**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики и информатики**

**Кафедра информационных систем управления**

Реферат по теме

«Тестирование путем покрытия логики программы

(“белый ящик”, “серый ящик”)»

Выполнил:

Студент 1 курса 13 группы

Никончик Даниил Викторович

Минск 2019

Оглавление

[Введение 2](#_Toc108184)

[Глава 1. «История индустриального тестирования» 4](#_Toc108185)

[1.1. Зарождение тестирования. 1950 – 1970 гг. 4](#_Toc108186)

[1.2. Развитие тестирования с 80-х до настоящего времени. 5](#_Toc108187)

[Глава 2. «Современное индустриальное тестирование» 8](#_Toc108188)

[2.1. Классификация индустриального тестирования. 8](#_Toc108189)

[2.2. Описание основных типов тестирования. 9](#_Toc108190)

[2.3. Тестовый цикл и основные этапы тестирования. 13](#_Toc108191)

[2.4. Тестирование как способ обеспечения качества. 15](#_Toc108192)

[Заключение 19](#_Toc108193)

[Список использованных источников: 19](#_Toc108194)

Введение

Тестирование программного обеспечения — проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом. В более широком смысле, тестирование — это одна из техник контроля качества, включающая в себя активности по планированию работ (Test Management), проектированию тестов (Test Design), выполнению тестирования (Test Execution) и анализу полученных результатов (Test Analysis).

Важными понятиями в индустриальном тестировании являются:

* Качество программного обеспечения (Software Quality) — это совокупность характеристик программного обеспечения, относящихся к его способности удовлетворять установленные и предполагаемые потребности.
* Верификация (verification) — это процесс оценки системы или её компонентов с целью определения удовлетворяют ли результаты текущего этапа разработки условиям, сформированным в начале этого этапа. А именно: выполняются ли наши цели, сроки, задачи по разработке проекта, определенные в начале текущей фазы.
* Валидация (validation) — это определение соответствия разрабатываемого ПО ожиданиям и потребностям пользователя, требованиям к системе.

Основными задачами тестирования являются:

1. Повышение вероятности того, что приложение, предназначенное для тестирования, будет работать правильно при любых обстоятельствах.
2. Повышение вероятности того, что приложение, предназначенное для тестирования, будет соответствовать всем описанным требованиям.
3. Предоставление актуальной информации о состоянии продукта на данный момент.

Глава 1. «История индустриального тестирования»

## Зарождение тестирования. 1950 – 1970 гг.

Тестирование зародилось в 50-е годы XX века. Поначалупроцесс тестирования был предельно формализован, отделён от процесса непосредственной разработки ПО и «математизирован». Фактически тестирование представляло собой скорее отладку программ (debugging). Первые программные системы разрабатывались в рамках программ научных исследований или программ для нужд министерств обороны. Тестирование таких продуктов проводилось строго формализовано с записью всех тестовых процедур, тестовых данных, полученных результатов. Тестирование выделялось в отдельный процесс, который начинался после завершения кодирования, но при этом, как правило, выполнялось тем же персоналом.

Существовала концепция т.н. «исчерпывающего тестирования (exhaustive testing)» — проверки всех возможных путей выполнения кода со всеми возможными входными данными. В 1960-х много внимания уделялось «исчерпывающему» тестированию, которое должно проводиться с использованием всех путей в коде или всех возможных входных данных. Было отмечено, что в этих условиях полное тестирование программного обеспечения невозможно, потому что, во-первых, количество возможных входных данных очень велико, во-вторых, существует множество путей, в-третьих, сложно найти проблемы в архитектуре и спецификациях. По этим причинам «исчерпывающее» тестирование было отклонено и признано теоретически невозможным.

В 70-х годахфактически родились две фундаментальные идеи тестирования: тестирование сначала рассматривалось как процесс доказательства работоспособности программы в некоторых заданных условиях (positive testing), а затем — строго наоборот: как процесс доказательства неработоспособности программы в некоторых заданных условиях (negative testing).

В начале 1970-х годов тестирование программного обеспечения обозначалось как «процесс, направленный на демонстрацию корректности продукта» или как «деятельность по подтверждению правильности работы программного обеспечения». В зарождавшейся программной инженерии верификация ПО значилась как «доказательство правильности». Хотя концепция была теоретически перспективной, на практике она требовала много времени и была недостаточно всеобъемлющей. Было решено, что доказательство правильности — неэффективный метод тестирования программного обеспечения. Однако, в некоторых случаях демонстрация правильной работы используется и в наши дни, например, приёмо-сдаточные испытания.

Во второй половине 1970-х тестирование представлялось как выполнение программы с намерением найти ошибки, а не доказать, что она работает. Успешный тест — это тест, который обнаруживает ранее неизвестные проблемы. Данный подход прямо противоположен предыдущему. Указанные два определения представляют собой «парадокс тестирования», в основе которого лежат два противоположных утверждения: с одной стороны, тестирование позволяет убедиться, что продукт работает хорошо, а с другой — выявляет ошибки в программах, показывая, что продукт не работает. Вторая цель тестирования является более продуктивной с точки зрения улучшения качества, так как не позволяет игнорировать недостатки программного обеспечения.

Что касается positive и negative testing – взаимодействие этих видов тестирования превосходно позволяет определить условия, при которых программа ведёт себя некорректно, а также удостовериться, что программа соответствует требования.

## Развитие тестирования с 80-х до настоящего времени.

80-х годах произошло ключевое изменение места тестирования в разработке ПО: вместо одной из финальных стадий создания проекта тестирование стало применяться на протяжении всего цикла разработки (software lifecycle), что позволило в очень многих случаях не только быстро обнаруживать и устранять проблемы, но даже предсказывать и предотвращать их появление. В этот же период времени отмечено бурное развитие, формализация методологий тестирования и появление первых элементарных попыток автоматизировать тестирование. В 1980-е годы тестирование расширилось таким понятием, как предупреждение дефектов.

Проектирование тестов — наиболее эффективный из известных методов предупреждения ошибок. В это же время стали высказываться мысли, что необходима методология тестирования, в частности, что тестирование должно включать проверки на всем протяжении цикла разработки, и это должен быть управляемый процесс.

В ходе тестирования надо проверить не только собранную программу, но и требования, код, архитектуру, сами тесты. «Традиционное» тестирование, существовавшее до начала 1980-х, относилось только к скомпилированной, готовой системе (сейчас это обычно называется системное тестирование), но в дальнейшем тестировщики стали вовлекаться во все аспекты жизненного цикла разработки. Это позволяло раньше находить проблемы в требованиях и архитектуре и тем самым сокращать сроки и бюджет разработки.

В середине 1980-х появились первые инструменты для автоматизированного тестирования. Предполагалось, что компьютер сможет выполнить больше тестов, чем человек, и сделает это более надёжно. Поначалу эти инструменты были крайне простыми и не имели возможности написания сценариев на скриптовых языках.

В 90-х годахпроизошёл переход от тестирования как такового к более всеобъемлющему процессу, который называется «обеспечение качества» (quality assurance), охватывает весь цикл разработки ПО и затрагивает процессы планирования, проектирования, создания и выполнения тест-кейсов, поддержку имеющихся тест-кейсов и тестовых окружений. Тестирование вышло на качественно новый уровень, который естественным образом привёл к дальнейшему развитию методологий, появлению достаточно мощных инструментов управления процессом тестирования и инструментальных средств автоматизации тестирования, уже вполне похожих на своих нынешних потомков.

В начале 1990-х годов в понятие «тестирование» стали включать планирование, проектирование, создание, поддержку и выполнение тестов и тестовых окружений, и это означало переход от тестирования к обеспечению качества, охватывающего весь цикл разработки программного обеспечения. В это время начинают появляться различные программные инструменты для поддержки процесса тестирования: более продвинутые среды для автоматизации с возможностью создания скриптов и генерации отчетов, системы управления тестами, ПО для проведения нагрузочного тестирования.

В середине 1990-х годов с развитием [Интернета](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82) и разработкой большого количества веб-приложений особую популярность стало получать «гибкое тестирование» (по аналогии с гибкими методологиями программирования).

В нулевые годы нынешнего века развитие тестирования продолжалось в контексте поиска всё новых и новых путей, методологий, техник и подходов к обеспечению качества. В 2000-х появилось ещё более широкое определение тестирования, когда в него было добавлено понятие «[оптимизация бизнес-технологий](https://ru.wikipedia.org/w/index.php?title=%D0%9E%D0%BF%D1%82%D0%B8%D0%BC%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D0%B1%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D1%82%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D0%B9&action=edit&redlink=1)». Основной подход заключается в оценке и максимизации значимости всех этапов жизненного цикла разработки программного обеспечения для достижения необходимого уровня качества, производительности, доступности. Серьёзное влияние на понимание тестирования оказало появление гибких методологий разработки и таких подходов, как «разработка под управлением тестированием [1].

Автоматизация тестирования уже воспринималась как обычная неотъемлемая часть большинства проектов, а также стали популярны идеи о том, что во главу процесса тестирования следует ставить не соответствие программы требованиям, а её способность предоставить конечному пользователю возможность эффективно решать свои задачи.

Глава 2. «Тестирование белого ящика и черного ящика»

## Классификация индустриального тестирования.

Упрощённую классификацию тестирования можно представить в виде следующей схемы:



Рисунок 1: Упрощённая классификация тестирования

Тестирование можно классифицировать по:

1. Запуску кода на исполнение:

* Статическое тестирование — без запуска.
* Динамическое тестирование — с запуском.

1. По доступу к коду и архитектуре приложения:

* Метод белого ящика — доступ к коду есть.
* Метод чёрного ящика — доступа к коду нет.
* Метод серого ящика — к части кода доступ есть, к части — нет.

1. По степени автоматизации:

* Ручное тестирование — тест-кейсы выполняет человек.
* Автоматизированное тестирование — тест-кейсы частично или полностью выполняет специальное инструментальное средство.

1. По уровню детализации приложения:

* Модульное (компонентное) тестирование — проверяются отдельные небольшие части приложения.
* Интеграционное тестирование — проверяется взаимодействие между несколькими частями приложения.
* Системное тестирование — приложение проверяется как единое целое.

1. По (убыванию) степени важности тестируемых функций (по уровню функционального тестирования):

* Дымовое тестирование — проверка самой важной, самой ключевой функциональности, неработоспособность которой делает бессмысленной саму идею использования приложения.
* Тестирование критического пути — проверка функциональности, используемой типичными пользователями в типичной повседневной деятельности. Расширенное тестирование — проверка всей (остальной) функциональности, заявленной в требованиях.

1. По принципам работы с приложением:

* Позитивное тестирование — все действия с приложением выполняются строго по инструкции без никаких недопустимых действий, некорректных данных и т.д. Можно образно сказать, что приложение исследуется в «тепличных условиях».
* Негативное тестирование — в работе с приложением выполняются (некорректные) операции и используются данные, потенциально приводящие к ошибкам (классика жанра — деление на ноль).

## Описание основных типов тестирования.

Статическое тестирование (static testing) — тестирование без запуска кода на исполнение. В рамках этого подхода тестированию могут подвергаться: документы (требования, тест-кейсы, описания архитектуры приложения, схемы баз данных и т.д.), графические прототипы (например, эскизы пользовательского интерфейса), код приложения (что часто выполняется самими программистами в рамках аудита кода (code review), являющегося специфической вариацией взаимного просмотра в применении к исходному коду), код приложения также можно проверять с использованием техник тестирования на основе структур кода.

Динамическое тестирование (dynamic testing) — тестирование с запуском кода на исполнение. Запускаться на исполнение может как код всего приложения целиком (системное тестирование), так и код нескольких взаимосвязанных частей (интеграционное тестирование), отдельных частей (модульное или компонентное тестирование) и даже отдельные участки кода. Основная идея этого вида тестирования состоит в том, что проверяется реальное поведение (части) приложения.

Метод белого ящика (white box testing, open box testing, clear box testing) — у тестировщика есть доступ к внутренней структуре и коду приложения, а также есть достаточно знаний для понимания увиденного. Выделяют даже сопутствующую тестированию по методу белого ящика глобальную технику — тестирование на основе дизайна (design-based testing). Для более глубокого изучения сути метода белого ящика рекомендуется ознакомиться с техниками исследования потока управления или потока данных, использования диаграмм состояний. Некоторые авторы склонны жёстко связывать этот метод со статическим тестированием, но ничто не мешает тестировщику запустить код на выполнение и при этом периодически обращаться к самому коду (а модульное тестирование и вовсе предполагает запуск кода на исполнение и при этом работу именно с кодом, а не с «приложением целиком»).

Метод чёрного ящика (black box testing, closed box testing)— у тестировщика либо нет доступа к внутренней структуре и коду приложения, либо недостаточно знаний для их понимания, либо он сознательно не обращается к ним в процессе тестирования. При этом абсолютное большинство перечисленных на рисунках 2.3.b и 2.3.c видов тестирования работают по методу чёрного ящика, идею которого в альтернативном определении можно сформулировать так: тестировщик оказывает на приложение воздействия (и проверяет реакцию) тем же способом, каким при реальной эксплуатации приложения на него воздействовали бы пользователи или другие приложения. В рамках тестирования по методу чёрного ящика основной информацией для создания тест-кейсов выступает документация (особенно — требования) и общий здравый смысл (для случаев, когда поведение приложения в некоторой ситуации не регламентировано явно; иногда это называют «тестированием на основе не-явных требований», но канонического определения у этого подхода нет).

Метод серого ящика (gray box testing) — комбинация методов белого ящика и чёрного ящика, состоящая в том, что к части кода и архитектуры у тестировщика доступ есть, а к части — нет.

Модульное (компонентное) тестирование (unit testing) направлено на проверку отдельных небольших частей приложения, которые (как правило) можно исследовать изолированно от других подобных частей. При выполнении данного тестирования могут проверяться отдельные функции или методы классов, сами классы, взаимодействие классов, небольшие библиотеки, отдельные части приложения. Часто данный вид тестирования реализуется с использованием специальных технологий и инструментальных средств автоматизации тестирования, значительно упрощающих и ускоряющих разработку соответствующих тест-кейсов.

Интеграционное тестирование (integration testing) направлено на проверку взаимодействия между несколькими частями приложения (каждая из которых, в свою очередь, проверена отдельно на стадии модульного тестирования). К сожалению, даже если мы работаем с очень качественными отдельными компонентами, «на стыке» их взаимодействия часто возникают проблемы. Именно эти проблемы и выявляет интеграционное тестирование.

Системное тестирование (system testing) направлено на проверку всего приложения как единого целого, собранного из частей, проверенных на двух предыдущих стадиях. Здесь не только выявляются дефекты «на стыках» компонентов, но и появляется возможность полноценно взаимодействовать с приложением с точки зрения конечного пользователя, применяя множество других видов тестирования, перечисленных в данной главе.

Позитивное тестирование (positive testing) направлено на исследование приложения в ситуации, когда все действия выполняются строго по инструкции без каких бы то ни было ошибок, отклонений, ввода неверных данных и т.д. Если позитивные тест-кейсы завершаются ошибками, это тревожный признак — приложение работает неверно даже в идеальных условиях (и можно предположить, что в неидеальных условиях оно работает ещё хуже). Для ускорения тестирования несколько позитивных тест-кейсов можно объединять (например, перед отправкой заполнить все поля формы верными значениями) — иногда это может усложнить диагностику ошибки, но существенная экономия времени компенсирует этот риск.

Негативное тестирование (negative testing, invalid testing) — направлено на исследование работы приложения в ситуациях, когда с ним выполняются (некорректные) операции и/или используются данные, потенциально приводящие к ошибкам (классика жанра — деление на ноль). Поскольку в реальной жизни таких ситуаций значительно больше (пользователи допускают ошибки, злоумышленники осознанно «ломают» приложение, в среде работы приложения возникают проблемы и т.д.), негативных тест-кейсов оказывается значительно больше, чем позитивных (иногда — в разы или даже на порядки). В отличие от позитивных негативные тест-кейсы не стоит объединять, т.к. подобное решение может привести к неверной трактовке поведения приложения и пропуску дефектов.

Тестирование веб-приложений (web-applications testing) сопряжено с интенсивной деятельностью в области тестирования совместимости (в особенности — кросс-браузерного тестирования) тестирования производительности, автоматизации тестирования с использованием широкого спектра инструментальных средств [2].

Тестирование мобильных приложений (mobile applications testing) также требует повышенного внимания к тестированию совместимости, оптимизации производительности (в том числе клиентской части с точки зрения снижения энергопотребления), автоматизации тестирования с применением эмуляторов мобильных устройств.

Тестирование настольных приложений (desktop applications testing) является самым классическим среди всех перечисленных в данной классификации, и его особенности зависят от предметной области приложения, нюансов архитектуры, ключевых показателей качества и т.д.

## Тестовый цикл и основные этапы тестирования.

Жизненный цикл тестирования схож с жизненным циклом ПО и может быть представлен в виде следующей схемы:

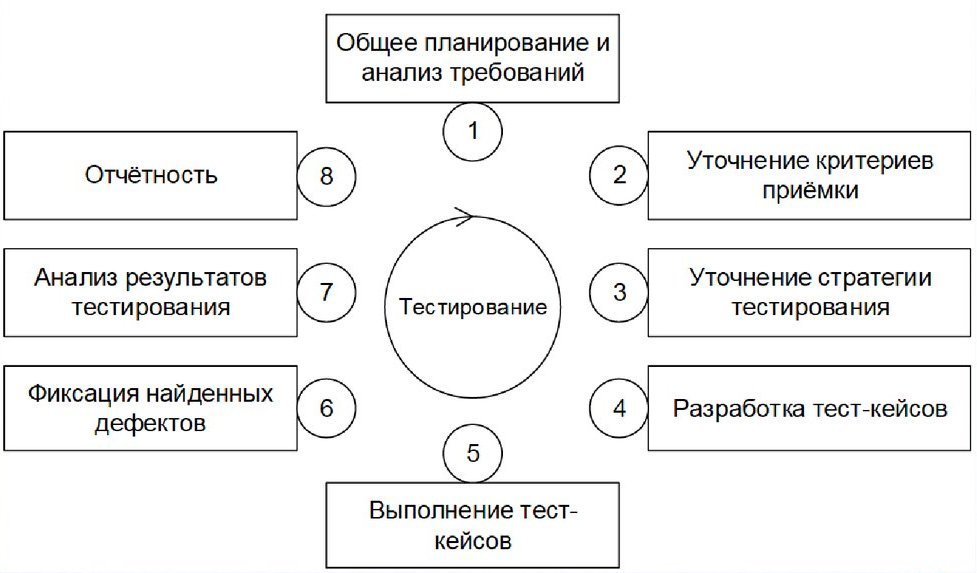


Рисунок 2: Жизненный цикл тестирования

* **Стадия 1** (общее планирование и анализ требований) объективно необходима как минимум для того, чтобы иметь ответ на такие вопросы, как: что нам предстоит тестировать; как много будет работы; какие есть сложности; всё ли необходимое у нас есть и т.п. Как правило, получить ответы на эти вопросы невозможно без анализа требований, т.к. именно требования являются первичным источником ответов.
* **Стадия 2** (уточнение критериев приёмки) позволяет сформулировать или уточнить метрики и признаки возможности или необходимости начала тестирования (entry criteria), приостановки (suspension criteria) и возобновления (resumption criteria) тестирования, завершения или прекращения тестирования (exit criteria).
* **Стадия 3** (уточнение стратегии тестирования) представляет собой ещё одно обращение к планированию, но уже на локальном уровне: рассматриваются и уточняются те части стратегии тестирования (test strategy), которые актуальны для текущей итерации.
* **Стадия 4** (разработка тест-кейсов) посвящена разработке, пересмотру, утонению, доработке, переработке и прочим действиям с тест-кейсами, наборами тест-кейсов, тестовыми сценариями и иными артефактами, которые будут использоваться при непосредственном выполнении тестирования.
* **Стадия 5** (выполнение тест-кейсов) и **стадия 6** (фиксация найденных дефектов) тесно связаны между собой и фактически выполняются параллельно: дефекты фиксируются сразу по факту их обнаружения в процессе выполнения тест-кейсов. Однако зачастую после выполнения всех тест-кейсов и написания всех отчётов о найденных дефектах проводится явно выделенная стадия уточнения, на которой все отчёты о дефектах рассматриваются повторно с целью формирования единого понимания проблемы и уточнения таких характеристик дефекта, как важность и срочность.
* **Стадия 7** (анализ результатов тестирования) и **стадия 8** (отчётность) также тесно связаны между собой и выполняются практически параллельно. Формулируемые на стадии анализа результатов выводы напрямую зависят от плана тестирования, критериев приёмки и уточнённой стратегии, полученных на стадиях 1, 2 и 3. Полученные выводы оформляются на стадии 8 и служат основой для стадий 1, 2 и 3 следующей итерации тестирования. Таким образом, цикл замыкается [3].

## Тестирование как способ обеспечения качества.

Тестирование не позиционируется в качестве единственного способа обеспечения качества. Оно является частью общей системы обеспечения качества продукта, элементы которой выбираются по критерию наибольшей эффективности применения в конкретном проекте.

Рассмотрим пример [4]. В качестве приложения возьмем программу для работы с сетью, критерии качества которой приведены в Таблице 1.

Таблица 1

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Пользователь** | **Заказчик** | **Инженер поддержки** |
| **Функциональная полнота** | + | - | - |
| **Цена разработки** | - | + | - |
| **Отсутствие дефектов** | + | Косвенно | + |
| **Удобство использования** | + | - | - |
| **Возможность внесения изменений в будущем** | - | Косвенно | + |
| **Легкость исправления дефектов** | - | - | + |
| **Документация на реализацию, в том числе комментарии** | - | - | + |
| **Своевременность исполнения проекта** | - | + | - |

Описание критериев качества заинтересованных в них участников для рассматриваемого проекта приведено в [Таблице 2](#table.9.2). Допустим, что вид матрицы критериев качества и проверяющих элементов системы обеспечения качества для данного проекта будет следующим:

Таблица 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Тестирова-ние** | **Анализ рынка и специаль-ные лаборато-рии** | **Обзоры кода** | **Анализ дизайна** | **Аудиты процесса разра-ботки** |
| **Полнота функциональности** | +, не всегда эффективно | + | - | - | - |
| **Стоимость разработки** | - | - | - | - | + |
| **Отсутствие дефектов** | + | - | + | - | - |
| **Удобство использования** | +, не всегда эффективно | + | - | - | - |
| **Возможность внесения изменений в будущем** | - | - | +- | + | - |
| **Легкость исправления дефектов** | - | - | + | + | - |
| **Документация на реализацию, в том числе комментарии** | - | - | + | - | + |
| **Своевременность исполнения проекта** | - | - | - | - | + |

Данные (Таблица 1, Таблица 2) показывают, что из восьми элементов общего качества продукта тестирование способно оценить и контролировать только три (1, 3, 4), причем наиболее эффективно тестирование контролирует отсутствие дефектов (3).

В каждом конкретном проекте элементы системы должны быть выбраны так, чтобы обеспечить приемлемое качество, исходя из приоритетов и имеющихся ресурсов. Выбирая элементы для системы обеспечения качества конкретного продукта, можно применить комбинированное тестирование, обзоры кода, аудит. При подобном выборе некоторые качества, например легкость модификации и исправления дефектов, не будут оценены и, возможно, выполнены. Задачей тестирования в рассматриваемом случае будет обнаружение дефектов и оценка удобства использования продукта, включая полноту функциональности. Исходя из задач, поставленных перед группой тестирования в конкретном проекте, выбирается соответствующая стратегия тестирования. Так, в данном примере, ввиду необходимости оценить удобство использования и полноту функциональности, преимущественный подход к разработке тестов следует планировать на основе использования сценариев.

Итак, основная последовательность действий при выборе и оценке критериев качества программного продукта включает:

1. Определение всех лиц, так или иначе заинтересованных в исполнении и результатах данного проекта.
2. Определение критериев, формирующих представление о качестве для каждого из участников.
3. Приоритезацию критериев, с учетом важности конкретного участника для компании, выполняющей проект, и важности каждого из критериев для данного участника.
4. Определение набора критериев, которые будут отслежены и выполнены в рамках проекта, исходя из приоритетов и возможностей проектной команды. Постановка целей по каждому из критериев.
5. Определение способов и механизмов достижения каждого критерия.
6. Определение стратегии тестирования исходя из набора критериев, попадающих под ответственность группы тестирования, выбранных приоритетов и целей.

# Заключение

В данной работе были кратко описаны основные этапы истории развития индустриального тестирования, а также кратко рассмотрено тестирование на современном этапе: представлена классификация (с кратким описанием основных типов), жизненный цикл, на примере тестирование представлено как способ обеспечения качества выпускаемого программного обеспечения.

Можно сделать вывод, что тестирование – важнейший процесс, позволяющий компаниям, выпускающим программное обеспечение, сэкономить время, материальные ресурсы, а также обеспечить высокое качество выпускаемого продукта. Качественное тестирование – гарант довольных клиентов и безупречной репутации компании на рынке.

## Список использованных источников:

1. Тестирование программного обеспечения. Базовый курс. © EPAM Systems, 2015–2018
2. Г. Майере. // Надежность программного обеспечения. - М.: Мир, 1980. - С. 171-262.
3. Канер С., Фолк Дж., Кек Нгуен Е. Тестирование программного обеспечения: Пер. с англ. — К: ДиаСофт, 2000.
4. Национальный открытый университет [Электронный ресурс], НОУ «ИНТУИТ», 2003 – 2018. Дата доступа: 05.02.2019. Режим доступа: <https://www.intuit.ru/studies/courses/48/48/lecture/1440?page=1>