МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Курский государственный университет»

кафедра программного обеспечения и администрирования информационных систем

-

Отчёт
по лабораторной работе № 1

«Обзор общих подходов к тестированию, верификация требований к разрабатываемой системе»

по дисциплине

«Верификация программного обеспечения»

Выполнил: студент(ка) группы 413
Мусонда Салиму

Проверил: к.т.н., доцент
кафедры ПОиАИС
Макаров К.С.

Курск, 2020

Цель работы: изучение общих подходов к тестированию, верификация требований к разрабатываемой системе на основе анализа спецификации.

Задачи:

1) Изучить основные понятия дисциплины «Верификация ПО»;

2) Проанализировать спецификацию к (учебному примеру) системе «Калькулятор», представленную в приложении 1 к Лабораторной работе №1. Найти ошибки и представить их согласно образцу («Шаблон отчёта о проблеме.docx») в качестве экспериментальных результатов работы.

Основные теоретические положения

Верификация – это процесс определения, выполняют ли программные средства и их компоненты требования, наложенные на них в последовательных этапах жизненного цикла разрабатываемой программной системы.

Основная **цель верификации** состоит в подтверждении того, что программное обеспечение соответствует требованиям. Дополнительной целью является выявление и регистрация дефектов и ошибок, которые внесены во время разработки или модификации программы.

Верификация является неотъемлемой частью работ при коллективной разработке программных систем. При этом в задачи верификации включается контроль результатов одних разработчиков при передаче их в качестве исходных данных другим разработчикам.

Для повышения эффективности использования человеческих ресурсов при разработке, верификация должна быть тесно интегрирована с процессами проектирования, разработки и сопровождения программной системы.

Заранее разграничим понятия верификации и отладки. Оба этих процесса направлены на уменьшение ошибок в конечном программном продукте, однако **отладка** – процесс, направленный на локализацию и устранение ошибок в системе, а **верификация** – процесс, направленный на демонстрацию наличия ошибок и условий их возникновения.

Кроме того, верификация, в отличие от отладки – контролируемый и управляемый процесс. Верификация включает в себя анализ причин возникновения ошибок и последствий, которые вызовет их исправление, планирование процессов поиска ошибок и их исправления, оценку полученных результатов. Все это позволяет говорить о верификации, как о процессе обеспечения заранее заданного уровня качества создаваемой программной системы.

Модели жизненного цикла разработки программного обеспечения

Коллективная разработка, в отличие от индивидуальной, требует четкого планирования работ и их распределения во время создания программной системы. Один из способов организации работ состоит в разбиении процесса разработки на отдельные последовательные стадии, после полного прохождения которых получается конечный продукт или его часть. Такие стадии называют жизненным циклом разработки программной системы. Как правило, жизненный цикл начинается с формирования общего представления о разрабатываемой системе и их формализации в виде требований верхнего уровня. Завершается жизненный цикл разработки вводом системы в эксплуатацию. Однако, нужно понимать, что разработка – только один из процессов, связанных с программной системой, которая также имеет свой жизненный цикл. В отличие от жизненного цикла разработки системы, жизненный цикл самой системы заканчивается выводом ее из эксплуатации и прекращением ее использования.

Жизненный цикл программного обеспечения – совокупность итерационных процедур, связанных с последовательным изменением состояния программного обеспечения от формирования исходных требований к нему до окончания его эксплуатации конечным пользователем.

Водопадная (каскадная) модель (waterfall model) сейчас представляет скорее исторический интерес, так как в современных проектах практически неприменима. Она предполагает однократное выполнение каждой из фаз проекта, которые, в свою очередь, строго следуют друг за другом (рисунок 1). Очень упрощённо можно сказать, что в рамках этой модели в любой момент времени команде «видна» лишь предыдущая и следующая фаза. В реальной же разработке ПО приходится «видеть весь проект целиком» и возвращаться к предыдущим фазам, чтобы исправить недоработки или что-то уточнить.

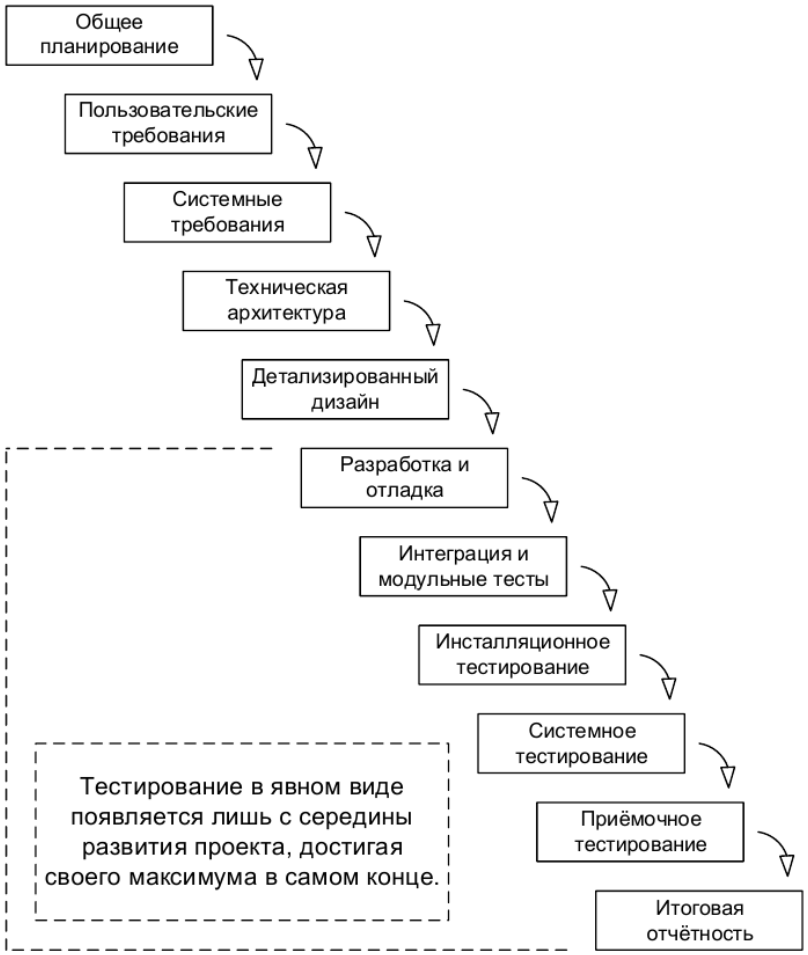

Рисунок 1 – Водопадная модель разработки ПО

К недостаткам водопадной модели принято относить тот факт, что участие пользователей ПО в ней либо не предусмотрено вообще, либо предусмотрено лишь косвенно на стадии однократного сбора требований. С точки зрения же тестирования эта модель плоха тем, что тестирование в явном виде появляется здесь лишь с середины развития проекта, достигая своего максимума в самом конце.

Тем не менее водопадная модель часто интуитивно применяется при выполнении относительно простых задач, а её недостатки послужили прекрасным отправным пунктом для создания новых моделей. Также эта модель в несколько усовершенствованном виде используется на крупных проектах, в которых требования очень стабильны и могут быть хорошо сформулированы в начале проекта (аэрокосмическая область, медицинское ПО и т.д.)

V-образная модель (V-model) является логическим развитием водопадной. Можно заметить (рисунок 2), что в общем случае как водопадная, так и v-образная модели жизненного цикла ПО могут содержать один и тот же набор стадий, но принципиальное отличие заключается в том, как эта информация используется в процессе реализации проекта.

Очень упрощённо можно сказать, что при использовании v-образной модели на каждой стадии «на спуске» нужно думать о том, что и как будет происходить на соответствующей стадии «на подъёме». Тестирование здесь

появляется уже на самых ранних стадиях развития проекта, что позволяет минимизировать риски, а также обнаружить и устранить множество потенциальных проблем до того, как они станут проблемами реальными.

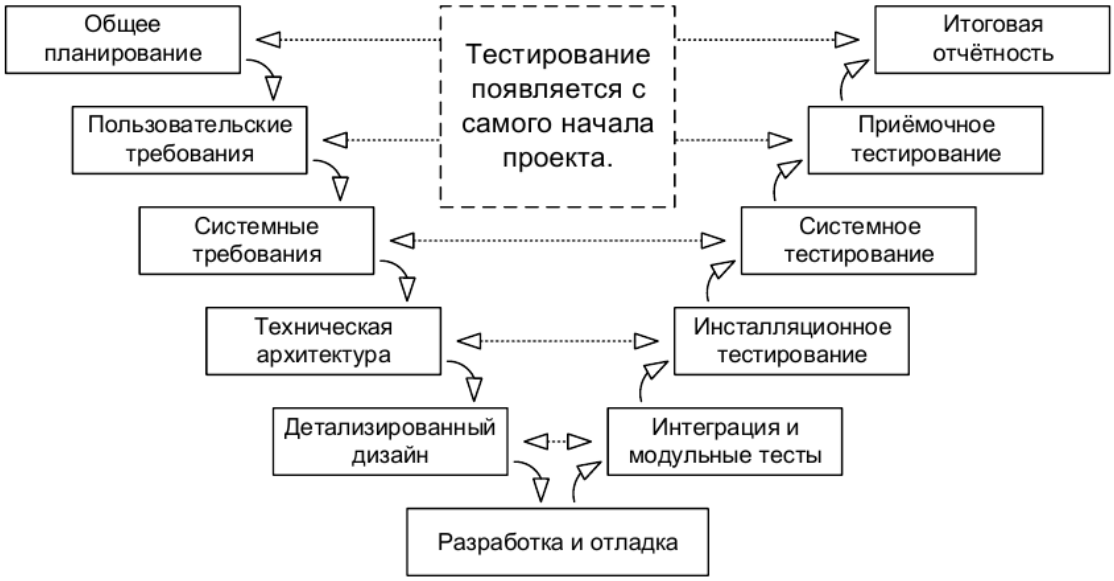

Рисунок 2 – V-образная модель разработки ПО

Итерационная инкрементальная модель (iterative model, incremental model) является фундаментальной основой современного подхода к разработке ПО. Как следует из названия модели, ей свойственна определённая двойственность (а ISTQB-глоссарий даже не приводит единого определения, разбивая его на отдельные части):

- с точки зрения жизненного цикла модель является итерационной, так как подразумевает многократное повторение одних и тех же стадий;
- с точки зрения развития продукта (приращения его полезных функций) модель является инкрементальной.

Ключевой особенностью данной модели является разбиение проекта на относительно небольшие промежутки (итерации), каждый из которых в общем случае может включать в себя все классические стадии, присущие водопадной и v-образной моделям (рисунок 3). Итогом итерации является приращение (инкремент) функциональности продукта, выраженное в промежуточном билде (build).

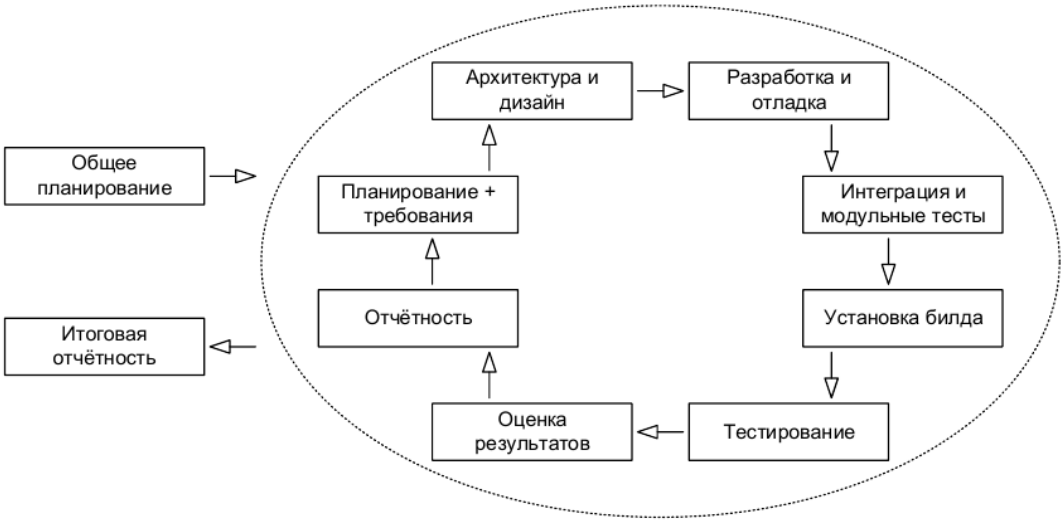

Рисунок 3 – Итерационная инкрементальная модель разработки ПО

Длина итераций может меняться в зависимости от множества факторов, однако сам принцип многократного повторения позволяет гарантировать, что и тестирование, и демонстрация продукта конечному заказчику (с получением обратной связи) будет активно применяться с самого начала и на протяжении всего времени разработки проекта.

Во многих случаях допускается распараллеливание отдельных стадий внутри итерации и активная доработка с целью устранения недостатков, обнаруженных на любой из (предыдущих) стадий.

Итерационная инкрементальная модель очень хорошо зарекомендовала себя на объёмных и сложных проектах, выполняемых большими командами на протяжении длительных сроков. Однако к основным недостаткам этой модели часто относят высокие накладные расходы, вызванные высокой «бюрократизированностью» и общей громоздкостью модели.

Спиральная модель (spiral model) представляет собой частный случай итерационной инкрементальной модели, в котором особое внимание уделяется управлению рисками, в особенности влияющими на организацию процесса разработки проекта и контрольные точки.

Схематично суть спиральной модели представлена на рисунке 4. Обратите внимание на то, что здесь явно выделены четыре ключевые фазы:

- проработка целей, альтернатив и ограничений;
- анализ рисков и прототипирование;
- разработка (промежуточной версии) продукта;
- планирование следующего цикла.

С точки зрения тестирования и управления качеством повышенное внимание рискам является ощутимым преимуществом при использовании спиральной модели для разработки концептуальных проектов, в которых требования естественным образом являются сложными и нестабильными (могут многократно меняться по ходу выполнения проекта).

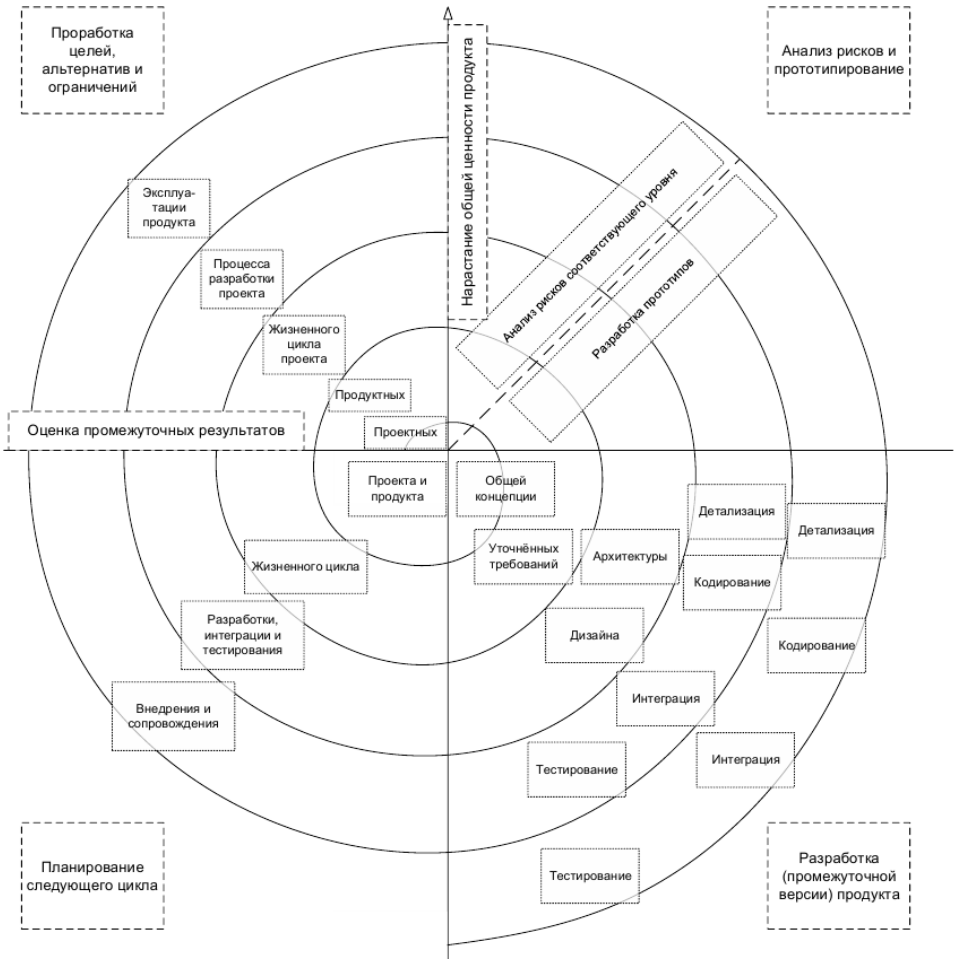

Рисунок 4 – Спиральная модель разработки ПО

Гибкая модель (agile model) представляет собой совокупность различных подходов к разработке ПО и базируется на так называемом «agile-манифесте»:

- люди и взаимодействие важнее процессов и инструментов;
- работающий продукт важнее исчерпывающей документации;
- сотрудничество с заказчиком важнее согласования условий контракта;
- готовность к изменениям важнее следования первоначальному плану.

Как несложно догадаться, положенные в основу гибкой модели подходы являются логическим развитием и продолжением всего того, что было за десятилетия создано и опробовано в водопадной, v-образной, итерационной инкрементальной, спиральной и иных моделях. Причём здесь впервые был достигнут ощутимый результат в снижении бюрократической составляющей и максимальной адаптации процесса разработки ПО к мгновенным изменениям рынка и требований заказчика.

Очень упрощённо (почти на грани допустимого) можно сказать, что гибкая модель представляет собой облегчённую с точки зрения документации смесь итерационной инкрементальной и спиральной моделей (рисунки 5 и 6); при этом следует помнить об «agile-манифесте» и всех вытекающих из него преимуществах и недостатках.

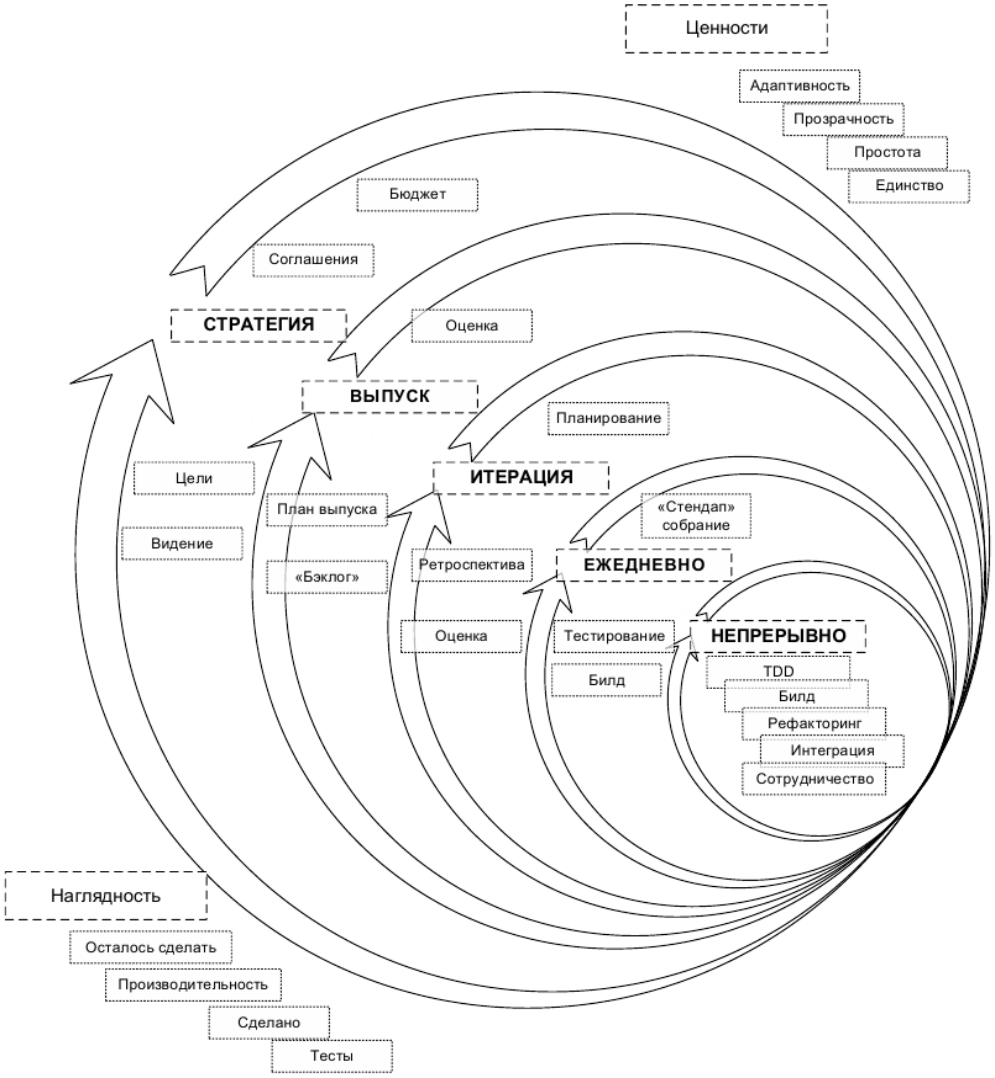

Рисунок 5 – Суть гибкой модели разработки ПО

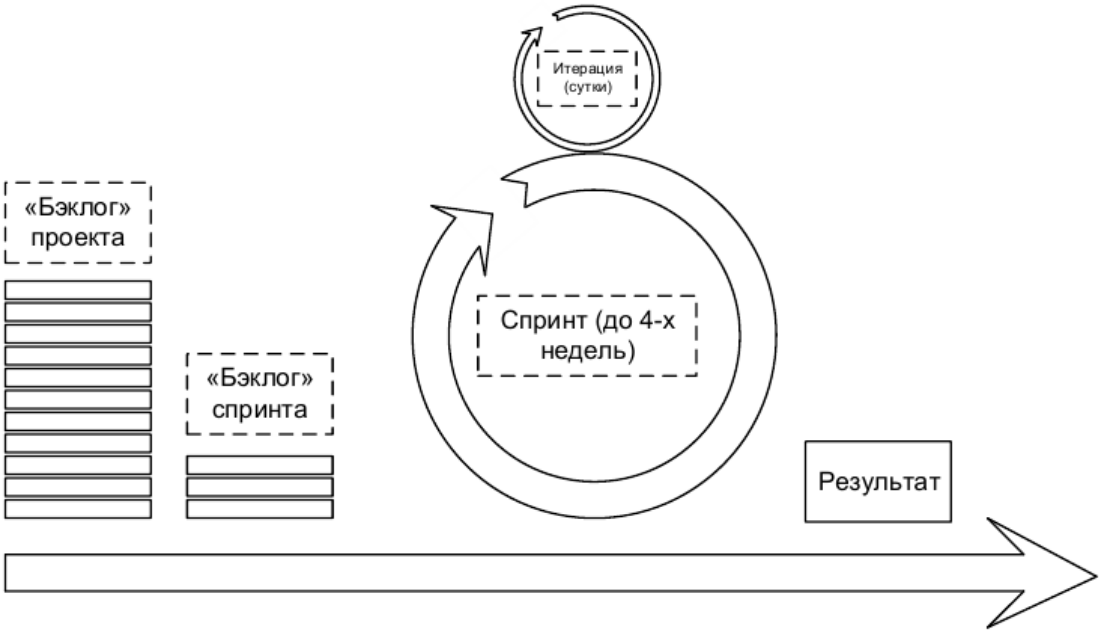

Рисунок 6 – Итерационный подход в рамках гибкой модели и scrum

Главным недостатком гибкой модели считается сложность её применения к крупным проектам, а также частое ошибочное внедрение её подходов, вызванное недопониманием фундаментальных принципов модели.

Тем не менее можно утверждать, что всё больше и больше проектов начинают использовать гибкую модель разработки.

Кратко можно выразить суть моделей разработки ПО таблицей 1.

Таблица 1 – Сравнение моделей разработки ПО

Модель	**Преимущества**	**Недостатки**	**Тестирование**
Водопадная	• у каждой стадии есть чёткий проверяемый результат; • в каждый момент времени команда выполняет один вид работы; • хорошо работает для небольших задач.	полная неспособность адаптировать проект к изменениям в требованиях; крайне позднее создание работающего продукта.	с середины проекта
V-образная	у каждой стадии есть чёткий проверяемый результат; внимание тестированию уделяется с первой же стадии; хорошо работает для проектов со стабильными требованиями.	• недостаточная гибкость и адаптируемость; • отсутствует раннее прототипирование; • сложность устранения проблем, пропущенных на ранних стадиях развития проекта.	на переходах между стадиями
Итерационная инкрементальная	• достаточно раннее прототипирование; • простота управления итерациями; • декомпозиция проекта на управляемые итерации.	• недостаточная гибкость внутри итераций; • сложность устранения проблем, пропущенных на ранних стадиях развития проекта.	• в определённые моменты итераций; • повторное тестирование (после доработки) уже проверенного ранее.
Спиральная	• глубокий анализ рисков; • подходит для крупных проектов; • достаточно раннее	высокие накладные расходы; сложность применения для небольших проектов;	

	прототипирование.	высокая зависимость успеха от качества анализа рисков.	
Гибкая	• максимальное вовлечение заказчика; • много работы с требованиями; • тесная интеграция тестирования и разработки; • минимизация документации.	сложность реализации для больших проектов; сложность построения стабильных процессов.	В определённые моменты итераций и в **любой необходимый момент**.

В приведенном выше описании различных моделей жизненного цикла по сути затрагивался только один процесс – процесс разработки системы. На самом деле в любой модели жизненного цикла можно увидеть четыре вида процессов:

1) основной процесс разработки;
2) процесс верификации;
3) процесс управления разработкой;
4) вспомогательные (поддерживающие) процессы.

Процесс верификации – процесс, направленный на проверку корректности разрабатываемой системы и определения степени ее соответствия требованиям заказчика. Подробному рассмотрению этого процесса и посвящен данный учебный курс.

Процесс управления разработкой – отдельная дисциплина, на содержание которой непосредственно влияет тип жизненного цикла основного процесса разработки. По сути, чем короче один этап жизненного цикла, тем активнее управление, и тем больше задач стоит на менеджере проекта. При классических схемах достаточно просто построить иерархическую пирамиду подчиненности, у которой каждый нижестоящий менеджер отвечает за разработку определенной части системы. В XP-подходе нет жесткого разделения системы на части и менеджер должен охватывать все истории. При этом процесс управления активен на протяжении всего жизненного цикла основного процесса разработки.

Вспомогательные (поддерживающие) процессы – процессы, обеспечивающие своевременное предоставление того, что может понадобиться разработчику или конечному пользователю. Сюда входит подготовка пользовательской документации, подготовка приемо-сдаточных тестов, управление конфигурациями и изменениями, взаимодействие с заказчиком и т.д. Вообще говоря, вспомогательные процессы могут существовать в течение всего жизненного цикла разработки, а могут быть

своеобразными стыкующими звеньями между процессом разработки и процессом эксплуатации.

Ролевой состав коллектива разработчиков, взаимодействие между ролями в различных технологических процессах

Когда проектная команда включает более двух человек неизбежно встает вопрос о распределении ролей, прав и ответственности в команде. Конкретный набор ролей определяется многими факторами – количеством участников разработки и их личными предпочтениями, принятой методологией разработки, особенностями проекта и другими факторами. Практически в любом коллективе разработчиков можно выделить перечисленные ниже роли. Некоторые из них могут вовсе отсутствовать, при этом отдельные люди могут выполнять сразу несколько ролей, однако общий состав меняется мало.

Заказчик (заявитель). Эта роль принадлежит представителю организации, заказавшей разрабатываемую систему. Обычно заявитель ограничен в своем взаимодействии и общается только с менеджерами проекта и специалистом по сертификации или внедрению. Обычно заказчик имеет право изменять требования к продукту (только во взаимодействии с менеджерами), читать проектную и сертификационную документацию, затрагивающую нетехнические особенности разрабатываемой системы.

Менеджер проекта. Эта роль обеспечивает коммуникационный канал между заказчиком и проектной группой. Менеджер продукта управляет ожиданиями заказчика, разрабатывает и поддерживает бизнес-контекст проекта. Его работа не связана напрямую с продажей продукта, он сфокусирован на продукте, его задача – определить и обеспечить требования заказчика. Менеджер проекта имеет право изменять требования к продукту и финальную документацию на продукт.

Менеджер программы. Эта роль управляет коммуникациями и взаимоотношениями в проектной группе, является в некотором роде координатором, разрабатывает функциональные спецификации и управляет ими, ведет график проекта и отчитывается по состоянию проекта, инициирует принятие критичных для хода проекта решений.

Менеджер программы имеет право изменять функциональные спецификации верхнего уровня, план-график проекта, распределение ресурсов по задачам. Часто на практике роль менеджера проекта и менеджера программы выполняет один человек.

Разработчик. Разработчик принимает технические решения, которые могут быть реализованы и использованы, создает продукт, удовлетворяющий спецификациям и ожиданиям заказчика, консультирует другие роли в ходе проекта. Он участвует в обзорах, реализует возможности продукта, участвует

в создании функциональных спецификаций, отслеживает и исправляет ошибки за приемлемое время. В контексте конкретного проекта роль разработчика может подразумевать, например, инсталляцию программного обеспечения, настройку продукта или услуги. Разработчик имеет доступ ко всей проектной документации, включая документацию по тестированию, имеет право на изменение программного кода системы в рамках своих служебных обязанностей.

Специалист по тестированию. Специалист по тестированию определяет стратегию тестирования, тест-требования и тест-планы для каждой из фаз проекта, выполняет тестирование системы, собирает и анализирует отчеты о прохождении тестирования. Тест-требования должны покрывать системные требования, функциональные спецификации, требования к надежности и нагрузочной способности, пользовательские интерфейсы и собственно программный код. В реальности роль специалиста по тестированию часто разбивается на две – разработчика тестов и тестировщика. Тестировщик выполняет все работы по выполнению тестов и сбору информации, разработчик тестов – всю остальные работы.

Специалист по контролю качества. Эта роль принадлежит члену проектной группы, который осуществляет взаимодействие с разработчиком, менеджером программы и специалистами по безопасности и сертификации с целью отслеживания целостной картины качества продукта, его соответствия стандартам и спецификациям, предусмотренным проектной документацией. Следует различать специалиста по тестированию и специалиста по контролю качества. Последний не является членом технического персонала проекта, ответственным за детали и технику работы. Контроль качества подразумевает в первую очередь контроль самих процессов разработки и проверку их соответствия определенным в стандартах качества критериям.

Специалист по сертификации. При разработке систем, к надежности которых предъявляются повышенные требования, перед вводом системы в эксплуатацию требуется подтверждение со стороны уполномоченного органа (обычно государственного) соответствия ее эксплуатационных характеристик заданным критериям. Такое соответствие определяется в ходе сертификации системы. Специалист по сертификации может либо быть представителем сертифицирующих органов, включенным в состав коллектива разработчиков, либо наоборот – представлять интересы разработчиков в сертифицирующем органе. Специалист по сертификации приводит документацию на программную систему в соответствие требованиям сертифицирующего органа, либо участвует в процессе создания документации с учетом этим требованиям. Также специалист по сертификации ответственен за все взаимодействие между коллективом разработчиков и сертифицирующим органом. Важной особенностью роли является независимость специалиста от проектной группы на всех этапах создания продукта. Взаимодействие специалиста с членами проектной группы ограничивается менеджерами по проекту и по программе.

Специалист по внедрению и сопровождению. Участвует в анализе особенностей площадки заказчика, на которой планируется проводить внедрение разрабатываемой системы, выполняет весь спектр работ по установке и настройке системы, проводит обучение пользователей.

Специалист по безопасности. Данный специалист ответственен за весь спектр вопросов безопасности создаваемого продукта. Его работа начинается с участия в написании требований к продукту и заканчивается финальной стадией сертификации продукта.

Инструктор. Эта роль отвечает за снижение затрат на дальнейшее сопровождение продукта, обеспечение максимальной эффективности работы пользователя. Важно, что речь идет о производительности пользователя, а не системы. Для обеспечения оптимальной продуктивности инструктор собирает статистику по производительности пользователей и создает решения для повышения производительности, в том числе с использованием различных аудиовизуальных средств. Инструктор принимает участие во всех обсуждениях пользовательского интерфейса и архитектуры продукта.

Технический писатель. Лицо, осуществляющее эту роль, несет обязанности по подготовке документации к разработанному продукту, финального описания функциональных возможностей. Так же он участвует в написании сопроводительных документов (системы помощи, руководство пользователя).

Задачи и цели процесса верификации

Цель верификации – доказательство того, что результат разработки соответствует предъявленным к нему требованиям. Обычно процесс верификации проводится сверху вниз, начиная от общих требований, заданных в техническом задании и/или спецификации на всю информационную систему до детальных требований на программные модули и их взаимодействие. В состав задач процесса верификации входит последовательная проверка того, что в программной системе:

- общие требования к информационной системе, корректно переработаны в спецификацию требований высокого уровня к комплексу программ, удовлетворяющих исходным системным требованиям;
- требования высокого уровня корректно переработаны в архитектуру ПО и в спецификации требований к функциональным компонентам низкого уровня, которые удовлетворяют требованиям высокого уровня;
- архитектура ПО и требования к компонентам низкого уровня корректно переработаны в удовлетворяющие им исходные тексты программных и информационных модулей;
- исходные тексты программ и соответствующий им исполняемый код не содержат ошибок.

Кроме того, верификации на соответствие спецификации требований на конкретный проект программного средства подлежат требования к технологическому обеспечению жизненного цикла ПО, а также требования к эксплуатационной и технологической документации.

Определим некоторые понятия и определения, связанные с процессом тестирования, как составной части верификации.

Тестирование – процесс выполнения программы с целью обнаружения ошибки.

Тестовые данные – входы, которые используются для проверки системы.

Тестовая ситуация (test case) – входы для проверки системы и предполагаемые выходы в зависимости от входов, если система работает в соответствии с ее спецификацией требований.

Хорошая тестовая ситуация – та ситуация, которая обладает большой вероятностью обнаружения пока еще необнаруженной ошибки.

Удачный тест – тест, который обнаруживает пока еще необнаруженную ошибку.

Ошибка – действие программиста на этапе разработки, которое приводит к тому, что в программном обеспечении содержится внутренний дефект, который в процессе работы программы может привести к неправильному результату.

Отказ – непредсказуемое поведение системы, приводящее к неожидаемому результату, которое могло быть вызвано дефектами, содержащимся в ней.

Тестирование, верификация и валидация – различия в понятиях

Несмотря на кажущуюся схожесть, термины «тестирование», «верификация» и «валидация» означают разные уровни проверки корректности работы программной системы. Определим данные понятия (**Error! Reference source not found.**7).

Тестирование программного обеспечения – вид деятельности в процессе разработки, связанный с выполнением процедур, направленных на обнаружение (доказательство наличия) ошибок (несоответствий, неполноты, двусмысленностей и т.д.) в текущем определении разрабатываемой программной системы. Процесс тестирования относится в первую очередь к проверке корректности программной реализации системы, соответствия реализации требованиям, то есть тестирование – это управляемое выполнение программы с целью обнаружения несоответствий ее поведения и требований.

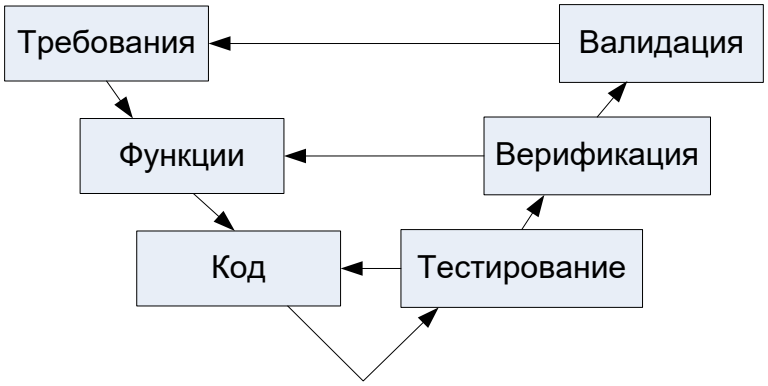

Рисунок 7 – Тестирование, верификация и валидация

Верификация программного обеспечения – процесс, целью которого является достижение гарантии того, что верифицируемый объект (требования или программный код) соответствует требованиям, реализован без непредусмотренных функций и удовлетворяет проектным спецификациям и стандартам. Процесс верификации включает в себя инспекции, тестирование кода, анализ результатов тестирования, формирование и анализ отчетов о проблемах. Таким образом, принято считать, что процесс тестирования является составной частью процесса верификации.

Валидация программной системы – процесс, целью которого является доказательство того, что в результате разработки системы мы достигли тех целей, которые планировали достичь благодаря ее использованию. Иными словами, валидация – это проверка соответствия системы ожиданиям заказчика. Вопросы, связанные с валидацией выходят за рамки данного учебного курса.

Если посмотреть на эти три процесса с точки зрения вопроса, на который они дают ответ, то тестирование отвечает на вопрос «Как это сделано?» или «Соответствует ли поведение разработанной программы требованиям?», верификация – «Что сделано?» или «Соответствует ли разработанная система требованиям?», а валидация – «Сделано ли то, что нужно?» или «Соответствует ли разработанная система ожиданиям заказчика?».

Типы процессов тестирования и верификации

Модульное тестирование. Модульному тестированию подвергаются небольшие модули (процедуры, классы и т.п.). При тестировании относительного небольшого модуля размером 100-1000 строк есть возможность проверить, если не все, то, по крайней мере, многие логические ветви в реализации, разные пути в графе зависимости данных, граничные значения параметров. В соответствии с этим строятся критерии тестового покрытия (покрыты все операторы, все логические ветви, все граничные точки и т.п.). Модульное тестирование обычно выполняется для каждого независимого программного модуля и является, пожалуй, наиболее

распространенным видом тестирования, особенно для систем малых и средних размеров.

Интеграционное тестирование. Проверка корректности всех модулей, к сожалению, не гарантирует корректности функционирования системы модулей. В литературе иногда рассматривается «классическая» модель неправильной организации тестирования системы модулей, часто называемая методом «большого скачка». Суть метода состоит в том, чтобы сначала оттестировать каждый модуль в отдельности, потом объединить их в систему и протестировать систему целиком. Для крупных систем это нереально. При таком подходе будет потрачено очень много времени на локализацию ошибок, а качество тестирования останется невысоким. Альтернатива «большому скачку» – интеграционное тестирование, когда система строится поэтапно, группы модулей добавляются постепенно.

Интеграционное тестирование. Проверка корректности всех модулей, к сожалению, не гарантирует корректности функционирования системы модулей. В литературе иногда рассматривается «классическая» модель неправильной организации тестирования системы модулей, часто называемая методом «большого скачка». Суть метода состоит в том, чтобы сначала оттестировать каждый модуль в отдельности, потом объединить их в систему и протестировать систему целиком. Для крупных систем это нереально. При таком подходе будет потрачено очень много времени на локализацию ошибок, а качество тестирования останется невысоким. Альтернатива «большому скачку» – интеграционное тестирование, когда система строится поэтапно, группы модулей добавляются постепенно.

Системное тестирование. Полностью реализованный программный продукт подвергается системному тестированию. На данном этапе тестировщика интересует не корректность реализации отдельных процедур и методов, а вся программа в целом, как ее видит конечный пользователь. Основой для тестов служат общие требования к программе, включая не только корректность реализации функций, но и производительность, время отклика, устойчивость к сбоям, атакам, ошибкам пользователя и т.д. Для системного и компонентного тестирования используются специфические виды критериев тестового покрытия (например, покрыты ли все типовые сценарии работы, все сценарии с нештатными ситуациями, попарные композиции сценариев и проч.).

Нагрузочное тестирование. Нагрузочное тестирование позволяет не только получать прогнозируемые данные о производительности системы под нагрузкой, которая ориентирована на принятие архитектурных решений, но и предоставляет рабочую информацию службам технической поддержки, а также менеджерам проектов и конфигурационным менеджерам, которые отвечают за создания наиболее продуктивных конфигураций оборудования и ПО. Нагрузочное тестирование позволяет команде разработки, принимать более обоснованные решения, направленные на выработку оптимальных архитектурных композиций. Заказчик со своей стороны, получает

возможность проводить приёмо-сдаточные испытания в условиях, приближенных к реальным.

Формальные инспекции. Формальная инспекция является одним из способов верификации документов и программного кода, создаваемых в процессе разработки программного обеспечения. В ходе формальной инспекции группой специалистов осуществляется независимая проверка соответствия инспектируемых документов исходным документам. Независимость проверки обеспечивается тем, что она осуществляется инспекторами, не участвовавшими в разработке инспектируемого документа.

Спецификация программного обеспечения

Разработка любого программного обеспечения начинается с анализа требований к будущему программному продукту. В результате анализа получают спецификации разрабатываемого программного обеспечения, выполняют декомпозицию и содержательную постановку решаемых задач, уточняют их взаимодействия и их эксплуатационные ограничения. В целом в процессе определения спецификаций строят общую модель предметной области как некоторой части реального мира, с которой будет тем или иным способом взаимодействовать разрабатываемое программное обеспечение, и конкретизируют его основные функции.

Спецификации представляют собой полное и точное описание функций и ограничений разрабатываемого программного обеспечения. При этом одна часть спецификаций (функциональные) описывает функции разрабатываемого программного обеспечения, а другая часть (эксплуатационные) определяет требования к техническим средствам надежности информационной безопасности и т. д.

Определение отражает главные требования к спецификациям. Применительно к функциональным спецификациям подразумевается, что:

- требование полноты означает, что спецификации должны содержать всю существенную информацию, где ничего важного не было бы упущено и отсутствует несущественная информация, например детали реализации, чтобы не препятствовать разработчику в выборе наиболее эффективных решений;

- требование точности означает, что спецификации должны однозначно восприниматься как заказчиком, так и разработчиком;

Последнее требование выполнить достаточно сложно в силу того, что естественный язык для описания спецификаций не подходит: даже подробные спецификации на естественном языке не обеспечивают необходимой точности. Точные спецификации можно определить, только разработав некоторую формальную модель разрабатываемого программного обеспечения.

Формальные модели, используемые на этапе определения спецификаций можно разделить на две группы: модели, зависящие от подхода к разработке (структурного или объектно-ориентированного), и модели, не зависящие от него. Так диаграммы переходов состояний, которые демонстрируют особенности поведения разрабатываемого программного обеспечения при получении тех или иных сигналов извне, и математические модели предметной области используют при любом подходе к разработке.

Отчет о проблеме

Технологические цепочки и роли участников проекта, использующих отчеты о проблемах. Связь отчетов о проблемах с другими типами проектной документации.

Каждое несоответствие с требованиями, найденное тестировщиком, должно быть задокументировано в виде отчета о проблеме. Вероятность обнаружения и исправления ошибки, вызвавшей это несоответствие, зависит от того, насколько качественно она задокументирована. Отчеты о проблемах могут поступать не только от тестировщиков, но и от специалистов технической поддержки или пользователей, однако их общая цель – указать на наличие проблемы в системе, которая должна быть устранена. Если отчет составлен некорректно, разработчик не сможет устранить проблему, поэтому можно считать этот отчет одним из самых важных документов в цепочке тестовой документации.

Главное, что должно быть включено в отчет об ошибке – это:

- Способ воспроизведения проблемы. Для того, чтобы разработчик смог устранить проблему, он должен разобраться в ее причинах, самостоятельно воспроизведя ее (и, возможно, не один раз). Один из самых тяжелых случаев в процессе разработки возникает при невоспроизводимой проблеме, т.е. проблеме, у которой точно не известен способ ее вызывать. Нахождение такого способа – одна из самых нетривиальных задач в работе тестировщика.

- Анализ проблемы с кратким ее описанием. Лучше всего приводить описание в тех же терминах, в которых составлены требования на часть системы, где обнаружена проблема. В этом случае минимизируется вероятность недопонимания сути проблемы.

Любой отчет о проблеме должен быть составлен немедленно после ее обнаружения. Если отчет будет составлен спустя значительное время, повышается вероятность того, что в него не попадет какая-либо важная информация, которая поможет устранить причину проблемы в кратчайшие сроки.

Структура отчетов о проблемах, их трассировка на программный код и документацию

Структура отчёта о проблеме в целом мало различается в различных проектах, изменения обычно касаются только порядка и имен следования полей. Некоторые поля могут отражать специфику данного конкретного проекта, однако обычно эти поля следующие:

- Объект, в котором найдена проблема. Здесь помещается максимально полная информация – для документации это название документа, раздел, автор, версия. Для исходных текстов это имя модуля, имя функции/метода или номера строк, версия.

- Выпуск и версия системы. Определяет место, откуда был взят объект с обнаруженной проблемой. Обычно требуется отдельная

идентификация версии системы (а не только версии исходных текстов), поскольку может возникнуть путаница с повторно выявленными проблемами. В этом случае, если проблема уже была когда-то выявлена разработчиком и потом вновь появилась из-за того, что в систему попала не самая последняя версия программного модуля, разработчик может решить, что ему пришел старый отчет и проблемы на самом деле не существует.

- Тип отчёта:
  - Ошибка кодирования – код не соответствует требованиям.
  - Ошибка проектирования – тестировщик не согласен с проектной документацией.
  - Предложение – у тестировщика возникла идея, как можно усовершенствовать код.
  - Расхождение с документацией – поведение ПО не соответствует руководству пользователя или проектной документации либо вообще нигде не описано. При этом у тестировщика нет оснований объявлять, где именно находится ошибка.
  - Взаимодействие с аппаратурой – неверная диагностика плохого состояния устройства, ошибка в интерфейсе с устройством.
  - Вопрос – тестировщик не уверен, что это проблема, и ему требуется дополнительная информация.
- Степень важности. Строгого критерия определения степени важности не существует, и обычно это поле кодируют от 1 (незначительно) до 10 (фатально). Однако способов обоснованной оценки нет – очень сложно определить, насколько фатальной может оказаться, например, опечатка в один символ в руководстве пользователя.
- Суть проблемы. Краткое (не более 2 строчек) определение проблемы. Даже если две проблемы очень похожи, их описания должны различаться.
- Можно ли воспроизвести проблемную ситуацию? Ответ = Да, Нет, Не всегда. Последнее – если проблема носит нерегулярный характер. Нужно

описывать, когда она проявляется, а когда – нет (например, если не вовремя нажать не ту клавишу).

Подробное описание проблемы и способ её воспроизведения. При этом нужны подробности – и в описании условий воспроизведения, и в описании причины объявления получившейся реакции ошибкой.

Экспериментальные результаты

Отчёт о проблеме

Порядковый номер отчёта:1

Автор отчёта: __Мусонда Салиму_____ Дата создания отчёта: _22._ _09._ _2020_

Документы/разделы, связанные с проблемой:

4

…….

Идентификация объекта/процесса, где проявляется проблема:

Требования к программе

……

Определение проблемы:

Отсутствие пункта 3.2 в документации

……

Автор решения: _____ Дата формирования решения __ __ __

Принятое решение: (возможно, ссылки на изменяемые компоненты/запросы на изменения)

……

Результаты анализа, определяющие, на содержание каких компонент влияет решение:

……

План проверок, восстанавливающих текущее состояние документов разработки

……

Оценка принятого решения автором отчёта о проблеме: _

……

(0 = полностью согласен

1 = не все аспекты проблемы учтены/разрешены

2 = основная часть проблемы осталась неразрешённой

3 = решение не адекватно проблеме – не устраняет её)

Отчёт о проблеме

Порядковый номер отчёта:2

Автор отчёта: __Мусонда Салиму_____ Дата создания отчёта: _22._ _09._ _2020_

Документы/разделы, связанные с проблемой:

4

…….

Идентификация объекта/процесса, где проявляется проблема:

Требования к программе

……

Определение проблемы:

Отсутствие пункта 3.2.1 в документации

……

Автор решения: _____ Дата формирования решения __ __ __

Принятое решение: (возможно, ссылки на изменяемые компоненты/запросы на изменения)

……

Результаты анализа, определяющие, на содержание каких компонент влияет решение:

……

План проверок, восстанавливающих текущее состояние документов разработки

……

Оценка принятого решения автором отчёта о проблеме: _

……

(0 = полностью согласен

1 = не все аспекты проблемы учтены/разрешены

2 = основная часть проблемы осталась неразрешённой

3 = решение не адекватно проблеме – не устраняет её)

Отчёт о проблеме

Порядковый номер отчёта:3

Автор отчёта: __Мусонда Салиму_____ Дата создания отчёта: _22._ _09._ _2020_

Документы/разделы, связанные с проблемой:

3

…….

Идентификация объекта/процесса, где проявляется проблема:

Метод проверки правильности расстановки скобок
……

Определение проблемы:

Алгоритм не обеспечивает выполнение задачи

……

Автор решения: _____ Дата формирования решения __ __ __

Принятое решение: (возможно, ссылки на изменяемые компоненты/запросы на изменения)

……

Результаты анализа, определяющие, на содержание каких компонент влияет решение:

……

План проверок, восстанавливающих текущее состояние документов разработки

……

Оценка принятого решения автором отчёта о проблеме: _

……

(0 = полностью согласен

1 = не все аспекты проблемы учтены/разрешены

2 = основная часть проблемы осталась неразрешённой

3 = решение не адекватно проблеме – не устраняет её)

Отчёт о проблеме

Порядковый номер отчёта:4

Автор отчёта: __Мусонда Салиму_____ Дата создания отчёта: _22._ _09._ _2020_

Документы/разделы, связанные с проблемой:

4

…….

Идентификация объекта/процесса, где проявляется проблема:

Требования к программе

……

Определение проблемы:

Отсутствие возможности непосредственной записи значения в память

……

Автор решения: _____ Дата формирования решения __ __ __

Принятое решение: (возможно, ссылки на изменяемые компоненты/запросы на изменения)

……

Результаты анализа, определяющие, на содержание каких компонент влияет решение:

……

План проверок, восстанавливающих текущее состояние документов разработки

……

Оценка принятого решения автором отчёта о проблеме: _

……

(0 = полностью согласен

1 = не все аспекты проблемы учтены/разрешены

2 = основная часть проблемы осталась неразрешённой

3 = решение не адекватно проблеме – не устраняет её)

Отчёт о проблеме

Порядковый номер отчёта:5

Автор отчёта: __Мусонда Салиму_____ Дата создания отчёта: _22._ _09._ _2020_

Документы/разделы, связанные с проблемой:

4

…….

Идентификация объекта/процесса, где проявляется проблема:

Дополнительные требования к входному выражению
……

Определение проблемы:

Различие максимальной длины символов, указанной в разделах 4.3.7 и 2.2.3

……

Автор решения: _____ Дата формирования решения __ __ __

Принятое решение: (возможно, ссылки на изменяемые компоненты/запросы на изменения)

……

Результаты анализа, определяющие, на содержание каких компонент влияет решение:

……

План проверок, восстанавливающих текущее состояние документов разработки

……

Оценка принятого решения автором отчёта о проблеме: _

……

(0 = полностью согласен

1 = не все аспекты проблемы учтены/разрешены

2 = основная часть проблемы осталась неразрешённой

3 = решение не адекватно проблеме – не устраняет её)

Выводы: в результате выполнения лабораторной работы я изучил общие подходы к тестированию, верификация требований к разрабатываемой системе на основе анализа спецификации.