

# **ПРОГРАММНАЯ ИНЖЕНЕРИЯ ДЛЯ ИГР**

ТЕХНОЛОГИИ, ИНСТРУМЕНТЫ И МЕТОДЫ  
РАЗРАБОТКИ

# ВВЕДЕНИЕ В ИГРОВУЮ ИНДУСТРИЮ

Любой разработчик игр может извлечь пользу из знаний в области разработки ПО, рассматривая её как набор инструментов, из которых можно выбирать. Понимание доступных инструментов и их применения позволяет усовершенствовать процесс разработки. Изучение этих тем открывает множество других аспектов:

- 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, применимым в любой сфере

Разработка ПО касается всего, что связано с созданием программного обеспечения, улучшая этот процесс и открывая новые горизонты в культуре разработки игр.

**dev team**



# КОМАНДА

В любом игровом проекте участвуют многие творческие специалисты: дизайнеры, художники, музыканты, сценаристы и др. Разработка ПО, хотя и важна, является лишь частью общей картины. Программное обеспечение помогает внедрять проверенные методы, удовлетворяющие потребности команды.

# КОМАНДА

A background image showing two men in business suits shaking hands. The man on the left is holding a large model of a rocket. The man on the right is holding a folder. The image is dimmed and serves as a background for the text.

Пример взаимодействия: Геймдизайнер и разработчик ПО

Геймдизайнер создает концепцию игрового мира и зависит от разработчиков, реализующих функциональность. Разработчики ПО должны понимать требования геймдизайнера для точного выполнения задачи.


Принципы программной инженерии:

- Тщательное понимание требований к системе.
- Создание дизайна, соответствующего требованиям.
- Проверка, чтобы дизайн и реализация соответствовали ожиданиям.

Преимущества для команды:

- Эффективное использование времени дизайнера.
- Минимизация избыточности и переделок.
- Знание, как задавать правильные вопросы и использовать информацию.

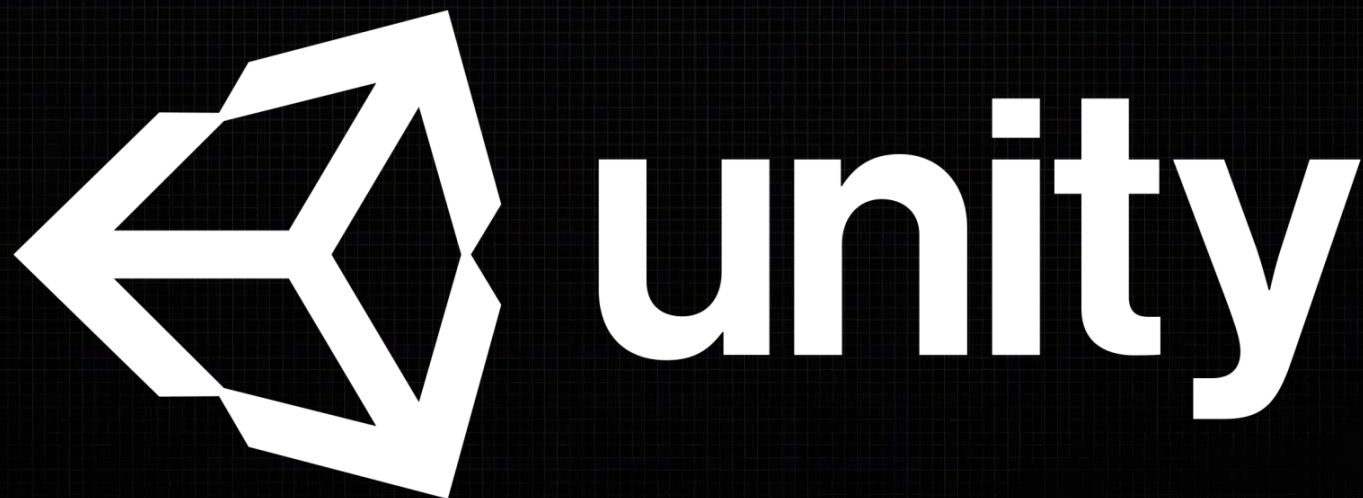


A decorative wavy line in yellow and white on the left side of the image.

# **ИГРОВЫЕ ДВИЖКИ И ИХ ОСОБЕННОСТИ**

# UNITY

В современном мире игры стали настолько сложными, что их практически никто не создает с нуля. Обычно используется фреймворк или движок, чтобы упростить и ускорить процесс разработки. Одним из таких инструментов является Unity, межплатформенная среда для разработки игр, работающая более чем на 20 различных операционных системах, включая настольные компьютеры, игровые консоли, мобильные устройства и Интернет-приложения.





## ПРЕИМУЩЕСТВА UNITY

Unity делает разработку игр доступной для всех, включая студентов, без больших финансовых затрат.

Unity поддерживает разработку для нескольких платформ, что делает его идеальным для создания игр, работающих на разных устройствах.

Использование Unity экономит время, так как не нужно разрабатывать собственный движок с нуля.

Unity предоставляет мощные инструменты для создания игр и управления контентом.



# ИСПОЛЬЗОВАНИЕ UNITY

## КОГДА UNITY ПОДХОДИТ:

- Создание игр для нескольких устройств.
- Высокая скорость разработки.
- Необходимость в полном наборе функций без создания собственного инструментария.

## КОГДА UNITY НЕ ПОДХОДИТ:

- Игры, не требующие частой перерисовки сцены.
- Когда требуется точное управление действиями движка на низком уровне без лицензии на исходный код.





# UNREAL ENGINE

Unreal Engine — мощный движок от Epic Games для создания игр и интерактивных приложений. Он поддерживает платформы ПК, консоли, мобильные устройства и VR.

Выпущенный в 1998 году, движок постоянно обновляется и совершенствуется.

# ПРЕИМУЩЕСТВА UNREAL ENGINE

- + Поддержка трассировки лучей (ray tracing) для фотореалистичного изображения.
- + Система Blueprints позволяет создавать логику без кода.
- + Возможность глубокой модификации движка на C++.
- + Множество готовых моделей, анимаций и текстур.
- + Инструменты для работы с анимацией, физикой, ИИ и сетевыми возможностями.
- + Поддержка множества платформ.





## СРАВНЕНИЕ С UNITY

Unreal Engine подходит для AAA-игр с высококачественной графикой, Unity — для средних, малых и 2D-игр.

Unreal Engine использует Blueprints и C++, Unity — C# и визуальный скриптинг Bolt.

Unity проще для начинающих и имеет больше учебных материалов. Unreal сложнее из-за C++.

Unreal Engine — для сложных, графически насыщенных проектов. Unity — для мобильных приложений, 2D-игр и проектов с меньшим бюджетом и временем разработки.



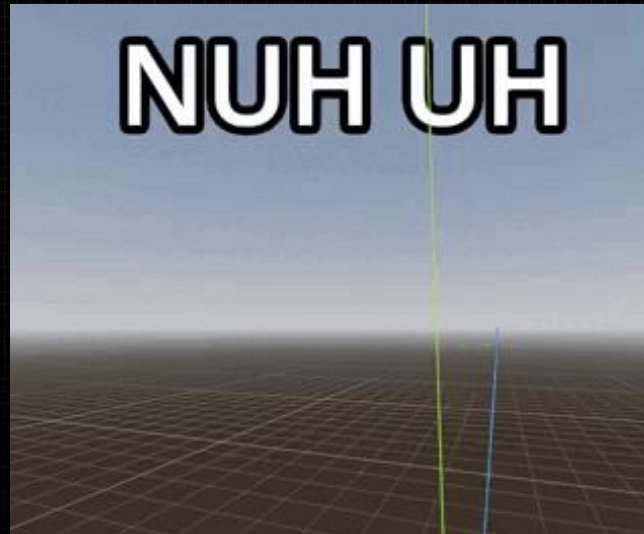
# GODOT ENGINE




Godot Engine — мощный и гибкий движок для разработки игр с открытым исходным кодом, доступный бесплатно. Его интуитивно понятный интерфейс и поддержка 2D и 3D графики делают его подходящим для начинающих разработчиков. Уникальная система сцен упрощает управление сложными проектами, а основной язык программирования GDScript, прост в освоении. Godot поддерживает множество платформ и легко интегрируется с системами управления версиями.

# GODOT ENGINE В СРАВНЕНИИ

В сравнении с Unreal Engine, известным своими передовыми графическими возможностями и использованием C++, Godot менее требователен к ресурсам и проще для новичков. Unity, также поддерживающий 2D и 3D графику, является универсальным инструментом с большим сообществом и обилием учебных материалов, но имеет бесплатную версию с ограничениями. Выбор между Godot, Unreal Engine и Unity зависит от потребностей проекта: Godot подходит для инди-разработчиков, Unreal — для проектов с высокими требованиями к графике, а Unity — для разнообразных проектов, включая мобильные и 2D игры.





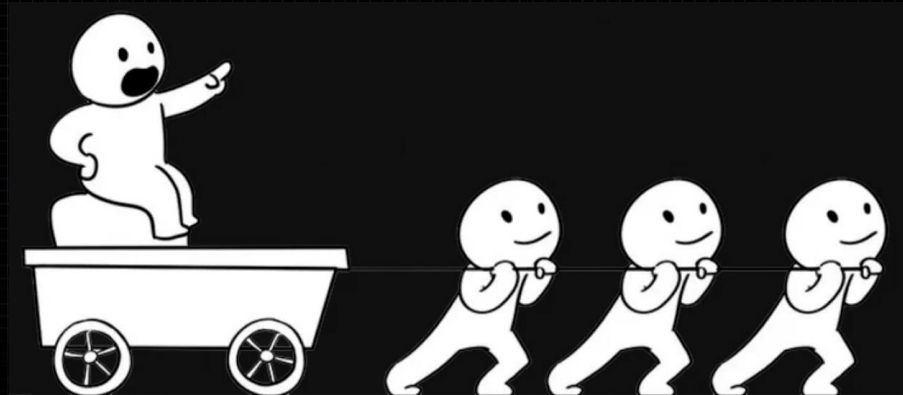


# **СПЕЦИФИКА РАЗРАБОТКИ ИГРОВЫХ ПРИЛОЖЕНИЙ**



# НАЧАЛО РАЗРАБОТКИ

Разработка игр начинается с директивы, исходящей от внешнего источника. Дизайн-документ игры является её выражением и определяет масштаб работы. Со временем специализация растет, и инженеры-программисты создают программное обеспечение для поддержания функций, разработанных геймдизайнерами. Разработка программного обеспечения начинается с контрактов, документации по дизайну и производственного плана. Руководитель проекта оценивает необходимые усилия и формирует команду для выполнения различных задач.



# СОЗДАНИЕ КОНЦЕПТА

Команда должна разработать программное обеспечение, учитывая требования, вытекающие из дизайн-документа игры, и планировать проект, определяя бюджет, сроки и необходимое количество сотрудников. Тестирование играет ключевую роль — для этого обычно привлекается специалист по тестированию, который работает с основным разработчиком над выявлением и устранением ошибок.

Описание области применения и спецификация требований помогают команде понять проект и определить, что именно должно быть разработано. Управление процессом разработки включает в себя согласование дизайна и документации для обеспечения систематического подхода. Центральным документом в этом процессе становится спецификация на разработку программного обеспечения.

Разработка проектной документации требует непрерывных итераций, чтобы адаптировать документ к изменениям проекта.





# ОРГАНИЗАЦИЯ РАБОТЫ

Команда должна настроить систему управления версиями кода на ранних этапах разработки, чтобы подготовиться к созданию прототипов и изучению концепций дизайна через код. Важно создать формальные версии этого кода, которые будут использоваться в дальнейшем проекте.

Настройка системы управления версиями должна соответствовать плану настройки, который включает определение структуры каталогов и группировку компонентов. Документ по разработке программного обеспечения служит основой для начала этого процесса.

План настройки может начинаться как документ с несколькими разделами, который следует озаглавить и поместить под контроль версий. Важно назначить номер версии прямо на титульной странице плана.



# ВЫПУСК ПРОЕКТА

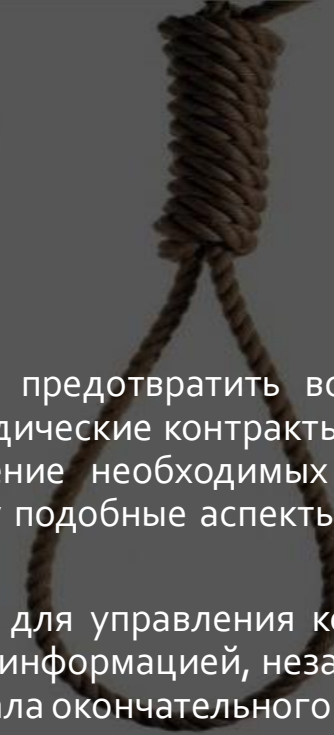
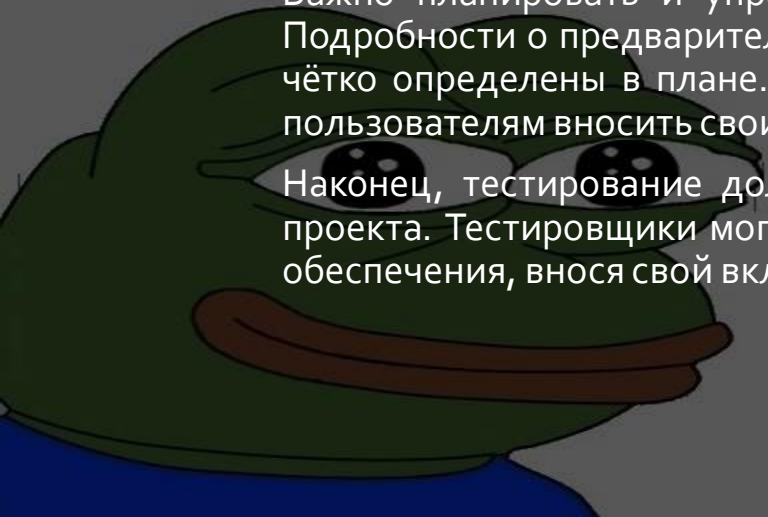
В начале проекта важно уделить внимание плану выпуска, чтобы предотвратить возникновение зависимостей, требующих дополнительного времени для разрешения, таких как юридические контракты с поставщиками. Даже если приобретение ресурсов может быть быстрым процессом, оформление необходимых прав и соглашений может потребовать значительных усилий и времени на переговоры. Поэтому подобные аспекты следует учитывать с самого начала проекта.


План выпуска, включающий нумерованный список элементов, важен для управления конфигурацией проекта. Этот план позволяет включать в него задачи, связанные с соответствующей информацией, независимо от её формализации. Такой подход гарантирует, что план разработки будет завершён до начала окончательного тестирования и выпуска.

Переход к окончательному выпуску включает изменения в задачах, таких как начало работы по техническому обслуживанию для исправления дефектов. Обязанности ключевых участников команды, включая менеджера по выпуску, могут быть перераспределены в соответствии с этапами проекта.

Важно планировать и управлять планом выпуска с учётом информации, собранной в ходе разработки проекта. Подробности о предварительных версиях и их распространении, а также о взаимодействии с клиентами, должны быть чётко определены в плане. Альфа- и бета-версии играют важную роль в этом процессе, позволяя тестировщикам и пользователям вносить свои отзывы и обнаруживать потенциальные проблемы с функциональностью продукта.

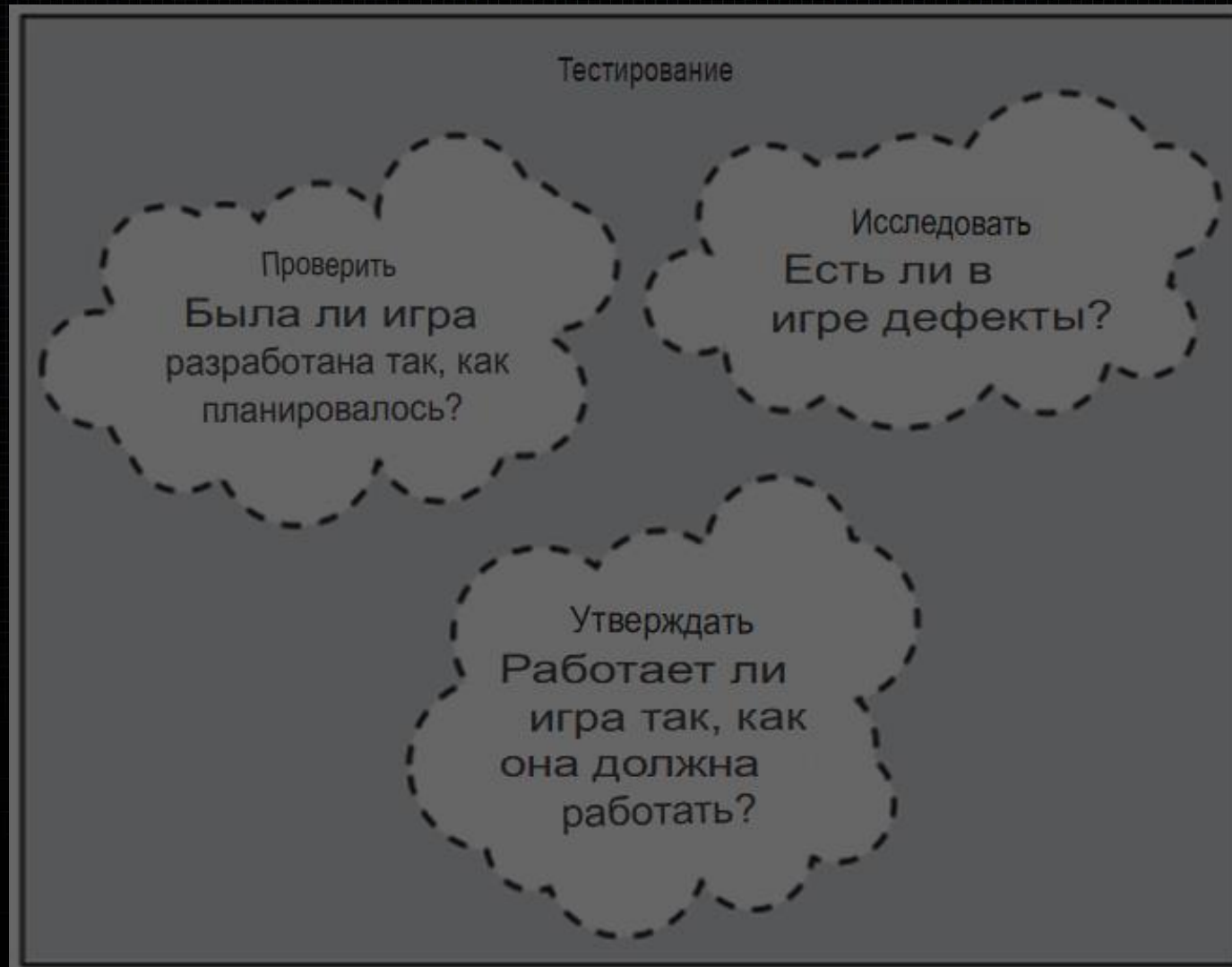
Наконец, тестирование должно быть важной составляющей всего процесса разработки, начиная с самого начала проекта. Тестировщики могут разрабатывать планы тестирования на основе спецификаций разработки программного обеспечения, внося свой вклад ещё до создания конкретных тестовых примеров.





# **ТЕСТИРОВАНИЕ И ОПТИМИЗАЦИЯ ИГРОВЫХ СИСТЕМ**

# 3 КОНЦЕПЦИИ ТЕСТИРОВАНИЯ





# КОНЦЕПЦИИ ТЕСТИРОВАНИЯ

Прежде чем разработчики начинают работу над программным продуктом, таким как игра, необходимо определить его требования. Требования могут быть представлены в виде набора предложений или вариантов использования, которые определяют ожидаемое поведение игры.

Однако даже если разработчики пытаются сделать игру такой, чтобы она казалась соответствующей требованиям, это может быть не так. Например, требование о том, что при сложении двух чисел должен получаться правильный ответ, может быть нарушено, если результатом будет строка "cat". Чтобы проверить такие ситуации, тестировщик должен разработать соответствующий валидационный тест.

Целью тестирования является выявление дефектов, которые могли быть внесены в программный продукт в процессе его разработки. Это может включать не только проверку того, что должно быть реализовано, но и исследование того, что не должно быть реализовано. Исследовательское тестирование означает, что тестировщики проверяют, как продукт ведет себя в условиях, которые не были задуманы или не предусмотрены.

$$2 + 2 = \text{cat} ?$$



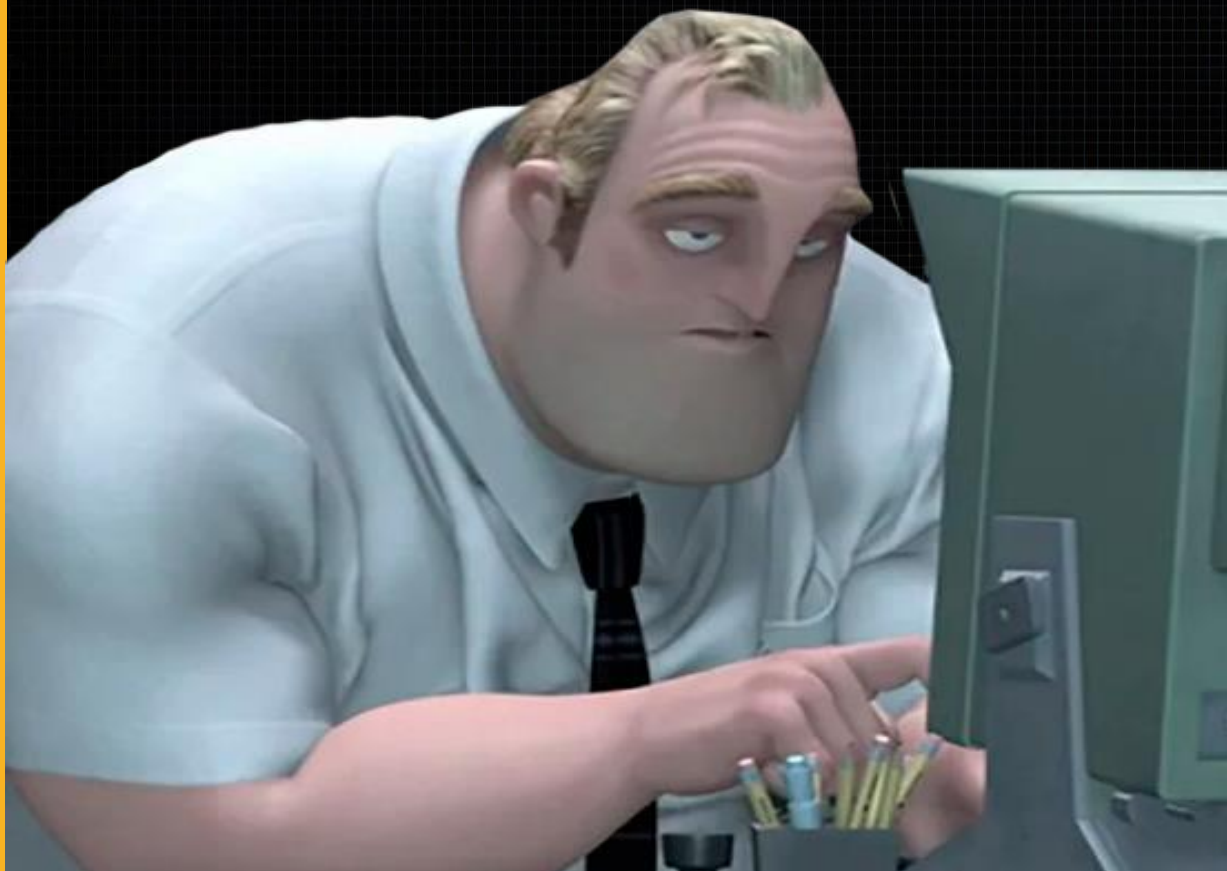
# ТЕСТИРОВАНИЕ ИГР

Игры отличаются от других видов программного обеспечения тем, что их часто разрабатывают без подробной документации. При тестировании игровых продуктов важно проводить анализ, чтобы понять, как именно их тестировать, особенно если игра представляет новый жанр или включает сложные игровые механики.

Без глубоких знаний об играх сложно определить, как тщательно тестировать и какие детали включать в тестовые случаи.

Знание жанра игры позволяет сосредоточить тестирование на ключевых аспектах, характерных для этого жанра. Взаимодействие с разработчиками и game-дизайнерами также играет ключевую роль в успешном тестировании.





# ТЕСТИРОВАНИЕ ИГР

После выпуска игры задача тестировщиков состоит в улучшении уже существующих функций и функциональности, а не в добавлении новых. Этот этап жизненного цикла продукта требует значительных усилий и часто занимает значительную часть бюджета на поддержку программного обеспечения после выпуска.

Тестирование играбельности и удобства использования отличается от тестирования на уровне классов и компонентов. Оно фокусируется на том, как игроки взаимодействуют с игрой, включая эстетические аспекты и общий игровой процесс.

Важно также инспектировать документацию и другие артефакты проекта с самого начала, чтобы устранить проблемы до поздних стадий разработки.



# V-образная модель



## V-МОДЕЛЬ

Этот рисунок иллюстрирует V-образную модель разработки программного обеспечения, которая представляет собой процесс, состоящий из этапов планирования, анализа требований, проектирования, кодирования и тестирования. Модель подчеркивает важность тестирования на каждом этапе разработки, начиная с модульного тестирования и заканчивая системным тестированием, завершаясь производством и эксплуатацией.

# МОДУЛЬНОЕ ТЕСТИРОВАНИЕ

Модульное тестирование изучает отдельные компоненты программного продукта до их интеграции в систему целиком. Целью такого тестирования является проверка поведения компонентов до их объединения, что помогает идентифицировать и изолировать дефекты.

Модульное тестирование включает использование различных инструментов для проверки функциональных требований. Основные аспекты модульного (или классового) тестирования включают планирование компонентного тестирования и использование тестирования черного ящика. Диаграммы перехода состояний применяются для анализа взаимодействия объектов на уровне "белого ящика". Интеграционное тестирование проверяет взаимодействие между классами и модулями. Системное тестирование оценивает поведение программы как единого целого.



**СПАСИБО ЗА  
ВНИМАНИЕ**