("end", function() {
			if (!json) {
				return outputJSON(response,
[APIDOG_LONGPOLL_RESULT_CODE_EMPTY_RESPONSE, null, { }]);
			};
			try {
				json = JSON.parse(json);
			} catch (e) {
				return outputJSON(response,
[APIDOG_LONGPOLL_RESULT_CODE_NOT_JSON, null, { reason: e.toString() }]);
			};
			if (json.failed) {
				return outputJSON(response,
[APIDOG_LONGPOLL_RESULT_CODE_FAILED, null, { failed: json.failed }]);
			};
			outputJSON(response, [APIDOG_LONGPOLL_RESULT_CODE_OK, json,
{server: data.server, key: data.key, ts: data.ts || json.ts}]);
		});
	}).on("error", function(e) {
		return outputJSON(response,
[APIDOG_LONGPOLL_RESULT_CODE_API_REQUEST_UNKNOWN, null, {}]);
	}).setTimeout(25000, function() {
		return outputJSON(response, [APIDOG_LONGPOLL_RESULT_CODE_FAILED, null,
{ }]);
	});
};
/**
 * Вывод ответа в JSON
 * @param  {Object} response Объект ответа
 * @param  {Object} data     Данные для вывода
 */
function outputJSON(response, data) {
```

```
	data.push({v: 2});
	response.write(JSON.stringify(data));
	response.end();
};
/**
 * Запрос к API
 * @param {String}   method   Метод
 * @param {Object}   params   Параметры
 * @param {Function} callback Обработчик
 * @param {Object}   response Объект ответа
 */
function API(method, params, callback, response) {
	https.get({ host: "api.vk.com", port: 443, path: "/method/" + method + "?" +
querystring.stringify(params) }, function(res) {
		var apiResponse = new String();
		res.setEncoding("utf8");
		res.on("data", function(chunk) { apiResponse += chunk; });
		res.on("end", function() {
			try {
				callback(JSON.parse(apiResponse));
			} catch (e) {
				callback(null);
			};
		});
	}).on("error", function(e) {
		return outputJSON(response,
[APIDOG_LONGPOLL_RESULT_CODE_SERVER_ISSUE, null, {}]);
	}).setTimeout(7500, function() {
		outputJSON(response, [APIDOG_LONGPOLL_RESULT_CODE_SERVER_ISSUE, null,
{}]);
	});
};
```