МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования «Северо-Кавказский федеральный университет»

Кафедра инфокоммуникаций

Реферат

«Программная инженерия для игр: технологии, инструменты и методы разработки»

по дисциплине « Программная инженерия»

Выполнил студент группы ПИЖ-б-о-21-1
Ботвинкин Н.С. « »_____20__г.
Подпись студента_____
Работа защищена « »_____20__г.
Проверил Воронкин Р.А. _____
(подпись)

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ В ИГРОВУЮ ИНДУСТРИЮ

1.1 Темы изучения

Любой, кто разрабатывает программное обеспечение для игр, может извлечь пользу из знаний в области software. Разработка программного обеспечения представлена как набор инструментов, из которого разработчики могут выбирать необходимые им инструменты. Большая свобода выбора инструментов для разработчиков и то, как их использовать очень полезно. Получение четкого представления о доступных вам инструментах и способах их применения позволит вам усовершенствовать свои усилия в области разработки. Изучение этих тем приведет ко многим другим темам:

- Сочетание инженерии и искусства
- Программное обеспечение как практика
- Программное обеспечение как профессиональная область деятельности
- Подходы к обучению проектированию и конструированию продукта
- Разработка программного обеспечения
- Обучение навыкам, которые можно применить, где угодно

Разработка программного обеспечения касается всего, что связано с разработкой программного обеспечения. Неофициально ее можно определить как изучение и практику того, как улучшить разработку программного обеспечения. Разработка игр имеет свою собственную культуру, и эта культура включает в себя гораздо больше, чем просто разработка программного обеспечения.

1.2 Команда

В любом проекте по разработке игр задействованы усилия многих творческих личностей, которые выполняют самую разнообразную работу. Среди этих людей есть те, кто разрабатывает дизайн игр, создает графические элементы, сочиняет музыку и диалоги. Это лишь некоторые из многих людей, которые могут быть вовлечены в процесс разработки игр.

Вряд ли имеет смысл утверждать, что те, кто занимается всеми этими вещами, должны испытывать какие-либо серьезные опасения по поводу разработки программного обеспечения. Хотя программное обеспечение остается основополагающим, оно составляет лишь часть общей картины. Тем не менее, разработка программного обеспечения приносит пользу всем, кто участвует в разработке игр, поскольку позволяет разработчикам программного обеспечения выявлять и применять надежные, проверенные методы в своей работе. Их усилия направлены на удовлетворение потребностей других людей.

Возьмем, к примеру, геймдизайнера. Этот человек работает долгие часы, чтобы точно определить мир игры. Этот человек зависит от разработчиков программного обеспечения, которые разрабатывают функциональность игры таким образом, чтобы мир часть игры может быть реализована. Когда разработчики программного обеспечения приступают к разработке этой функциональности, они должны позаботиться о том, чтобы полностью понять, что требуется геймдизайнеру, прежде чем приступать к разработке программного обеспечения; в противном случае у них мало шансов оправдать ожидания дизайнера.

Программная инженерия включает в себя совокупность знаний, которыми руководствуются разработчики при разработке программного обеспечения. Одна из рекомендаций, которую эта совокупность знаний предлагает разработчикам, заключается в том, что они должны тщательно собирать и анализировать требования к программной системе, которую они разрабатывают. их попросили разработать. Еще одна рекомендация заключается в том, что они должны разработать дизайн программного

обеспечения, который полностью соответствует требованиям. Среди других рекомендаций разработчики должны протестировать разрабатываемую ими систему, чтобы убедиться, что ее дизайн и реализация точно соответствуют требованиям. Если разработчики программного обеспечения будут следовать этим рекомендациям при работе с геймдизайнером, они будут надлежащим образом учитывать потребности дизайнера. Они будут эффективно использовать время дизайнера и использовать устоявшиеся методы и процессы. Они также разработает продукт без избыточности и переделок.

Используя методы разработки программного обеспечения, разработчики программного обеспечения будут знать, как задавать правильные вопросы и как наилучшим образом использовать собранную информацию.

1.3 Программный продукт

Программная инженерия — это формальная дисциплина, которая направлена на повышение качества процессов разработки программного обеспечения и продуктов, которые являются их результатом. Когда программный продукт обладает качеством, он, помимо прочего, надежен, его можно обслуживать и расширять.

Надежный программный продукт — это тот, который работает должным образом и не имеет дефектов.

Устойчивый программный продукт — это тот, который вы можете исправить, если он неисправен.

Разработка программного обеспечения широко распространена, поскольку разработка развлекательных программных продуктов сопряжена с финансовыми рисками. Корпорации сталкиваются с рисками, когда они нанимают людей, не обладающих соответствующей профессиональной квалификацией. Корпорации ищут людей, которые могут выполнять свою работу эффективно. Такая эффективность в конечном счете является результатом применения дисциплинированных подходов к работе.

Подобные заявления отдают своего рода авторитаризмом, который многие программисты и другие лица, занимающиеся разработкой игр, находят тревожным. Учитывая романтические образы, присутствующие во многих ранних историях разработки игр, идея о том, что разработка игр превратилась в сферу инженерного мастерства, а не героического энтузиазма, оказывается обескураживающей.

Степень разочарования зависит от того, как вы смотрите на общую картину. Общая картина определенно показывает, что несколько десятков разработчиков игр и издателей доминируют в промышленности.

Разработка игр часто характеризуется составлением производственного плана, обсуждением бюджета и крупными мероприятиями по управлению персоналом. Выжить в этом мире сложно. Конкуренция безжалостна, и шансы на успех минимальны.

Однако за пределами внутреннего круга преобладающих корпоративных интересов многие тысячи независимых разработчиков создают игры, которые они надеются продать массовому рынку.

Ситуация аналогична некоторым аспектам музыкальной и киноиндустрии. Хотя несколько дистрибьюторов, студий, издателей и художников занимают центральное место на рынке и обеспечивают большую часть его доходов, в любой день любого года талантливые люди объединяются в группы и работают вместе над созданием новых продуктов.

2. ИГРОВЫЕ ДВИЖКИ И ИХ ОСОБЕННОСТИ (UNITY, UNREAL ENGINE, GODOT ENGINE)

2.1 Unity

В современном мире игры настолько сложные, с нуля их никто практически не создает. Как правило, используется какой-то фреймворк (движок) для создания компьютерных игр. Использование фреймворка существенно упрощает (следовательно, ускоряет) процесс разработки игры.

Одним из таких средств является Unity. Unity-межплатформенная среда разработки компьютерных игр. С ее помощью можно создавать игры, работающие под более чем 20 различными операционными системами - настольные компьютеры (стационарные и ноутбуки), игровые консоли, мобильные устройства, Интернет-приложения и др.

Основная задача Unity - демократизировать разработку игр. Теперь разработка собственной игры доступна каждому желающему, в том числе студенту и для старта вам не понадобится космическая сумма денег

На протяжении многих лет основное внимание разработчиков Unity было сосредоточено на демократизации разработки игр, чтобы любой желающий мог написать игру и сделать ее доступной самой широкой аудитории. Однако никакой программный пакет не может идеально подходить для всех ситуаций, поэтому вы необходимо знать, когда с успехом можно использовать Unity, а когда лучше поискать другой программный пакет.

Движок Unity особенно хорошо подходит в следующих ситуациях.

- При создании игры для нескольких устройств. Кроссплатформенная поддержка Unity является, пожалуй, лучшей в индустрии, и если необходимо создать игру, действующую на нескольких платформах (или даже на нескольких мобильных платформах), Unity может оказаться лучшим выбором для этого.
- Когда важна скорость разработки. Вы можете потратить месяцы на разработку игрового движка, обладающего всеми необходимыми

функциями. Или можете использовать сторонний движок, такой как Unity. Справедливости ради нужно сказать, что существуют другие движки, такие как Unreal или Cocos2D;

- Когда требуется полный набор функций, но нет желания создавать собственный комплект инструментов. Unity обладает идеальными возможностями для создания игр и поддерживает очень простые способы формирования их контента.

При этом в некоторых ситуациях Unity оказывается не так полезен. В том числе:

В играх, не требующих частой перерисовки сцены.

- Unity хуже подходит для реализации игр, не требующих интенсивных операций с графикой, так как движок Unity перерисовывает каждый кадр. Это необходимо для поддержки анимации в масштабе реального времени, но требует больших затрат энергии.
- Когда требуется точное управление действиями движка.
- Отказавшись от приобретения лицензии на исходный код Unity (такая возможность есть, но она используется редко), вы теряете возможность контролировать поведение движка на низком уровне. Это не значит, что вы вообще теряете контроль над работой движка Unity (в большинстве случаев этого и не требуется), но кое-что окажется вам недоступно.

2.2 Unreal Engine

Unreal Engine — это мощный движок для создания игр и других интерактивных приложений, разработанный и поддерживаемый компанией Epic Games. Он используется для разработки игр на различных платформах, включая ПК, консоли, мобильные устройства и VR. Первая версия движка была выпущена в 1998 году, и с тех пор он прошел множество обновлений и улучшений.

Unreal Engine известен своими возможностями в области графики. Он поддерживает передовые визуальные эффекты, включая трассировку лучей (ray tracing), что позволяет создавать фотореалистичное изображение.

Система blueprints визуального скриптинга, позволяет создавать сложную логику без написания кода. Это делает движок доступным для дизайнеров, не обладающих навыками программирования.

Unreal Engine предоставляет доступ к исходному коду на C++, что позволяет разработчикам глубоко модифицировать движок под свои нужды.

Существует обширный рынок контента, где можно найти множество готовых моделей, анимаций, текстур и других ресурсов для разработки игр.

Встроенные инструменты для работы с анимацией, физикой, искусственным интеллектом и сетевыми возможностями.

Поддерживает множество платформ, от ПК до мобильных устройств и VR.

Unreal Engine Известен своими передовыми графическими возможностями и качеством визуализации. Часто используется для AAA-игр.

Unity: Тоже обладает мощными графическими возможностями, но считается более подходящим для разработки игр средней и малой сложности, а также 2D-игр. Основной язык Unreal Engine — C++. Также есть визуальный скриптинг (Blueprints). В Unity же, основной язык — C#. Unity предоставляет удобный API для работы с различными аспектами разработки.

Система скриптинга Unreal Engine: Blueprints позволяют создавать логику игры без написания кода, что удобно для дизайнеров и прототипирования. Unity: Использует визуальный скриптинг Bolt, но он менее интегрирован, чем Blueprints в Unreal.

Оба движка поддерживают разработку под множество платформ, но Unity имеет преимущество в мобильных и 2D-играх.

Unreal Engine может быть более сложным для начинающих, особенно если учитывать C++ и мощные графические возможности. Unity считается

более дружелюбным для новичков и имеет большую базу учебных материалов.

Unity имеет большое сообщество разработчиков и множество доступных ресурсов, учебников и курсов. В то же время Unreal Engine обладает активным сообществом и обширной библиотекой ресурсов, но количество учебных материалов может быть меньше, чем у Unity.

Выбор между Unreal Engine и Unity зависит от конкретных потребностей проекта. Unreal Engine лучше подходит для проектов, требующих высококачественной графики и сложной логики, тогда как Unity может быть предпочтительнее для мобильных приложений, 2D-игр и проектов с меньшим бюджетом и временем разработки.

2.3 Godot Engine

Godot Engine — это мощный и гибкий движок для разработки игр, который обладает уникальными особенностями. Он полностью бесплатен и имеет открытый исходный код, что позволяет разработчикам свободно изменять и улучшать его в соответствии с потребностями их проектов. Одним из главных преимуществ Godot является интуитивно понятный интерфейс, который делает его доступным даже для начинающих разработчиков.

Godot поддерживает как 2D, так и 3D графику, причем 2D-режим выполнен на высоком уровне и работает отдельно от 3D, что позволяет избежать многих проблем с производительностью. Уникальная система сцен позволяет организовывать проект в виде вложенных сцен, что значительно упрощает управление сложными проектами. Основным языком программирования является GDScript, который напоминает Python и прост в освоении. Кроме того, движок поддерживает C# и VisualScript для визуального скриптинга.

Одним из значимых плюсов Godot является его мультиплатформенность: он поддерживает множество платформ, включая Windows, macOS, Linux, HTML5, Android и iOS. Встроенный редактор Godot

позволяет редактировать сцены, скрипты, анимации и другие ресурсы прямо в движке, что делает процесс разработки более удобным и быстрым. Он также легко интегрируется с системами управления версиями, такими как Git, и активно развивается благодаря усилиям большого сообщества разработчиков.

Сравнивая Godot с другими популярными движками, можно выделить несколько ключевых различий. Например, Unreal Engine, разработанный Epic Games, известен своими передовыми графическими возможностями и качеством визуализации, включая поддержку трассировки лучей. Unreal Engine используется для создания высокобюджетных AAA-игр и предлагает мощные инструменты для работы с графикой и физикой. Основным языком программирования для Unreal является C++, что может представлять сложность для начинающих. Кроме того, движок поддерживает Blueprints — систему визуального скриптинга, которая позволяет создавать игровую логику без написания кода. В сравнении с Unreal, Godot менее требователен к ресурсам и более прост в освоении, особенно для новичков.

Другой популярный движок — Unity также поддерживает как 2D, так и 3D графику и является универсальным инструментом для разработки игр на различных платформах. Основным языком программирования в Unity является C#, что делает его доступным для многих разработчиков. Unity известен своим огромным сообществом и обилием учебных материалов, что облегчает процесс обучения. В отличие от Godot, Unity имеет бесплатную версию с ограничениями и платные лицензии для профессионального использования. Unity и Godot схожи в плане доступности и удобства, но Unity может предложить больше возможностей для профессиональной разработки и имеет более развитую экосистему.

В итоге, выбор между Godot, Unreal Engine и Unity зависит от конкретных нужд проекта. Godot отлично подходит для разработчиков, ищущих бесплатный, открытый и интуитивно понятный движок для создания 2D и 3D игр. Unreal Engine является лучшим выбором для проектов с высокими требованиями к графике и сложной логике, в то время как Unity

является универсальным решением, подходящим для различных типов проектов, включая мобильные и 2D игры. Разработка на Godot может стать отличным выбором для инди-разработчиков и тех, кто хочет быстро освоить основы создания игр.

3. СПЕЦИФИКА РАЗРАБОТКИ ИГРОВЫХ ПРИЛОЖЕНИЙ.

3.1 Начало разработки

С точки зрения разработки программного обеспечения, отправной точкой усилий по разработке игр является директива. Директива исходит из источника, который обычно находится за пределами группы разработчиков. Документ по дизайну игры обеспечивает наиболее наглядное выражение директивы. Документ по дизайну игры содержит информацию, которую вы можете использовать для определения масштаба ваших усилий. По мере роста организаций специализация увеличивается, поэтому, если вы отвечаете за разработку программного обеспечения, вы можете обнаружить, что ваша роль в разработке игры сводится к созданию программного обеспечения, поддерживающего функции, о которых мечтали другие люди. Если эта картина вас обескураживает, подумайте о том, что инженеры почти всегда находятся в одинаковом положении. Архитекторы работают с инженерами-конструкторами, автомобильные дизайнеры - с механиками. инженеры и геймдизайнеры работают вместе с инженерами-программистами. Одно поддерживает другое. Практически ни в одной отрасли не наблюдается такого разделения труда. В разных ситуациях все может быть по-разному, но если вы занимаете должность инженера, а другие - дизайнера, ваша роль начинается с предоставления определенного продукта другим пользователям.

Предоставляемый вами продукт — это программное обеспечение, которое делает игру возможной. По этой причине вашей отправной точкой будет понимание того, что необходимо для поддержки функций игры.

Разработка игр начинается с нескольких вещей, которые не связаны с программированием. Среди них контракты, документация по дизайну игры и корпоративный производственный план. Необходимо заручиться правами, утвердить финансирование и составить корпоративный производственный график. После завершения всех этих работ для разработки программы выбирается руководитель проекта или руководитель группы.

Первой задачей руководителя проекта, как правило, является оценка того, сколько усилий потребует разработка программного обеспечения, поддерживающего игру, и сколько людей потребуется для завершения проекта в срок. Хотя ни одна формула не охватывает все ситуации, если вы окажетесь на должности менеджера проекта, вы, вероятно, придете к выводу, что вам нужна команда, способная выполнять несколько основных видов деятельности:

- Планировать разработку и управлять ею.
- Знать ответы на сложные технические вопросы.
- Отвечать на вопросы о том, как была разработана игра и какие цели должна обеспечивать ее функциональность.
- Выполнять второстепенные или специализированные задачи по программированию, требующие значительных временных затрат.
- Создавать документацию для проекта.
- Выполнять или координировать работу, связанную с игровыми ресурсами, такими как звук и графика.
- Работать над инструментами, которые могут поддерживать различные мероприятия по разработке.
- Настраивать конфигурацию программного обеспечения.
- Формально определить роль тестировщика программного обеспечения.

В процессе работы над проектом каждый в конечном итоге возьмет на себя и определит роли, которые будут входить в эту деятельность. Здесь не утверждается, что, если вы хотите создать игру, вам следует найти определенное количество людей и поручить им выполнять определенные роли. На самом деле все обстоит не так. Даже в самых монотонных корпоративных условиях люди по-прежнему изучают и открывают для себя новые роли в процессе разработки.

В начале работы над игровым программным обеспечением вы, скорее всего, будете сидеть в комнате с группой разработчиков и обсуждать игру. Все,

вероятно, полны энтузиазма. У каждого на уме что-то замечательное. Это, безусловно, хорошо. На такие обсуждения может уйти немало времени. В ходе обсуждения могут обсуждаться прошлые усилия, существующие ресурсы для разработки кода, чего следует избегать и какие инструменты использовать.

3.2 Создание концепта

Однако в какой-то момент, чтобы что-то было сделано, группа должна сформулировать несколько общих тем:

- Команда разработчиков должна разработать программное обеспечение. Дизайн вытекает из требований, и требования должны основываться на документации по дизайну игры. Команда должна понимать сферу применения игры и возможности продажи. Аналогичным образом, она должна разработать дизайн программного обеспечения.
- Команда должна планировать проект и управлять им. Руководитель проекта должен понимать требования, определять бюджет, сроки и количество необходимых сотрудников. Добавьте к этому, что каждый член команды должен понимать план разработки продукта и то, что он должен делать в рамках командной работы.
- Команда должна протестировать все. Для проведения тестирования к разработке должен быть привлечен специалист по тестированию. Разные школы придерживаются разных подходов к тестированию программного обеспечения. Специалист по тестированию — это тот, кто работает с основным разработчиком над обнаружением и устранением программных ошибок.

Хороший способ узнать о своем продукте можно, создав описание области применения и спецификацию требований. Описание области применения позволяет всем членам команды разработчиков иметь общее представление о проекте, а описание требований заставляет каждого потратить несколько дней на точное определение того, что именно должно быть разработано.

Менеджмент предполагает сдерживание желания начать работу над игрой, прежде чем тратить время на то, чтобы понять, что изображено в документе по дизайну игры. Дизайн предполагает начало болезненного процесса обдумывания того, что требуется, и придания этому определенной формы это позволяет разрабатывать его систематически. Тестирование включает в себя анализ действий при запуске, чтобы убедиться, что разработчики с самого начала уточняют, проверяют и валидируют функции.

В центре внимания при разработке проекта разработки программного обеспечения находится спецификация на разработку программного обеспечения. Эта спецификация может стать центральным документом в вашей работе по разработке. По мере продвижения проектирования, команда все больше полагалась на документ по разработке программного обеспечения.

Разработка проектной документации программного обеспечения требовала непрерывных итераций, и документ не был неизменным до конца проекта.

С самого начала было важно представить игровое программное обеспечение в виде набора разделов (или компонентов). Создание разделов заставило команду сконцентрироваться на конкретных задачах по внедрению.

3.3 Организация работы

Команда должна настроить способ управления версиями кода как можно раньше на этапе разработки. Это связано с тем, что на этапе проектирования разработчики могут решить изучить концепции дизайна с помощью кода. Команда должна выполнить формальную версию такого кода, поскольку он, скорее всего, будет представлен позже в рамках проекта.

При настройке системы управления версиями следует руководствоваться планом настройки. Создание полной структуры каталогов невозможно до тех пор, пока вы не создадите документ по разработке программного обеспечения и не будете знать, какие разделы (также

называемые модулями или компонентами) и какие группы компонентов внутри разделов вы хотите создать.

На ранних этапах проекта план настройки может начинаться как документ, состоящий из нескольких частей. Важно озаглавить документ и поместить его под контроль версий. Поместить документ под контроль версий можно так же просто, как присвоить ему номер версии, который вы можете ввести непосредственно под названием плана на титульной странице.

Прежде чем приступить к написанию кода, вам необходимо спланировать его. Прежде чем приступить к планированию кода, вам необходимо понять, какой продукт вас просят создать. Чтобы понять, какой продукт вам предлагается создать, вы можете использовать различные хорошо протестированные инструменты, которые помогут вам понять и концептуализировать систему, которую вы хотите создать. Одними из лучших инструментов являются обсуждения, списки и диаграммы.

Когда вы начнете обсуждать проект программного обеспечения, попросите всех участников ознакомиться с документом по игровому дизайну. Если вы руководитель проекта, вы можете подумать о том, чтобы сделать бумажные копии для всех членов вашей команды. Другой подход заключается в сохранении документа по-игровому дизайну в формате HTML и размещении его на веб-сайте проекта.

Причина, по которой все должны ознакомиться с документом по дизайну, заключается в том, что он знакомит всех с основными принципами работы. Просмотр макета в целом позволяет каждому члену команды почувствовать свою причастность к каждому аспекту игры.

3.4 Ранние тесты

Тестирование должно начинаться как можно раньше, и все, что можно протестировать, должно быть протестировано. Тестирование в начале проекта предполагает анализ. Как правило, анализ отличается от того типа тестирования, который многие люди представляют себе. Стоит упомянуть, что

представление о том, что тестирование начинается после завершения разработки, представляет собой устаревший взгляд на разработку программного обеспечения. Такой взгляд на тестирование, как правило, слишком сильно фокусируется на выполнении тестирования, на представлении о том, что тестирование должно выполнять с помощью кода. Это вполне справедливо, но тестирование также имеет отношение ко всему, что связано с кодом, включая требования, дизайн и документацию.

Применительно к требованиям и дизайну одним из основных направлений тестирования может быть тестируемость требований и вариантов использования. Например, когда разработчики разрабатывают варианты использования для удовлетворения требований, тестировщики могут оценить варианты использования, чтобы определить, могут ли они предоставить подходящие сценарии для тестовых случаев и наборов тестов. Одним из наиболее надежных критериев тестируемости для любого конкретного варианта использования является то, предоставляет ли он достаточно информации, чтобы тестировщик мог определить вводные данные для варианта использования и ожидаемый результат использования.

Такая информация требуется тестировщику для написания тестового примера. Вы также можете протестировать варианты использования, чтобы определить, представляют ли они различную степень риска.

Можно многое выиграть, превратив проект в процесс, который позволяет постоянно улучшать понимание дизайна продукта и способов его реализации. Тот факт, что вы разрабатываете продукт в виде набора полос, а затем последовательно создаете каждую полосу, делает процесс разработки итеративным. Тот факт, что вы проводите тестирование и анализ рисков на каждой итерации, делает процесс разработки поэтапным. Важно, чтобы вы приступили к разработке, используя дизайн, который дает вам четкое представление о том, как и когда вы сможете реализовать функциональность, указанную в требованиях. По ходу работы, если вы работаете поэтапно и итеративно в направлении полной реализации функциональности, вы

находитесь в отличном положении для управления такими вещами, как изменения требований, проблемы с дизайном и сбои в графике разработки.

Если вы работаете в организации, достигшей определенного уровня зрелости, есть вероятность, что кто-то создал документ, в котором отражены соглашения по кодированию, согласованные группой разработчиков. Если это не так, команде следует потратить время на создание набора стандартов кодирования. В этой работе должны участвовать все члены команды.

Разработка стандартов кодирования обычно предполагает проведение группового совещания и проработку списка стандартных вопросов, связанных с кодированием. Интенсивность обмена мнениями может быть поразительной, особенно когда кто-то пренебрежительно отзывается об одной из ваших любимых практик.

Поскольку разработка стандартов кодирования может быть сложной задачей, это мероприятие дает членам команды возможность внедрить на ранней стадии проекта методы проведения командных проверок. Одной из практических мер является создание совета арбитров, в состав которого входят трое самых высокопоставленных разработчиков. Обсуждение может продолжаться до тех пор, пока у кого-то есть что сказать, но после обсуждения арбитры должны провести голосование, чтобы определить, будет ли принят стандарт или отложен в сторону.

Несколько хороших книг содержат рекомендации по кодированию, и один из подходов к уменьшению противоречий заключается в принятии стандартов, представленных в одной из этих книг. Это может, по крайней мере, послужить отправной точкой.

Рекомендуется опубликовать в Интернете документ, в котором вы перечисляете правила программирования, принятые вашей командой. Вы также можете назначить администратора, которому можно будет отправлять комментарии. Позже в ходе проекта, если позволит время, кому-то может быть предложено предоставить примеры кода для иллюстрации принятого использования.

Некоторые подходы к разработке программного обеспечения предполагают, что до двух третей времени, затрачиваемого на разработку проекта, отводится на формирование концепции продукта.

Как правило, чем больше времени вы тратите на планирование внедрения, тем меньше времени вы тратите на реализацию. Причины этого становятся очевидными после небольшого размышления. Рассмотрим, например, что произойдет, если вы начнете внедрение без планирования и захотите втиснуть огромное количество операций в один класс. Затем вы разрабатываете несколько классов, зависящих от первого класса. Но потом вы обнаруживаете, что хотите разделить функциональность первого класса на несколько более мелких классов. В результате приходится переписывать большое количество кода. Планирование устранило бы необходимость в переписывании и сократило бы общее время, необходимое для реализации. Планирование включает в себя создание документов, списков, диаграмм и обсуждений. По этой причине создание документов сводится к определению типов мышления и постановки вопросов, которые характеризуют разработку продукта. Документ представляет собой повод для размышлений и постановки вопросов. Работа с шаблоном, написание отрывков и добавление диаграмм во многом второстепенны. Основная работа заключается в формировании идей и планировании мероприятий по разработке.

Одной из основных целей документа с требованиями к программному обеспечению является определение, как внутреннего, так и внешнего, охвата игры. Другими словами, вы должны знать, насколько интенсивными будут ваши усилия по разработке чего-то нового, и вы должны знать, какого рода игра будет создана. В качестве предварительного условия к документу о требованиях к программному обеспечению вы можете написать описание области применения.

Другой основной целью создания требований к программному обеспечению является составление списка предложений, в которых указывается, какие функциональные возможности вы должны реализовать,

чтобы охватить внутреннюю и внешнюю область применения продукта. Если вы создаете игру, функциональность программного обеспечения в лучшем случае имеет тенденцию косвенно выражать функции, которые вы видите в документе по дизайну игры. Если вы создаете набор компонентов для обнаружения коллизий или графического рендеринга, абстракция, как правило, еще более выражена. В конечном счете, именно это затрудняет разработку требований.

Когда вы разрабатываете программную систему, вы стремитесь найти способы сгруппировать функциональные возможности, установленные в спецификации требований к программному обеспечению. Класс инкапсулирует множество операций. Это один из способов сгруппировать функциональные возможности. Вы можете сгруппировать классы в виде полос. Полосы объединяет классы, которые выполняют общую задачу.

Тестировщики могут приступить к написанию тестовых примеров, как только требования будут готовы. Они также могут работать на этапе разработки требований, проверяя их на концептуальную и практическую полноту. Эти усилия распространяются и на работу по проектированию.

Тестирование — это отдельная от программирования деятельность, но она явно связана с программированием. Тестирование требует специального набора навыков. Даже если вы обладаете этими навыками, общепризнано, что вы не сможете надежно протестировать свой собственный код. Надежное тестирование требует вмешательство человека, обладающего видением, разработанным отдельно от взгляда человека, создающего код.

. Одним из итогов обсуждения стало замечание о том, что, поскольку тестировщики предоставляют особый взгляд на все аспекты разработки, у них есть претензии к требованиям и проектной документации. Их цель - оценить такие артефакты на ранней стадии, чтобы определить, являются ли они действительными и поддающимися проверке в качестве ориентиров для разработки. Помимо расширения своих знаний о программном обеспечении с помощью концептуального анализа и изучения документации, тестировщики

могут извлечь выгоду из раннего включения в проект, поскольку они могут приступить к разработке инструментов тестирования, как только поймут масштаб проекта. Во многих случаях они захотят приобрести существующие инструменты и модифицировать их для текущей работы. Команда должна предусмотреть такую деятельность в бюджете проекта с самого начала.

3.5 Выпуск проекта

В начале проекта команда должна уделить некоторое внимание плану выпуска, поскольку он может выявить зависимости, для устранения которых требуется время. К числу таких зависимостей относятся юридические контракты с поставщиками. Возможно, вам удастся приобрести библиотеку или коллекцию ресурсов практически за одну ночь, но юридические тонкости обеспечения прав на такие объекты могут потребовать месяцев переговоров. Если существуют какие-либо подобные зависимости, вам следует ожидать их с самого начала. Если план выпуска состоит из нумерованного списка элементов, добавленных к разделу плана управления конфигурацией, он по-прежнему выполняет свою задачу. Независимо от того, где формализована информация, это позволяет руководителю проекта включать связанные с ней задачи в план проекта.

Такое планирование гарантирует, что доработка плана будет завершена до начала окончательного тестирования и выпуска. По мере приближения времени, когда начнется серьезная работа по выпуску, команда может назначить менеджера по выпуску и разработать официальный план выпуска.

Споры случаются. Лучшее, что можно сделать, — это разбираться с ними по мере их возникновения и завершать их решительными действиями. Часто именно руководитель проекта должен определять, как разрешать споры. Гораздо лучше мгновенно вывести из себя нескольких членов команды своим решением, чем позволить разногласиям нарастать до тех пор, пока они не начнут отнимать значительное количество времени и энергии.

Переход к окончательному выпуску предполагает несколько изменений. Работа по разработке прекращается и начинается программирование технического обслуживания (в форме устранения дефектов). Кроме того, обязанности человека, который считается ведущим разработчиком, могут быть распределены. Например, менеджер по выпуску, который может и не быть техническим специалистом, может внезапно стать самым заметным человеком в команде. По крайней мере, временно, подобно боцману, ведущему корабль в открытое море, менеджер по выпуску должен командовать проектом

Вы должны запланировать выполнение плана выпуска в соответствии с данными о завершении разработки кода. Вы можете настроить план в соответствии с данными, которые вы собираете в ходе проекта. В плане выпуска должно быть подробно указано, какие предварительные версии вы собираетесь создавать и распространять. В нем также должно быть описано, как вы хотите взаимодействовать со своей группой клиентов. Что касается альфа- и бета-версий, то основными критериями для их выбора являются следующие:

■ Альфа-версия. Вы еще не освоились с функциями своей игры. В вашей организации есть группа игроков и тестировщиков. Вы настроили способ, с помощью которого они могут сообщать о своих выводах по ходу использования игры. Даже если функциональность не стабильна или неполна, вам интересно узнать, как она работает и привлекательна ли она. Однако на данный момент игра слишком сложна, чтобы доверять ее неопытным игрокам или пользователям. Вам нужна дружелюбная аудитория.

■ Бета-версия. Функциональность игры завершена, но все еще нуждается в тестировании. Тестировщики в вашей организации тщательно изучили игру, но вы думаете, что лучше всего провести тестирование с привлечением внешней группы, состоящей из потенциальных или реальных клиентов. Вы определили способ, с помощью которого ваши клиенты могут сообщать о своем опыте, и заключили соглашение с вашими клиентами, в

котором четко определены границы и обязательства. Вы заинтересованы в выявлении всех возможных дефектов в продукте. Только в том случае, если пользователи обнаружат особенно неприятную проблему с игрой, вы сможете рассмотреть возможность изменения ее функциональности.

Практика разработки программного обеспечения для игр предполагает, что требуется время, чтобы понять масштаб задачи, а затем разработать набор требований, которые формализуют понимание. На основе требований вы можете создать дизайн. Один из подходов к проектированию заключается в разделении функциональности на группы, называемые полосами. Полоски служат двойной цели. Они позволяют вам понять, как реализовать функциональность вашего игрового программного обеспечения, и позволяют планировать свою деятельность по разработке.

После того, как вы определитесь с проектом и распланируете свои действия по разработке, вы захотите приступить к написанию кода. Перед созданием кода у вас должен быть готов план управления конфигурацией. Этот план содержит соглашения о структурировании каталогов и именовании файлов. Кроме того, у вас должен быть определенный набор правил программирования. Чтобы обеспечить безопасное управление кодом, вам следует установить систему контроля версий. Если вы дополните эту систему соглашениями, изложенными в плане настройки, вы сможете создавать базовые параметры вашего проекта через определенные промежутки времени.

Тестирование является важным аспектом процесса разработки, и процесс тестирования должен быть в центре внимания. Начните с самого начала. Тестировщики могут разработать план тестирования на основе спецификации разработки программного обеспечения, но еще до разработки конкретных тестовых примеров они могут внести свой вклад в качество работы по выявлению требований и разработке спецификации.

4. ТЕСТИРОВАНИЕ И ОПТИМИЗАЦИЯ ИГРОВЫХ СИСТЕМ

4.1 Смысл тестирования и дефекты

Тестирование программного обеспечения представляет собой один из основных аспектов разработки программного обеспечения. Оно включает в себя исследование программного обеспечения с целью выявления его дефектов, которые бывают двух основных типов. Один тип дефектов возникает, когда вы не разработали свое программное обеспечение в соответствии со спецификацией. Другой тип дефектов возникает, когда вы разрабатываете программное обеспечение таким образом, что добавляете в него то, что не предусмотрено спецификациями.

С одной стороны, ваша игра не выполняет то, что вы от нее ожидаете. С другой стороны, ваша игра выполняет действия, которых вы от нее не ожидаете и не хотите. Чтобы устранить дефекты, вы подвергаете свою игру различным тестам. Среди этих тестов есть те, которые касаются компонентов, интеграции и системы в целом. Тестирование представляет собой огромное поле для практики и изучения.

Специалисты по тестированию программного обеспечения исследуют программное обеспечение на предмет наличия в нем дефектов. Чтобы понять смысл этого утверждения, рассмотрим, что значит найти дефект.

Что значит сказать, что вы считаете программу дефектной? Возможно, это означает, что программа не выполняет то, что вы от нее ожидаете. Но тогда что значит ожидать чего-то? Один из ответов на этот вопрос заключается в том, что, когда у вас есть ожидания, вам сказали об этом или вы интуитивно поняли пришли к пониманию того, что программное обеспечение должно вести себя определенным образом. Когда программное обеспечение ведет себя не так, как вы ожидаете, что-то не так. Программное обеспечение неисправно. Некоторые виды неправильного поведения очевидны. Другие - нет. Вы играете в игру, и внезапно она зависает, и вам приходится перезагружать

компьютер вручную. Вы пытаетесь снова, и происходит то же самое. Это считается дефектом, который почти каждый может распознать интуитивно.

С другой стороны, вы играете в игру, и кажется, что все в порядке. Однако даже хотя вы знаете, что нанесли смертельный удар по врагу, его сила необъяснимо возрастает, бросая вызов логике, намерениям и целям, которые управляют всеми остальными аспектами игры.

Этот тип дефекта может быть понятен интуитивно, но гораздо сложнее понять, как именно устранить проблему, особенно если вы не являетесь опытным игроком или программистом. Некоторые игроки могут попытаться объяснить странное поведение.

Дефект, который ищет тестировщик, был внедрен в игру в ходе разработки и представляет собой нечто, что необходимо устранить. Дефект возникает не столько из-за требований или дизайна (при условии, что они согласованы), сколько из-за неправильной реализации.

Тестировщик программного обеспечения работает на основе спецификаций программного обеспечения и создает тестовый пример. Тестовый пример определяет способ, с помощью которого тестировщик может использовать спецификации для проверки поведения программного обеспечения в строго определенном контексте. Используя информацию, полученную из спецификаций, тестировщик определяет начальные и результирующие условия для тестового примера.

Если спецификации предусматривают, что противник должен потерять работоспособность, когда при сбое, и тестировщик обнаруживает, что вместо этого Манфред восстанавливает работоспособность при сбое, программное обеспечение не проходит тестовый набор. Когда программное обеспечение не проходит тестовый набор, тестировщик отправляет отчет о дефекте программисту по техническому обслуживанию.

Тестировщики программного обеспечения отличаются от программистов тем, что они обнаруживают дефекты вместо того, чтобы вносить изменения в код для их устранения. Тестировщики применяют к

программному обеспечению формализованные тестовые примеры и, основываясь на результатах тестирования, направляют отчеты о дефектах программистам. Работая с информацией, предоставленной тестировщиками, программисты устраняют дефекты. Затем тестировщики проверяют, устранили ли программисты дефекты.

4.2 Концепции тестирования

На рисунке 1 показаны три концепции, которые являются ключевыми для тестирования. Тестирование включает в себя помимо всего прочего, понимание того, что такое проверка, валидация и исследование, и помогает понять основные принципы, которые мотивируют разработку различных подходов к тестированию.

Рисунок 1 – Три концепции тестирования

Прежде чем разработчики планируют программный продукт — игру, они определяют его в той или иной форме. К игре предъявляется ряд требований. Требования состоят из набора предложений, набора вариантов

использования или того и другого вместе. Они определяют, как должна вести себя игра. Когда вы проверяете, соответствует ли игра требованиям, вы подтверждаете, что разработчики разработали продукт в соответствии с требованиями.

Даже если разработчики разрабатывают игру таким образом, чтобы она внешне выглядела так, как будто удовлетворяет требованиям, это все равно может быть не так. Рассмотрим, например, требование о том, что при сложении двух чисел должен получаться правильный ответ. Предположим, что 2 и 4 сложены, и результатом является “cat”.

Чтобы проверить эту ситуацию, тестировщик сначала должен определить подходящие ответы. (В этом случае предположим, что требования действительно означают числа.) Возможно, в метафизическом измерении, выходящем за рамки обыденного, 2 и 4 действительно означают “кошка”. Но тестировщик знает, что ожидаются только цифры. Используя эту информацию, тестировщик разрабатывает валидационный тест. Если в результате вычислений получаются числа и если результат верен в соответствии со стандартными арифметическими расчетами, внедренное программное обеспечение является валидным.

Тестирование направлено на выявление дефектов, которые были внесены в программный продукт в процессе разработки. Проверить и исследовать предлагают наиболее прямые способы обнаружения дефектов, но у тестировщиков есть и другие подходы к тестированию. Эти тесты включают в себя не столько то, что должно быть реализовано (что, опять же, вы можете определить, изучив требования и дизайн программного обеспечения), сколько то, что не должно быть реализовано. Когда вы исследуете то, что не должно быть реализовано, вы вступаете в исследовательскую область тестирования.

Bugs становятся butterflies, потому что поисковое тестирование предполагает выяснение того, продолжает ли программное обеспечение работать так, как указано, когда к нему предъявляются требования о том,

чтобы оно вело себя не так, как указано. Тестировщики выясняют, как исправить ситуацию.

Чтобы поломать программное обеспечение требуется заставить его вести себя не так, как указано. Например, в некоторой игре, в конструкторе персонажей есть поле, позволяющее игроку ввести произвольное имя персонажа. Обычно игроки вводят имя, например, “Бабба”, “Рах” или “Слик”. Но что произойдет, если кто-то из игроков решает ввести последовательность пробелов? Для поиска ответа требуется предварительное тестирование.

Если система принимает пробелы, она сможет работать нормально. Однако в более поздний момент игры игрок увидит пустую строку в списке выбора персонажа. Когда вы нажимаете на пустую строку, появляется персонаж, имя которого указано через пробелы. Вы хотите, чтобы игра вела себя именно так?

Вопрос не в том, как вы формализуете тестирование, а в том, как вы тестируете без формализации. Когда вы тестируете без формализации, вы проводите так называемое реактивное тестирование, или тестирование методом поиска и исправления. Под это понятие подпадает значительная часть программирования. Программист набирает несколько строк кода, пытается скомпилировать или построить компоновку и наблюдает, находит ли компилятор синтаксическую ошибку.

Когда в игре используются десятки или сотни тысяч строк сложного кода, реактивное программирование больше не работает. Вместо этого тестирование должно проводиться в соответствии с основным планом. Тестировщики разрабатывают тесты, которые охватывают всю область применения продукта и исследуют как можно больше логических или операционных путей в продукте. Разработка тестов включает в себя рассмотрение этапов разработки.

Программное обеспечение начинается с определенного типа спецификации. Оно приобретает форму в процессе проектирования. Он внедряется, развертывается и поддерживается. Каждому из этих этапов

соответствует определенный тип тестирования. Например, вы можете провести поведенческое или функциональное тестирование, чтобы убедиться, что разработчики внедрили требования. Вы можете провести интеграционное тестирование, чтобы подтвердить, что модули или компоненты, созданные в соответствии с требованиями, могут быть объединены вместе. При проведении системного тестирования вы можете взять систему и посмотреть, может ли она быть установлена и работать на разных ОС компьютеров.

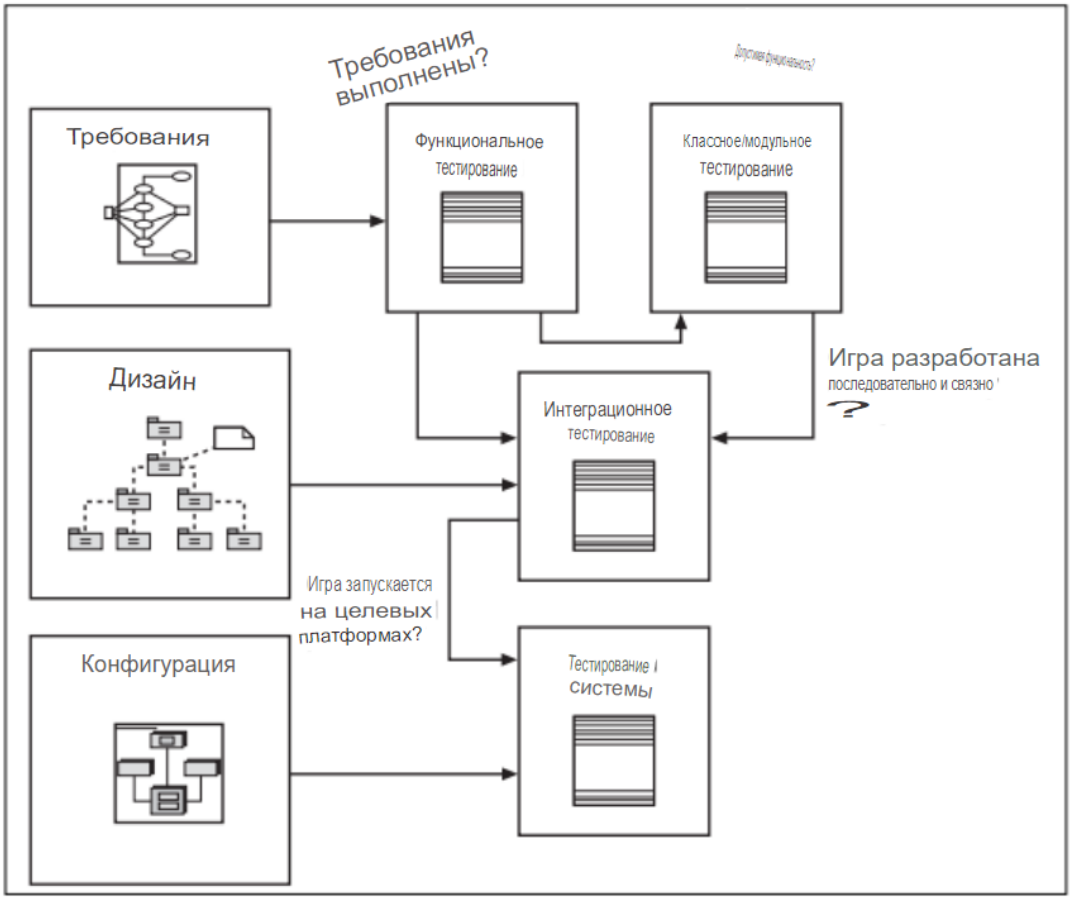

Рисунок 2 - Планы тестирования отражают этапы разработки.

Как показано на рисунке 2, вы можете объединить свои действия по тестированию в относительно небольшом количестве формальных тестов. Многие другие тесты с экзотическими названиями могут дополнять эти базовые формы тестирования, но базовые тесты остаются отправной точкой практически для любого комплексного тестирования.

4.3 Тестирование игр

Игры отличаются, по крайней мере, в одном отношении от других видов программного обеспечения, поскольку в них редко отсутствует предварительная документация. С другой стороны, игры похожи на большинство других видов программного обеспечения, поскольку в документации, на которой они основаны, часто не хватает деталей.

Когда вы официально тестируете игровой программный продукт, вам, возможно, придется потратить много времени на анализ игрового программного обеспечения, чтобы понять, как его тестировать.

Компьютерные игры, как правило, представляют собой область программного обеспечения. Эта область включает игровые жанры, типы персонажей, обучающие сценарии и множество других элементов. Как тестировщик игрового программного обеспечения, вы применяете знания об этих элементах различными способами. Среди множества возможностей, например, вы формулируете сценарии тестирования, планируете стратегии тестирования и понимаете, какую часть поведения игры должен охватывать план тестирования. Без знания предметной области может быть непросто определить, в какой степени необходимо протестировать поведение игры или какой уровень детализации требуется для конкретных тестовых случаев.

Общие знания в области разработки игр помогут вам, в частности, в решении следующих задач:

- Понимание жанра игры, что позволит вам сфокусировать тестирование на поведении, наиболее важном для жанра вашей игры
- Технические требования к тестированию
- Оценка стандартов производительности, которых должна достичь ваша игра, в свете критериев, полученных на основе знаний о других играх
- Определение эксплуатационных характеристик игры, чтобы определить, соответствуют ли они характеристикам других игр.

- Возможность легко общаться как с геймдизайнерами, так и с разработчиками игрового программного обеспечения во время тестирования.

Вы можете рассматривать прикладные знания как часть более обобщенной области знаний о компьютерных играх. Когда вы рассматриваете прикладные проблемы, ваше внимание переключается на контекст, связанный с особенностями игры, которые отражают современные технологии, воплощенные в играх.

В новейшей истории к ним относятся такие элементы, как частицы, тени, эффекты колеблющегося движения, карты неровностей, усовершенствования в обнаружении столкновений, голосовые взаимодействия в многопользовательской игре, базы данных и расширения безопасности.

В дополнение к текущим и благодаря передовым технологиям игры воплощают в себе стандартные технологии игровых приложений. К ним относятся диалоговые окна, кнопки, ползунки, макеты консоли, параметры справки и компоненты навигации. Знание предметной области приложения позволяет вам более эффективно тестировать игру одним из следующих способов:

- Разработка тестовых примеров для проверки и валидации требований
- Понимание того, какие части игры представляют собой новые технологии и заслуживают особого внимания
- Использование стандартных артефактов тестирования для решения стандартных игровых задач

Непосредственно перед выпуском игры тестировщикам приходится прилагать значительные усилия, чтобы провести приемочное тестирование игры. Приемочное тестирование программного обеспечения в целом ориентировано на заказчика. В какой-то степени это справедливо и для игрового мира, но правильнее будет сказать, что приемочное тестирование в игровом мире состоит из формальных мероприятий, которые проводят тестировщики.

Представим, что вы выбрали компанию Microsoft как клиента. Майкрософт разработала обширный список критериев, которым должны соответствовать игры, чтобы они могли быть включены в ее набор игр.

По последним подсчетам, этот список критериев насчитывает более 100 страниц. Ниже приведены некоторые перефразированные условия тестирования для Microsoft.:

- Приложение выполняет основные функции и остается стабильным во время тестирования функциональности.
- Приложение не создает временные папки и не размещает файлы в неправильных местах.
- Приложение остается стабильным, когда оно обрабатывает длинное имя файла.
- Приложение остается стабильным, когда ему предписывается использовать недоступные устройства.

После выхода игры ее характеристики как программного продукта меняются. Во-первых, жизнь игры обычно не включает в себя добавление новых функций. Совершенствование существующих функций и функциональности определяет направленность программирования и усилия по тестированию, приложенные для этого. Как правило, этот этап жизненного цикла программного продукта далеко не тривиален. В среднем 60 процентов расходов, связанных с программным продуктом, приходится на период после его выпуска.

Вы можете согласовать действия по тестированию с циклом разработки программного обеспечения. Одним из подходов к этому является модель “V”. Модель “V” начинается с требований и продвигается вниз, по левой части буквы “V", через проектирование и детальную проработку. Затем все начинается снизу вверх, справа от буквы “V”, с реализации классов, функций, компонентов и, наконец, всей системы в целом.

Рисунок 3 иллюстрирует последовательность действий, связанных с областями, рассмотренными ранее

Рисунок 3 – Модель V

Тестирование включает в себя нечто большее, чем просто участие в игре. Участие в игре составляет значительную часть работы, связанной с тестированием; однако создаваемые вами тесты часто ориентированы на реализацию, а не на игровой процесс. Вы можете рассматривать опыт игры как общую картину, работая на уровне разработки программного обеспечения, вы концентрируетесь на фрагментарном представлении об игре, которое включает в себя поиск дефектов в том, как было реализовано поведение.

По этой причине вы можете склоняться к просмотру игра — это система, состоящая из компонентов, модулей и рабочих характеристик, а не то, во что вы играете.

Тестирование играбельности и удобства использования отличается от тестирования, которое фокусируется на классах, интеграции, компонентах и системах. Тесты на играбельность и удобство использования напрямую касаются того, как игроки взаимодействуют с видимыми функциями игры. Они также затрагивают эстетические качества игры.

Инспекция включает в себя изучение элементов, которые обычно относятся к категории документации. Вы используете инспекцию в качестве основного способа тестирования, когда у вас есть возможность приступить к тестированию в начале проекта. Это хороший способ работы, но, к сожалению, многие группы разработчиков не утруждают себя привлечением тестировщиков до относительно поздней стадии проекта. Вот некоторые основные моменты проверок:

- Требования. Протестируйте требования, чтобы определить, являются ли они избыточными, плохо сформулированными или не отвечают требованиям
- Варианты использования. Помощь в разработке вариантов использования для требований. Определите, соответствуют ли эти варианты требованиям. Убедитесь, что были включены условия запуска и завершения. Если сценарий использования слишком длинный, предложите разбить его на два или более вариантов использования.
- Дизайн. Ознакомьтесь с описанием проекта программного обеспечения. Если в команде разработчиков работают Диаграммы UML, проверьте их на точность и полноту.
- Конфигурация. Ознакомьтесь с планом управления конфигурацией, чтобы определить, обеспечивают ли соглашения о кодировании, политики конфигурации и другие подобные элементы всеобъемлющую основу для разработки.
- Документация. Проверьте документы, чтобы убедиться, что они правильно идентифицированы и пронумерованы.

В каждом случае вы можете протестировать документацию и другие проверяемые артефакты, используя те же инструменты, которые вы используете для проверки кода. Другими словами, создайте тест. Если тест выявит проблему, сообщите о ней.

4.4 Модульное тестирование

Традиционным термином, обозначающим тестирование, которое включает в себя изучение мельчайших компонентов программного продукта, является модульное тестирование. Модульное тестирование чаще всего определяет деятельность тестировщиков по мере их работы над внедрением. Модульное тестирование может включать в себя практически любую часть программного продукта, прежде чем она будет интегрирована в целостную систему.

Целью тестирования на этом уровне является изолированное изучение поведения, прежде чем оно будет объединено с другим поведением и, как следствие, станет сложным.

Другим термином, обозначающим модульное тестирование, является классовое тестирование. Эти термины взаимозаменяемы, поскольку оба относятся к наименьшим компонентам программной системы. При объектно-ориентированной разработке наименьшей единицей является класс. При процедурной разработке наименьшей единицей является функция. Модульное тестирование включает в себя большой набор инструментов. Модульное (или классовое) тестирование является основным средством, с помощью которого вы можете протестировать функциональные требования. Обратите внимание на следующее:

■ План тестирования компонентов. Вы можете использовать план тестирования компонентов, чтобы подробно описать большинство процедур тестирования, которые вы собираетесь выполнить для проверки и валидации реализации функциональных возможностей, указанных в требованиях.

■ Тестирование с помощью черного ящика. Вы можете использовать тестирование с помощью черного ящика для проверки поведения компонентов. Объект класса — это компонент, который получает информацию о параметрах и возвращает результат. Вы можете создать тестовый пример на основе этих двух точек взаимодействия.

■ Тестирование в режиме "белого ящика". Диаграммы перехода состояний иллюстрируют работу тестирования в режиме "белого ящика". Такие диаграммы позволяют понять, как изменяются атрибуты объектов по мере того, как они выполняют свою работу. Тестирование в режиме "белого ящика" предлагает мощный аналитический инструмент для оценки надежности компонентов на самом элементарном уровне.

Интеграционное тестирование включает в себя создание тестовых процедур для выявления дефектов, возникающих при сборке разработчиками компонентов, созданных на уровне классов, в модули. Взаимодействие между несколькими классами обычно характеризует срок службы модуля. Когда два объекта обмениваются сообщениями, сообщение, которое первый (клиентский) объект передает второму (серверному) объекту, может вызвать проблемы. Среди распространенных проблем - проблемы, связанные с неправильным созданием экземпляров серверных объектов. Неправильное создание экземпляров серверных объектов приводит к сбою в работе клиентского объекта. Однако обнаруженная вами ошибка может указывать только на сбой в работе клиента. Необходимо проанализировать взаимодействие собранных объектов, чтобы определить реальный источник сбоя.

Планирование интеграционного тестирования требует от вас понимания структуры программной системы. По этой причине наиболее эффективно создать план интеграционного тестирования можно после завершения разработки спецификации программного обеспечения.

Рассмотрим, что происходит, когда игрок устанавливает игру. Установка завершается успешно, если игрок может начать играть в игру сразу после завершения установки. В противном случае она завершается неудачей. Системное тестирование включает в себя изучение поведения программного обеспечения как целостной системы. Его цели в основном заключаются в том, чтобы игрок воспринимал игру только как опыт, который начинается и заканчивается тем, что происходит в игре.

Контекст системного тестирования включает в себя использование систем, которые представляют те, которые вам нравятся. Предполагается сборка готового программного обеспечения в изолированный объект. Для эффективного тестирования этого объекта сначала необходимо удалить зависимости. Для этого тестировщики создают среду тестирования системы. Такая среда обычно представляет собой простую операционную систему, установленную на компьютере, который вы считаете типичным для своих клиентов. В большинстве случаев при разработке для тестирования систем используется несколько сред, аналогичных тем, которые используются предполагаемыми клиентами.

План тестирования системы состоит из спецификаций целевых операционных систем и ПК. Он также предоставляет тестовые примеры, которые требуют от вас выполнения установки. План, на более детальных уровнях может устанавливать критерии для стресс-тестов.

Стресс-тест может включать, например, проверку того, что произойдет, если в системе, на которую вы устанавливаете свою игру, недостаточно доступной оперативной памяти. Как в таком случае будет работать игра? Другой тест связан с системным сбоем. Например, если произойдет отключение питания, сколько данных будет потеряно?

Как и в случае с тестами компонентов и интеграции, это эффективный подход к разработке системных тестов включает в себя создание вариантов использования. Вы можете использовать общие сценарии установки или стрессовые условия в качестве основы для разработки вариантов использования.

Когда вы планируете тестовые мероприятия, вы создаете планы тестирования, которые учитывают как различные подходы к тестированию, так и различные аспекты тестируемой системы. Вы ставите для себя эти два приоритета, потому что ни одного вида тестирования недостаточно для тестирования всех аспектов системы.

Аналогичным образом, лучший способ протестировать различные аспекты системы — это использовать различные подходы к тестированию. Чтобы подтвердить выполнение требований, например, поведенческое тестирование позволяет настроить тесты на основе вариантов использования, указанных в спецификации требований к программному обеспечению. для изучения проблем производительности и сложности вы можете использовать структурное тестирование.

Планы тестирования позволяют вам продумать подходы, которые вы хотите использовать, и определить, как вы хотите анализировать систему для тестирования.

ЗАКЛЮЧЕНИЕ

Таким образом, в ходе написания реферата были изучены основы разработки игр со стороны программной инженерии. Были рассмотрены такие области, как формулирование идеи игры, разработка концепта, составление документации, распределение обязанностей в команде, составление модульных тестов, модель тестирования «V». Раскрыта специфика разработки игровых приложений

Были изучены такие игровые движки, как unreal engine, unity и godot engine. Раскрыты их особенности в сравнении друг с другом и даны рекомендации использования различных игровых движков для различных задач.

Раскрыты нюансы сопровождения продукта после выпуска его различных версий (альфа и бета). Также мы рассмотрели понятие дефекта и его виды.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. John P. Flynt Software Engineering for Game Developers [Текст] / John P. Flynt — 1-е изд.. — : , 2005 — 890 с.

2. Luiz F. C. Game development software engineering process life cycle: a systematic review / Luiz F. C. [Электронный ресурс] // SpringerOpen : [сайт]. — URL: https://jserd.springeropen.com/articles/10.1186/s40411-016-0032-7 (дата обращения: 12.06.2024).

3. Understanding the Role of a Video Game Software Engineer / [Электронный ресурс] // INSTITUTE OF DATA : [сайт]. — URL: https://www.institutedata.com/blog/role-of-video-game-software-engineer/ (дата обращения: 12.06.2024).

4. Что такое программирование игр и как стать игровым программистом? / [Электронный ресурс] // Хабр : [сайт]. — URL: https://habr.com/ru/articles/819723/ (дата обращения: 12.06.2024).

5. Липаев, В. В. ПРОГРАММНАЯ ИНЖЕНЕРИЯ СЛОЖНЫХ ЗАКАЗНЫХ ПРОГРАММНЫХ ПРОДУКТОВ [Текст] / В. В. Липаев — 1-е изд.. — Москва: Макс Пресс, 2014 — 311 с.

6. Зеленко Л. С. Программная инженерия [Текст] / Зеленко Л. С. — 1-е изд.. — Самара: Макс Пресс, 2012 — 263 с.