МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ КЫРГЫЗСКОЙ РЕСПУБЛИКИ

МЕЖДУНАРОДНЫЙ УНИВЕРСИТЕТ КЫРГЫЗСКОЙ РЕСПУБЛИКИ

СРС:

По Введению в ПИ на тему:

«Тестирование в ПО»

Выполнена:

Манас кызы Айзада. ПИ-2-21

Проверена:

Досболова Ф.К.

Бишкек 2022

Оглавление

**Введение**……………………………………………………………………… 4

**Глава 1.1** **По знанию системы**

Глава 1.1.2 Тестирование черного ящика (black box) …………5

Глава 1.1.3 Тестирование белого ящика (white box) …………5

Глава 1.1.4 Тестирование серого ящика (grey box) …………6

**Глава 1.2** **По степени автоматизации**

Глава 1.2.2 Ручное тестирование (manual testing) …………6

Глава 1.2.3 Автоматизированное тестирование

(automated testing) …………7

Глава 1.2.4 Полу автоматизированное тестирование (semi-automated testing) …………7

**Глава 2** **По объекту тестирования**

Глава 2.1 Тестирование интерфейса пользователя (UI testing) …………8

Глава 2.2 Тестирование локализации (localization testing) …………9

Глава 2.3 Тестирование производительности

(Performance testing) …………10

Глава 2.3.2 Нагрузочное тестирование (load testing) …………10

Глава 2.3.3 Стресс-тестирование (stress testing) …………11

Глава 2.3.4 Тестирование стабильности (stability /endurance / soak testing) …………15

Глава 2.4 Юзабилити-тестирование (usability testing) …………13

Глава 2.5 Тестирование безопасности (security testing) …………14

Глава 2.6 Тестирование совместности (compatibility testing) …………14

Глава 2.7 Функциональное тестирование (functional testing) …………

**Глава 3** **По степени изолированности компонентов и времени проведения тестирования**

Глава 3.2 Компонентное(модульное) тестирование

(component / unit testing) …………13

Глава 3.3 Интеграционное тестирование (integration testing) …………

Глава 3.4 Системное тестирование

(system / end-to-end testing) …………14

Глава 3.5 Альфа-тестирование (alpha testing) …………14

Глава 3.5.2 Тестирование новой функциональности

(new feature testing) …………

Глава 3.5.3 Регрессивное тестирование (regression testing) …………

Глава 3.5.4 Дымовое тестирование (smoke testing) …………

Глава 3.5.5 Подтверждающее тестирование

(Confirmation testing) …………

Глава 3.5.6 Приемочное тестирование (**Acceptance testing**) …………15

Глава 3.6 Бета-тестирование (beta testing) …………16

**Заключение**

**Список литературы**

**Введение**

**Тестирование**— данное внутренний проектирование

программного постановления, Основы проектирования, а также испытания. В данном виде тестирований правило  
очевидно тестировщику. Задача заключается в этом, для того чтобы проконтролировать течение входных, а также выходящих информации посредством дополнение, усовершенствовать проектирование, а также практичность применения, а кроме того увеличить защищенность. Испытание белоснежного ящика кроме того именуется испытанием прямого ящика, испытанием прямого ящика, испытанием дизайна, испытанием прямого ящика, испытанием кодовой основы, а также испытанием пустого ящика. Как правило таким образом совершают изготовители.

**Тестирование** программного обеспечения — данное экспериментальный

 процедура, что обнаруживает погрешности o а также программное

 предоставление с целью установления программного предоставления   
практическое, а также ожидаемое действия. Соотношение среди позициями подобрано конкретным способом  
программное предоставление, основанное в серии исследований. В просторном значении

**Глава 1.1** **По знанию системы**

**Глава 1.1.2. Тестирование черного ящика (black box)**

Тестирование методом «черного ящика» — данное способ испытания программного предоставления, что проводит проверку работоспособность программного дополнения, никак не понимая внутренней текстуры программный код, элементов осуществлении, а также внутренних линий. Испытание способом «черного ящика» фокусируется в главную очередность в входных, а также выходящих информации программных дополнений, а также базируется только в условиях, а также спецификациях программного предоставления. Кроме того, именуется поведенческим

 испытанием.

**Глава 1.1.3 Тестирование белого ящика (white box)**

Тестирование методом «белого ящика» — данное программное заключение с целью испытания внутренней текстуры,

дизайна а также кодировки. Данный пробный шифр заметен тестерам.

Я испытываем, а также проектируем течение входных, а также выходящих информации посредством наши дополнения Focus, уделяя особенное интерес увеличению комфорт применения, а также увеличению защищенности. Испытание способом «белого ящика» Испытание «открытого ящика» Состав испытания «открытого ящика»

**Глава 1.1.4 Тестирование серого ящика (grey box)**

**Тестирование серого ящика (grey box)** - Тестирование сероватого ящика  
подразумевает неполное понимание внутренних действий. Данный способ  
предполагает собою комбинацию 2ух предшествующих способов

 (испытание белоснежного ящика, а также темного ящика).

 Специалист стремится отыскать всебезисключения заброшенные трудности, а также погрешности в коде. В данном стадии авто тестер способен

осуществить анализ end-to-end.

**Глава 1.2** **По степени автоматизации**

**Глава 1.2.2 Ручное тестирование (manual testing) -**  часть процесса [тестирования](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) на этапе [контроля качества](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D1%87%D0%B5%D1%81%D1%82%D0%B2%D0%BE_%D0%BF%D1%80%D0%BE%D0%B4%D1%83%D0%BA%D1%86%D0%B8%D0%B8) в процессе разработки [программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5). Оно производится [тестировщиком](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D1%89%D0%B8%D0%BA) без использования программных средств, для проверки программы или сайта путём моделирования действий пользователя. В роли тестировщиков могут выступать и обычные пользователи, сообщая разработчикам о найденных ошибках.

**Глава 1.2.3 Автоматизированное тестирование (automated testing)**

Автоматизированное тестирование или автоматизация тестирования – это метод тестирования программного обеспечения, который выполняется с использованием специальных программных средств, которые, в свою очередь необходимы для выполнения набора тестовых примеров. Напротив, ручное тестирование выполняется человеком, сидящим перед компьютером и тщательно выполняющим каждый шаг теста «руками». Программное обеспечение для автоматизации тестирования также может вводить тестовые данные в тестовую среду, сравнивать ожидаемые и фактические результаты и создавать подробные отчеты о тестах. Как правило, автоматизация тестирования требует значительных вложений денег и ресурсов.

**Глава 1.2.4 Полу автоматизированное тестирование**

**(semi-automated \testing)**

**Автоматизированное тестирование** предполагает использование специального программного обеспечения (помимо тестируемого) для контроля выполнения тестов и сравнения ожидаемого результата работы программы с фактическим.

Этот тип тестирования **помогает автоматизировать часто повторяющиеся**, но необходимые для максимизации тестового покрытия, задачи.

Это отдельная дисциплина искусства тестирования. Значительная часть эффективности работы отдела тестирования зависит от того, какие задачи отданы для автоматизации и как эта автоматизация была осуществлена.

. **Глава 2** **По объекту тестирования**

**Глава 2.1 Тестирование интерфейса пользователя (UI testing)**

Тестирование пользовательского интерфейса — это процесс проверки того, соответствует ли пользовательский интерфейс программного обеспечения требованиям, и удобно ли пользователям работать с программным продуктом.  
Этот тип тестирования можно выполнять вручную и с помощью различных инструментов автоматизированного тестирования.

Тестирование пользовательского интерфейса предполагает проверку того, ведет ли себя программное обеспечение в соответствии со спецификацией, когда пользователь взаимодействует с ним с помощью клавиатуры и мыши в случае тестирования десктопного приложения, или с помощью сенсорного экрана, жестов или движений устройства, когда тестируется мобильное приложение.

**Глава 2.2 Тестирование локализации (localization testing)**

**Локализационное тестирование** — это проверка содержимого приложения или сайта на соответствие лингвистическим, культурным требованиям, а также специфике конкретной страны или региона.  
  
Тестирование локализации — один из видов контроля качества, который проводится во время разработки продукта. Этот тип тестирования помогает найти баги или ошибки перевода в локализованной версии до того, как конечный продукт попадет к пользователю. Цель тестирования — поиск и устранение ошибок в различных локализованных версиях продукта, предназначенных для разных рынков и локалей.

Главная задача тестирования заключается в том, чтобы продукт выглядел так, как будто изначально был создан на языке целевой аудитории и полностью соответствовал культурным и региональным особенностям.  
  
Локализация повышает лояльность клиентов к вашему бренду. Вот конкретные цифры: около [72,1% пользователей Интернета](https://insights.csa-research.com/reportaction/8057/Marketing) предпочитают делать покупки на сайтах на своем родном языке. Даже те, кто хорошо владеет английским, по-прежнему [предпочитают просматривать веб-страницы на своем родном языке](https://insights.csa-research.com/reportaction/8057/Marketing).

**Глава 2.3 Тестирование производительности (Performance testing)**

**Тестирование производительности** [инженерии программного обеспечения](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B6%D0%B5%D0%BD%D0%B5%D1%80%D0%B8%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) — [тестирование](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F), которое проводится с целью определения, как быстро работает вычислительная система или её часть под определённой [нагрузкой](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D0%B3%D1%80%D1%83%D0%B7%D0%BA%D0%B0). Также может служить для проверки и подтверждения других атрибутов качества системы, таких как [масштабируемость](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%88%D1%82%D0%B0%D0%B1%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D0%BE%D1%81%D1%82%D1%8C), [надёжность](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D0%B4%D1%91%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C) и потребление ресурсов.

Тестирование производительности — это одна из сфер деятельности развивающейся в области [информатики](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0) [инженерии производительности](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B6%D0%B5%D0%BD%D0%B5%D1%80%D0%B8%D1%8F_%D0%BF%D1%80%D0%BE%D0%B8%D0%B7%D0%B2%D0%BE%D0%B4%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D0%B8), которая стремится учитывать производительность на стадии моделирования и проектирования системы, перед началом основной стадии кодирования[

**Глава 2.3.2 Нагрузочное тестирование (load testing)**

**Нагрузочное тестирование** — подвид тестирования производительности, сбор показателей и определение производительности и времени отклика программно-технической системы или устройства в ответ на внешний запрос с целью установления соответствия требованиям, предъявляемым к данной системе (устройству).

Для исследования времени отклика системы на высоких или пиковых нагрузках производится [стресс-тестирование](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%B5%D1%81%D1%81-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), при котором создаваемая на систему нагрузка превышает нормальные сценарии её использования. Не существует чёткой границы между нагрузочным и стресс-тестированием, однако эти понятия не стоит смешивать, так как эти виды тестирования отвечают на разные бизнес-вопросы и используют различную методологию.

В общем случае под нагрузочным тестированием понимается практика моделирования ожидаемого использования приложения с помощью эмуляции работы нескольких пользователей одновременно. Таким образом, подобное тестирование больше всего подходит для многопользовательских систем, чаще — использующих клиент-серверную

**Глава 2.3.3 Стресс-тестирование (stress testing)**

**Стресс-тестирование** (stress testing) — одна из форм [тестирования](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F), которая используется для определения устойчивости системы или модуля в условиях превышения пределов нормального функционирования.

Модель стресс-теста определяет сценарий, использующий специальный алгоритм для определения ожидаемого воздействия на возврат портфеля в случае реализации сценария.

Существуют три основных типа сценариев:

* Экстремальные события: в качестве события часто используют уже происходившее историческое событие.
* Шок риск-фактора: шок любого фактора в выбранной модели риска на заданную пользователем сумму. Фактор воздействия остается неизменным, а для оценки возврата используется ковариационная матрица связи с риск-фактором.
* Внешний риск-фактор: фактором риска является любой макроэкономический индекс (например, цены на нефть, цены на недвижимость), либо устанавливаемые индексы (например, курсы валют).

**Глава 2.3.4 Тестирование стабильности (stability /endurance/ soak testing)**

**Тестирование стабильности или надежности** (Stability / Reliability Testing) — один из видов нефункционального [тестирования ПО](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F), целью которого является проверка работоспособности приложения при длительном тестировании с ожидаемым уровнем нагрузки.

Перед тем как подвергать ПО экстремальным нагрузкам стоит провести проверку стабильности в предполагаемых условиях работы, то есть погрузить продукт в полную рабочую атмосферу. При тестировании, длительность его проведения не имеет первостепенного значения, основная задача — наблюдая за потреблением ресурсов, выявить утечки памяти и проследить чтобы скорость обработки данных и/или время отклика приложения в начале теста и с течением времени не уменьшалась. В противном случае вероятны сбои в работе продукта и перезагрузки системы.

**Глава 2.4 Юзабилити-тестирование (usability testing)**

**Проверка эргономичности** (**Юзабилити-тестирование**) — [исследование](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), выполняемое с целью определения, удобен ли некоторый искусственный объект (такой, как [веб-страница](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0), [пользовательский интерфейс](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81) или устройство) для его предполагаемого применения. Таким образом, проверка эргономичности измеряет [эргономичность](https://ru.wikipedia.org/wiki/%D0%AD%D1%80%D0%B3%D0%BE%D0%BD%D0%BE%D0%BC%D0%B8%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C) объекта или системы. Проверка эргономичности сосредоточена на определённом объекте или небольшом наборе объектов, в то время как исследования [взаимодействия человек-компьютер](https://ru.wikipedia.org/wiki/%D0%A7%D0%B5%D0%BB%D0%BE%D0%B2%D0%B5%D0%BA%D0%BE-%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81) в целом — формулируют универсальные принципы.

Проверка эргономичности — метод оценки удобства продукта в использовании, основанный на привлечении пользователей в качестве тестировщиков, испытателей и суммировании полученных от них выводов.

**Глава 3.** **По степени изолированности компонентов и времени проведения тестирования**

**Глава 3.2 Компонентное(модульное) тестирование (component / unit testing)**

Компонентное (модульное) тестирование проверяет функциональность и ищет дефекты в частях приложения, которые доступны и могут быть протестированы по-отдельности (модули программ, объекты, классы, функции и т.д.). Обычно компонентное (модульное) тестирование проводится вызывая код, который необходимо проверить и при поддержке сред разработки, таких как Фреймворки (frameworks - каркасы) для модульного тестирования или инструменты для отладки. Все найденные [дефекты](http://www.protesting.ru/testing/bugreport.html), как правило исправляются в коде без формального их описания в системе менеджмента багов (Bug Tracking System).

Один из наиболее эффективных подходов к компонентному (модульному) тестированию - это **подготовка автоматизированных тестов** до начала основного кодирования (разработки) программного обеспечения. Это называется разработка от тестирования (test-driven development) или подход тестирования вначале (test first approach).

**Глава 3.4 Системное тестирование (system / end-to-end testing)**

**Системное тестирование** – это тестирование программного обеспечения, выполняемое на полной, интегрированной системе, с целью проверки соответствия системы исходным требованиям, как функциональным, так и не функциональным.

**Глава 3.5 Альфа-тестирование (alpha testing)**

Альфа-тестирование — имитация реальной работы с системой штатными [разработчиками](https://ru.m.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%81%D1%82), либо реальная работа с системой потенциальными пользователями/заказчиком. Чаще всего альфа-тестирование проводится на ранней стадии разработки продукта, но в некоторых случаях может применяться для законченного продукта в качестве внутреннего приёмочного тестирования. Иногда альфа-тестирование выполняется под отладчиком или с использованием окружения, которое помогает быстро выявлять найденные ошибки. Обнаруженные ошибки могут быть переданы тестировщиком для дополнительного исследования в окружении, подобном тому, в котором будет использоваться программа.

**Глава 3.5.6 Приемочное тестирование** (**Acceptance testing)**

**Приемочное (acceptance)** – вид тестирования, проводимый на этапе сдачи готового продукта (или готовой части продукта) заказчику. Целью приемочного тестирования является определение готовности продукта, что достигается путем прохода тестовых сценариев и случаев, которые построены на основе спецификации требований к разрабатываемому ПО.

Результатом приемочного тестирования может стать:

• Отправка проекта на доработку.

• Принятие его заказчиком, в качестве выполненной задачи.

Это финальный этап тестирования продукта перед его релизом. При этом, он не является сверх тщательным, всеохватывающим и полным – тестируется, в основном, только основной функционал.

**Глава 3.6 Бета-тестирование (beta testing)**

[**Бета-тестирование**](https://ru.m.wikipedia.org/wiki/%D0%91%D0%B5%D1%82%D0%B0-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) — в некоторых случаях выполняется распространение предварительной версии (в случае пропри тарного программного обеспечения иногда с ограничениями по функциональности или времени работы) для некоторой большей группы лиц с тем, чтобы убедиться, что продукт содержит достаточно мало ошибок. Иногда бета-тестирование выполняется для того, чтобы получить обратную связь о продукте от его будущих пользователей.

Часто для свободного и открытого программного обеспечения стадия [альфа-тестирования](https://ru.m.wikipedia.org/wiki/%D0%90%D0%BB%D1%8C%D1%84%D0%B0-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) характеризует функциональное наполнение кода, а [бета-тестирования](https://ru.m.wikipedia.org/wiki/%D0%91%D0%B5%D1%82%D0%B0-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) — стадию исправления ошибок. При этом как правило на каждом этапе разработки промежуточные результаты работы доступны конечным пользователям.