Содержание

[ВВЕДЕНИЕ 4](#_Toc19528032)

[1 ОБЩИЕ СВЕДЕНИЯ ОБ ОРГАНИЗАЦИИ ООО «ОПЕНМАЙГЕЙМ» 5](#_Toc19528033)

[1.1 О компании ООО «ОпенМайГейм» 5](#_Toc19528034)

[1.2 Ключевые факторы развития игровой индустрии 6](#_Toc19528035)

[2 ПРОЦЕССЫ ТЕСТИРОВАНИЯ И РАЗРАБОТКА ТЕСТОВОЙ ДОКУМЕНТАЦИИ 12](#_Toc19528036)

[2.1 Индивидуальное задание от компании ООО «ОпенМайГейм» 12](#_Toc19528037)

[2.2 Модели разработки программного обеспечения 12](#_Toc19528038)

[2.3 Жизненный цикл тестирования 18](#_Toc19528039)

[2.3 Требования к программному обеспечению 18](#_Toc19528040)

[2.4 Описание тест-кейса и дефекта 20](#_Toc19528041)

[2.5 Покрытие продукта тестами и их прогон 23](#_Toc19528042)

[ЗАКЛЮЧЕНИЕ 26](#_Toc19528043)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 27](#_Toc19528044)

[ПРИЛОЖЕНИЕ А. Охрана труда в организации и техника безопасности на рабочем месте программиста 28](#_Toc19528045)

[ПРИЛОЖЕНИЕ Б. Должностная инструкция инженера-программист 30](#_Toc19528046)

# ВВЕДЕНИЕ

Практика на производстве дает возможность будущему специалисту ознакомиться со специальным оборудованием, программным обеспечением, применяемыми при разработке, сопровождении и эксплуатации информационных систем.

В ходе ее прохождения студент получает углубление и закрепления знаний и профессиональных навыков, полученных в процессе обучения на основе изучения практических ситуаций.

Для прохождения технологической практики была направлена в организацию ООО «ОпенМайГейм». По прибытию в организацию передо мной были поставлены следующие задачи:

* ознакомление с организацией: структура, основные управленческие и производственные функции подразделения;
* непосредственное участие в текущей деятельности компании;
* разработка рабочего процесса в мобильном геймдеве.

# ОБЩИЕ СВЕДЕНИЯ ОБ ОРГАНИЗАЦИИ ООО «ОПЕНМАЙГЕЙМ»

## О компании ООО «ОпенМайГейм»

Общество с ограниченной ответственностью ОпенМайГейм является продуктовой компанией, деятельность которой направлена на создание собственных проектов – мобильных игр развивающей и развлекательной направленности.

На протяжении последних 7 лет многие члены команды активно занимаются разработкой программного обеспечения (ПО) и игр, в частности разработкой под мобильные устройства.

Деятельность сфокусирована на мобильном рынке, который показывает отличную динамику роста. Разработка ведется под две самые популярные платформы – *Android* и *Apple* *iOS*, используя среду для кроссплатформенной разработки *Unity*. В качестве исследования сейчас также ведется разработка на движке *Defold* для игр с моментальным запуском.

Инновационность деятельности заключается в создании собственных продуктов в сфере программного обеспечения в игровой индустрии, которые решают потребности и реализуют запросы большого числа пользователей со всего мира, следовательно, деятельность имеет крайне высокий экспортный потенциал, на текущий момент 100% выручки от реализации услуг приходится на экспорт. Самостоятельная работа команды над собственными проектами показала, что у продуктов есть перспектива и большой потенциал, суммарное количество установок игр превысило 20 миллионов.

Партнерами компании ОпенМайГейм как продуктовой фирмы в мобильной сфере являются такие гиганты, как *Google* и *Apple*. На их платформе и с помощью их инструментов производится доступ к клиентской базе, продажа виртуальных товаров и маркетинг. Налажена работа с большим количеством рекламных сетей и агрегаторов:

* *Admob*;
* *Appodeal*;
* *Chartboost*;
* *Tapjoy*;
* *Unityads*.

Высокая степень диверсификации заработка на рекламе обеспечена большим количеством рекламодателей, что очень полезно для бизнеса.

Сбыт продукции, в терминах мобильной разработки – установки, происходит по каналам мобильных площадок – *Google* *Play* и *Apple* *AppStore*.

В продуктах компании поддерживаются основные европейские языки:

* русский;
* английский;
* французский;
* немецкий;
* испанский;
* португальский;
* итальянский.

Доступ к играм имеют все страны без ограничения. На данный момент наиболее успешные позиции у проектов на рынках России, странах Латинской Америки, Европейского Союза и Великобритании.

## Ключевые факторы развития игровой индустрии

Научный прогресс, ускоренный экономический рост и глобализация обусловили переход качества жизни человечества на новый уровень, что, в свою очередь, привело к появлению новых форм досуговой деятельности и изобилию развлекательного контента. Одной из наиболее востребованных на сегодняшний день форм развлечений стали видеоигры. В XXI в. игровая индустрия стала не только неотъемлемой частью поп-культуры и индустрии развлечений, но и двигателем технологического прогресса: дополненная реальность, 3D-моделирование и новые разработки в области робототехники и *IT*-технологий применяются как для создания нового аппаратного обеспечения, так и в медицинских, образовательных и даже в военных целях.

Тем не менее формирование ныне существующей международной игровой индустрии нельзя рассматривать без ключевых игроков на этом рынке: США, Европы и Японии. Подход японских разработчиков к созданию видеоигр в корне отличается от западных представителей индустрии, что говорит о разном не только технологическом, но и культурном подходе в работе. Этому способствовало множество факторов: от исторических, социально-экономических, политических до культурных особенностей и трансформации национального сознания японцев.

Изучение такой индустрии, как игровая, предполагает комплексный подход, однако наиболее распространенным является подход экономический. Он предполагает анализ экономических показателей и бизнес-стратегий конкурирующих на рынке компаний. Экономический подход позволяет составить общую картину, сложившуюся на рынке в определенный промежуток 204 времени, определить роль и место игровой индустрии в экономике страны на разных этапах развития общества. Рассматривая историю игровой индустрии с точки зрения технологий, мы можем проследить развитие технологий, применяемых в видеоиграх, их аппаратном и программном обеспечении, то есть качественные изменения в процессе создания игр. В то же время нельзя исключать творческую составляющую при разработке, а именно историю жанров и геймдизайна. Многие игры в качестве сюжета берут за основу реальные исторические события (Гражданская война в США, эпоха Троецарствия в Китае), литературные произведения (*TheWitcher*, *TheLordoftheRings*, серия романов о Гарри Поттере). Продукция разрабатывается с учетом вкусовых предпочтений и интересов аудитории, что обусловливает тесную взаимосвязь популярной культуры в конкретной стране и мире. В свою очередь, уровень культуры и ее содержание соответствует определенной социокультурной ситуации в стране, на основании чего мы можем анализировать историю игровой индустрии с точки зрения истории культуры и современной истории отдельно взятого общества [1, c. 167].

Формирование игровой индустрии происходило совместно с изменениями, пришедшимися на вторую половину XX в. Здесь в первую очередь следует говорить о всеобъемлющем влиянии США и послевоенной оккупации не только на политическую и экономическую жизнь, но и на культуру. Контроль и финансовая поддержка со стороны американского правительства стали фундаментом для экономического чуда, что в свою очередь привело к потребительскому буму, в том числе на товары и услуги в сфере развлечений.

Первые японские разработчики аппаратного и программного обеспечения опирались на уже существующие концепции и разработки американских коллег, однако более поздний старт и сложившаяся социально-экономическая ситуация предопределили особый путь развития игровой индустрии в Японии.

В отличие от США, где началом игровой индустрии принято считать выход легендарной игры *Spacewar* в 1961 г., в Японии первый этап охватывает 205 период с момента рождения отрасли в конце 1970-х гг. и до 1980 г. Это объясняется личной инициативой японских компаний и отдельных разработчиков, в то время как в США создание видеоигр проводилось на базе университетов, научных лабораторий при участии военных. Основной движущей силой развития технологий в игровой индустрии Японии стали предприятия электротехники и бытовой электроники, а также компании по производству игрушек для детей [2, c. 48].

Наиболее успешным примером стала компания *Nintendo*, основанная в 1889 г., перейдя от производства простых игрушек и игральных карт к разработке инновационных на тот момент электронных игрушек. С 1964 г. для разработки независимого программного и аппаратного обеспечения *Nintendo* провела совместные исследования с крупнейшими производителями бытовой электроники *Sharp* и *Mitsubishi* *Electric*. Первым успехом стало получение лицензии на производство игровой домашней приставки *Odyssey* от американской компании *Magnavox* в 1975 г. Инженеры, которые изначально были наняты для разработки подающих машин для бейсбола и игрушечных раций, впоследствии были перенаправлены в отдел по разработке аркадных игр, что показывает еще одну специфическую черту японской индустрии видеоигр – преемственность кадров из смежных отраслей, например, таких, как мультипликационные студии, сфера *IT* и т. д. [3].

После так называемого шока Атари в 1983 г. рынок игровых консолей, прежде всего американский, настиг кризис перенасыщения. Это стало следствием относительно невысокого качества игр для популярных на тот момент консолей (*Atari* 5200, *ColecoVision*, *MagnavoxOdyssey* и т. д.). Многие компании раскрыли спецификацию оборудования, чтобы увеличить количество игр, выпускаемых для своих приставок, что, в конечном счете, и привело к потере контроля над качеством выпускаемой продукции. Несмотря на то что первая самостоятельная игровая приставка компании *NintendoEntertainmentSystem* (*Famicom*) вышла только в 1983 г. и уже тогда была технически устаревшей в сравнении с продукцией конкурентов, *Nintendo* 206 пошла другим путем, сделав ставку на приемлемую цену и обеспечив качественно новое разнообразие игр для своей консоли, и заняла 90 % японского рынка.

С выходом таких игр, как *DonkeyKong* (1981 г.) и *Pac*-*Man* (1981 г.), начинается второй этап. В отличие от предыдущих, более примитивных игр, где игрок управлял каким-либо предметом (бита, оружие и т. д.) или транспортным средством (автомобиль, космический корабль), *DonkeyKong* и *Pac*-*Man* стали первыми играми, в которых появился прототип главного героя, имеющего образ человека или животного [2, с. 293]. Это стало отправной точкой у разработчиков во всем мире для изменения сюжетной структуры и геймплея. Переход японских разработчиков к персонализации и более глубокому развитию сюжетных линий связан с влиянием поп-культуры Японии (аниме и манга). Этому способствовали и технические факторы: изобразить персонажей из комиксов и мультфильмов намного проще, чем реалистичный прототип из фильма.

Массовое признание комиксов и анимационных фильмов как форм развлечения для детей и взрослых, а также переход высококвалифицированных кадров (художников и сценаристов) из анимации в видеоигры стали ключевыми факторами более раннего начала второго этапа в истории игровой индустрии Японии (по сравнению с США, где второй этап в связи с кризисом начался в 1985 г., с выходом *NintendoEntertainmentSystem* на американский рынок) [1]. Дальнейший успех таких крупных проектов от *Nintendo*, как *Mario* и *Pokemon*, породил новую волну конкуренции не только в самой Японии, но и в среде западных разработчиков. Помимо технической гонки появление *Mario* стало своеобразным переходом игр на абсолютно новый уровень: если раньше они были связаны в основном с космосом, спортом, то после выхода *Mario* в 1983 г. концепция сменилась на более игривую и юмористическую, что стало еще одним вкладом *Nintendo* не только в техническую, но и в содержательную часть создания видеоигр и их жанровое разнообразие.

Конъюнктура как мирового, так и японского рынка позволила расширить потенциальную аудиторию. Если раньше продукция разрабатывалась в 207 основном для детей и молодежи, то привлечение более взрослой аудитории посредством разработки более серьезной категории игр (маджонг, го, спортивные симуляторы) стало для японской индустрии важнейшим вектором в дальнейшем развитии. Старение населения, низкий уровень рождаемости и увеличение численности рабочего сегмента потребителей, которые не в состоянии уделять время для игр на персональных компьютерах или домашних консолях, обусловило появление новой формы вовлеченности аудитории. Подобные изменения не только разнообразили внутреннюю составляющую игр, но и обновили подходы к разработке бизнес-моделей и маркетинговых стратегий.

Видеоигры трансформировались в жизнеспособное коммерческое предприятие, поскольку место, где потребитель должен был проводить время за игрой, переместилось из научных лабораторий в общественные места с игровыми автоматами и, в конечном счете, в дом. Одновременно удалось достигнуть сразу двух целей: удовлетворить запросы относительно широкой аудитории высокопрофессиональных любителей и в то же время пользователей домашней системы на базе консоли, которые надеялись окупить стоимость приставки более длительным сроком использования, в отличие от аркадных игр, которые рассчитаны на нескольких минут. Более продолжительное использование консоли, а значит, и ее игр, позволило увеличить уровень вовлеченности игрока.

В 1990-х гг. с появлением *CD*-*ROM* для персональных компьютеров и консолей, включая *SegaMegaCD* в 1993 г., *PlayStation* и *SegaSaturn* в 1995 г., начинается третий этап в истории японских видеоигр [1, c. 202]. Внедрение систем *CD*-*ROM* совпало с распространением инновационных технологий, привнесенных американскими и канадскими компаниями, как в программном, так и в аппаратном обеспечении. На третьем этапе сильные стороны японской индустрии (разнообразие сюжетов и персонажей, графика) постепенно теряли свое значение, поскольку постоянное совершенствование технологий сняло целый ряд ограничений с западных разработчиков.

Новые технологии позволили расширить детализацию графики и увеличивать объемы памяти. В индустрии видеоигр появились североамериканские специализированные производители графических чипов, 208 *Nvidia* (1993 г.) и *ATI* *Technologies* (1985 г.), и уже в начале 1990-х гг. был разработан 3*D*-движок. Игры на основе 3*D*-движка характеризовались более реалистичным графическим представлением и симуляцией физического мира. Кроме того, реалистичное графическое представление стало важным маркетинговым инструментом, так как потребители и разработчики стали судить о качестве, основываясь именно на этом параметре. Разработчики игр, в частности, начали оценивать достижения и статус программистов на основе сложности, создаваемых ими графики и геймплея [2, c. 401]. Расходы, связанные с ростом стоимости разработки одной игры (так, средняя стоимость разработки игры для *Nintendo* *Entertainment* *Systems* в 1990-х гг. составляла 100 тыс. долл., и к 2007 г. она достигала 10 млн долларов для *PlayStation*), только закрепили зависимость от лицензированных персонажей из кино, анимационных фильмов и других секторов индустрии развлечений, таких как страховка и увеличение шансов на то, что игра станет бестселлером [2, c. 166].

Глобализация не обошла стороной и игровую индустрию. Ускорение технического прогресса в 1990-х гг. неизбежно привело к началу спекулятивных войн. Техническое обновление проходило каждые несколько лет, что осложняло положение не только производителей консолей, но и разработчиков игр, которым приходилось разрабатывать игры в перспективе на дальнейшее портирование на новейшие платформы. Спекулятивная конкуренция продолжала усиливаться в 2000-х гг. в дополнение к тому, что *Microsoft*, первый производитель неяпонской консоли, занимал значительную долю рынка с *Xbox* (12 % в мире на 2001-2005 гг.) [4].

Несмотря на успех, в частности, в области разработки программного обеспечения, сегодня в японской игровой индустрии проявляется множество кризисных явлений. В XXI в. японский культурный экспорт является одним из наиболее успешных примеров работы мягкой силы. Однако в отличие от остальной продукции массовой культуры, большая часть видеоигр все еще базируется на внутрияпонском рынке. Это связано прежде всего с культурными особенностями и менталитетом японцев, что отражено в жанровой специфике (симуляторы свиданий, платформеры, аркадные автоматы). Несмотря на то, что 209 аниме и манга получили признание во всем мире, именно сосредоточенность основной части игр на поп-культуре и вкусовых предпочтениях японцев значительно усложняет выход на международный рынок. В отличие от общепонятной для большинства игроков западной культурной традиции, японская культура и язык стали достаточно серьезным препятствием для японских компаний. Стратегия японского правительства *Cool Japan*, принятая в 2012 г. И направленная на популяризацию японской культуры в мире, включает в себя продвижение и поддержку японских разработчиков видеоигр [5].

Расширение и рост Японии на международном игровом рынке осложняется не только непосредственным содержанием игр, но и сохранением закрытого характера всей индустрии и, в частности, самих японцев. Во-первых, японские компании медленно адаптируются к глобальным трендам, игнорируя последние тенденции внешних рынков. Отказ от расширения студий в других странах и найма иностранных разработчиков привел к потере конкурентоспособности. Несмотря на то что Япония является одной из крупнейших экономик мира, стремящейся активно участвовать во всех международных процессах, остается множество аспектов, в которых Япония не готова меняться.

Во-вторых, разработчики программного обеспечения достаточно медленно внедряют межплатформенное (промежуточное) программное обеспечение, которое упрощает создание игр для разных игровых платформ (ПК, *Playstation*, *Xbox* и т. д.) [3]. Жесткая конкуренция японских компаний на внутреннем и внешнем рынках вынуждает компании скрывать спецификацию своих новейших разработок, ограничивать разработчиков игр в выборе той или иной платформы с целью привлечь покупателя так называемым «эксклюзивом». Подобная политика тормозит техническое развитие всего рынка, в отличие от американских и европейских разработчиков, более склонных к взаимовыгодному сотрудничеству.

В-третьих, с момента появления игр стоимость их разработки значительно изменилась. Если в 1990-е гг. разработка игр для приставок 210 компании *Nintendo* составляла в среднем 100 тыс. долларов, то сегодня она может доходить до 10 млн долларов и выше. С учетом закрытой спецификации игровых приставок от японских компаний (прим. *SonyPlaystation* и *NintendoSwitch*) и планки, которую устанавливают «гиганты» индустрии, небольшие японские компании-разработчики не могут удержаться на рынке. Кроме того, в последнее время усиливается тенденция слияния компаний. Наиболее ярким примером стало слияние конкурирующих *Squaresoft* и *Enix* в 2003 г. [1, c. 227].

С повсеместным распространением Интернета появление так называемых социальных игр, а именно игр для социальных сетей, стало поворотным моментом для японской индустрии. В отличие от США, где социальные игры базируются прежде всего в социальных сетях и, соответственно, воспроизводятся в веб-браузере на ПК, японцы компьютерам и ноутбукам предпочитают смартфоны. На 2016 г. на мобильные платформы приходилось 52 % игрового рынка [4]. Для крупнейших представителей японской индустрии, в частности, для производителей консолей, переход значительной части аудитории на мобильные платформы стал достаточно сильным ударом. Если до этого разработчики должны были считаться с компаниями-производителями ПО и дистрибьюторами, то сегодня социальные игры не имеют каких-либо ограничений и свободно распространяются в интернете без какого-либо посредничества.

Феномен японской игровой индустрии является следствием ряда внутренних и внешних факторов. Несмотря на заимствованный характер, именно в Японии игровая индустрия вышла на качественно новый уровень, вобрав в себя массовую и традиционную культуру, создав новые бизнесстратегии и подходы к продвижению медиафраншиз. Нельзя исключать и всеобъемлющее американское влияние, которое проявилось не только в послевоенном переустройстве всей японской экономики, но и в культурном экспорте западных ценностей, которые сегодня являются одним из ключевых элементов развлекательной индустрии Японии. Тем не менее, несмотря на 211 успех политики мягкой силы, культурный экспорт через видеоигры все еще крайне незначителен и уступает остальной развлекательной индустрии.

Экономическое чудо не только стало фундаментом для потребительского роста, но оформило специфическую структуру индустрии видеоигр, где ключевую роль сыграли крупнейшие инновационные отрасли промышленности того времени: электроника, электротехника и робототехника. Игровая индустрия явилась еще одним примером успешного заимствования, трансформации с учетом национальной специфики и феноменального успеха на мировом рынке.

# ПРОЦЕССЫ ТЕСТИРОВАНИЯ И РАЗРАБОТКА ТЕСТОВОЙ ДОКУМЕНТАЦИИ

## Индивидуальное задание от компании ООО «ОпенМайГейм»

На период прохождения технологической практики поставлены следующие задачи:

* ознакомится с организационной структурой и деятельностью организации ООО «ОпенМайГейм»;
* ознакомится с требованиями техники безопасности и охраны труда;
* изучить обязанности инженера-программиста;
* ознакомиться с заданной предметной областью;
* разработать тестовую документацию к проекту «*WordPizza*».

## Модели разработки программного обеспечения

Процесс создания ПО – совокупность процессов, приводящих к созданию программного продукта.

Фундаментальные процессы, присущие любому проекту создания ПО:

* разработка спецификации требований на ПО (определяют функциональные характеристики системы и обязательны для выполнения);
* создание программного обеспечения (создание ПО согласно спецификации);
* аттестация ПО (созданное ПО должно пройти аттестацию для подтверждения соответствию требованиям заказчика);
* модернизация ПО (совершенствование ПО согласно измененным требованиям потребителя).

Модель разработки ПО (*Software* *Development* *Model*, *SDM*) – структура, систематизирующая различные виды проектной деятельности, их взаимодействие и последовательность в процессе разработки ПО. Выбор той или иной модели зависит от масштаба и сложности проекта, предметной области, доступных ресурсов и множества других факторов.

Выбор модели разработки ПО серьёзно влияет на процесс тестирования, определяя выбор стратегии, расписание, необходимые ресурсы и т.д. Моделей разработки ПО много, но в общем случае классическими можно считать водопадную, *v*-образную, итерационную инкрементальную, спиральную и гибкую.

***2.1.1*** Водопадная модель (*waterfall model*) сейчас представляет скорее исторический интерес, так как в современных проектах практически неприменима. Она предполагает однократное выполнение каждой из фаз проекта, которые, в свою очередь, строго следуют друг за другом (рисунок 2.1.) [6, с. 21].

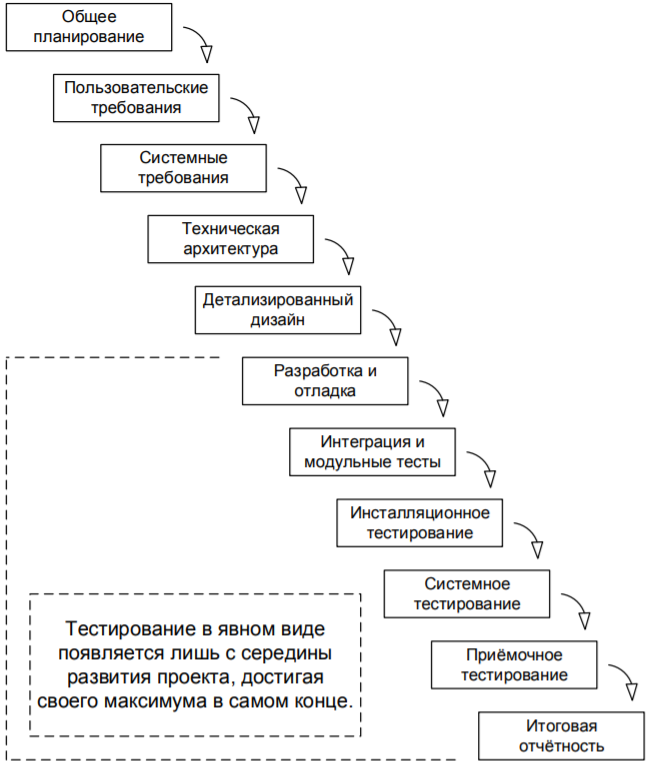


Рисунок 2.1 – Водопадная модель разработки ПО

Очень упрощённо можно сказать, что в рамках этой модели в любой момент времени команде «видна» лишь предыдущая и следующая фаза. В реальной же разработке ПО приходится «видеть весь проект целиком» и возвращаться к предыдущим фазам, чтобы исправить недоработки или что-то уточнить

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

***2.1.2*** *V*-образная модель (*V-model*) является логическим развитием водопадной. Можно заметить (рисунок 2.2), что в общем случае как водопадная, так и *v*-образная модели жизненного цикла ПО могут содержать один и тот же набор стадий, но принципиальное отличие заключается в том, как эта информация используется в процессе реализации проекта [6, с. 22].

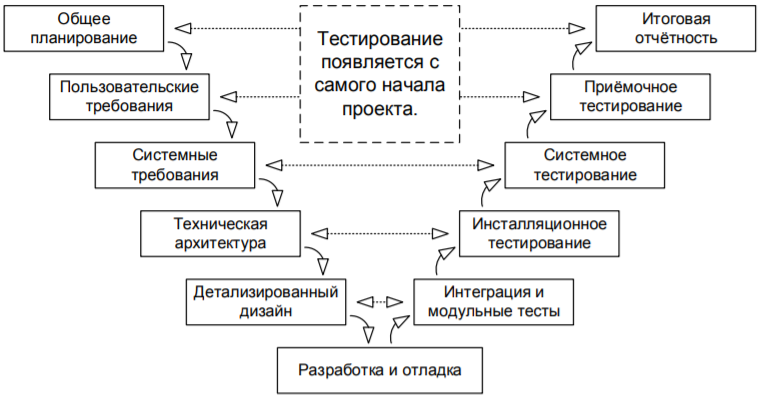


Рисунок 2.2 – *V*-образная модель разработки ПО

Очень упрощённо можно сказать, что при использовании *v*-образной модели на каждой стадии «на спуске» нужно думать о том, что и как будет происходить на соответствующей стадии «на подъёме». Тестирование здесь появляется уже на самых ранних стадиях развития проекта, что позволяет минимизировать риски, а также обнаружить и устранить множество потенциальных проблем до того, как они станут проблемами реальными.

***2.1.3*** Итерационная инкрементальная модель (*iterative model*, *incremental model)* является фундаментальной основой современного подхода к разработке ПО. Как следует из названия модели, ей свойственна определённая двойственность:

* с точки зрения жизненного цикла модель является итерационной, т. к. подразумевает многократное повторение одних и тех же стадий;
* с точки зрения развития продукта (приращения его полезных функций) модель является инкрементальной [6, с. 23].

Проблема итерационно-инкрементальной модели заключается в том, что, каждый новый билд должен быть интегрирован с предыдущим во всех системах. Разборка на билды тоже не элементарна. Если есть несколько билдов, которые еще и необходимо разобрать, все превращается в модель «компилируй и исправляй ошибки». Как бы то ни было, большое количество сборок означает малое количество изменений между ними.

Ключевой особенностью данной модели является разбиение проекта на относительно небольшие промежутки (итерации), каждый из которых в общем случае может включать в себя все классические стадии, присущие водопадной и v-образной моделям (рисунок 2.3). Итогом итерации является приращение (инкремент) функциональности продукта, выраженное в промежуточном билде (*build*).

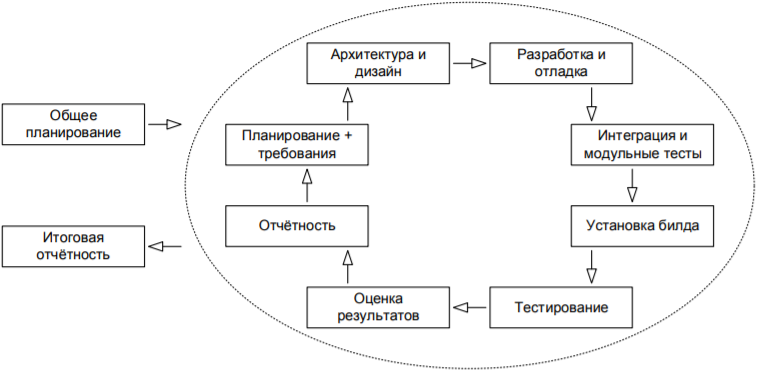


Рисунок 2.3 – Итерационная инкрементальная модель разработки ПО

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

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

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

***2.1.4*** Спиральная модель (*spiral model*) представляет собой частный случай итерационной инкрементальной модели, в котором особое внимание уделяется управлению рисками, в особенности влияющими на организацию процесса разработки проекта и контрольные точки [6, с. 24].

В спиральной модели жизненный путь разрабатываемого продукта изображается в виде спирали, которая, начавшись на этапе планирования, раскручивается с прохождением каждого следующего шага. Таким образом, на выходе из очередного витка мы должны получить готовый протестированный прототип, который дополняет существующий билд. Прототип, удовлетворяющий всем требованиям – готов к релизу.

Схематично суть спиральной модели представлена на рисунке 2.4, где явно выделены четыре ключевые фазы:

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

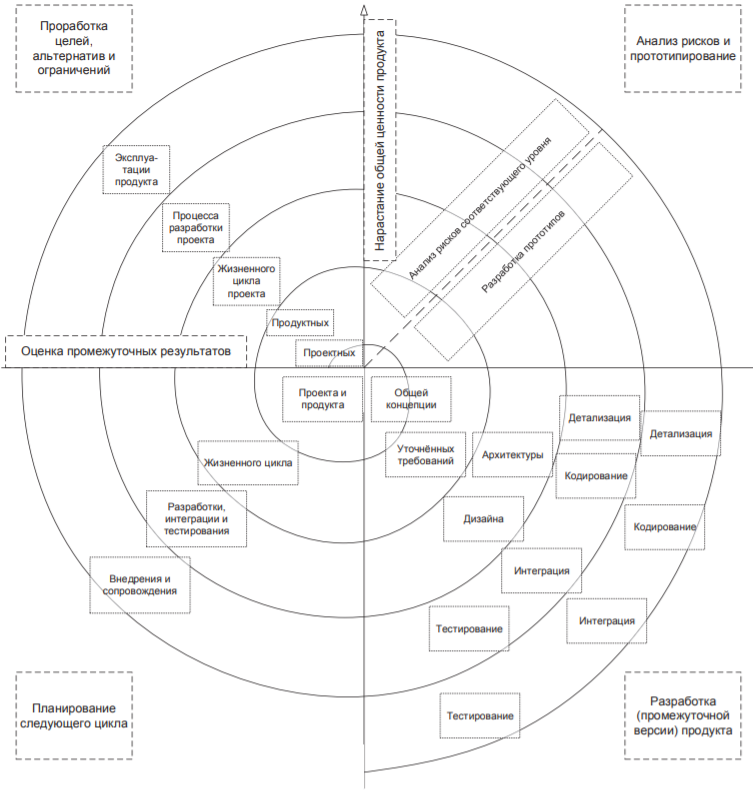


Рисунок 2.4 – Спиральная модель разработки ПО

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

***2.1.5*** Гибкая модель (*agile model*) представляет собой совокупность различных подходов к разработке ПО и базируется на т. н. «*agile*-манифесте»:

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

Положенные в основу гибкой модели подходы являются логическим развитием и продолжением всего того, что было за десятилетия создано и опробовано в водопадной, *v*-образной, итерационной инкрементальной, спиральной и иных моделях. Причём здесь впервые был достигнут ощутимый результат в снижении бюрократической составляющей и максимальной адаптации процесса разработки ПО к мгновенным изменениям рынка и требований заказчика [6, с. 26].

Очень упрощённо можно сказать, что гибкая модель представляет собой облегчённую с точки зрения документации смесь итерационной инкрементальной и спиральной моделей (рисунок 2.5), при этом следует помнить об «*agile*-манифесте» и всех вытекающих из него преимуществах и недостатках.

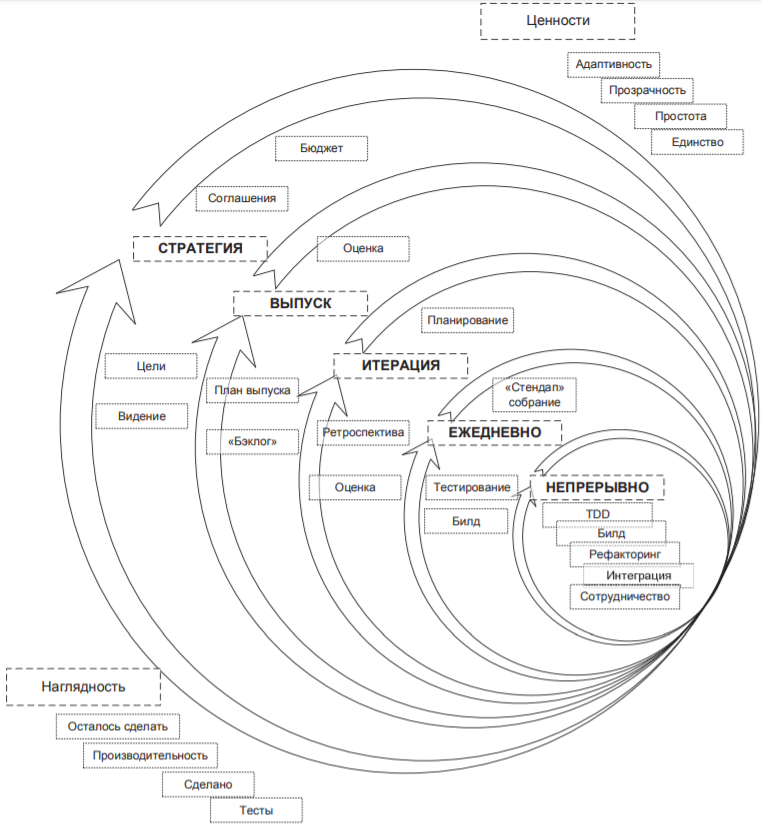


Рисунок 2.5 – Гибкая модель разработки ПО

Главным недостатком гибкой модели считается сложность её применения к крупным проектам, а также частое ошибочное внедрение её подходов, вызванное недопониманием фундаментальных принципов модели. Тем не менее можно утверждать, что всё больше и больше проектов начинают использовать гибкую модель разработки

## Жизненный цикл тестирования

Следуя общей логике итеративности, превалирующей во всех современных моделях разработки ПО, жизненный цикл тестирования также выражается замкнутой последовательностью действий (рисунок 2.6).

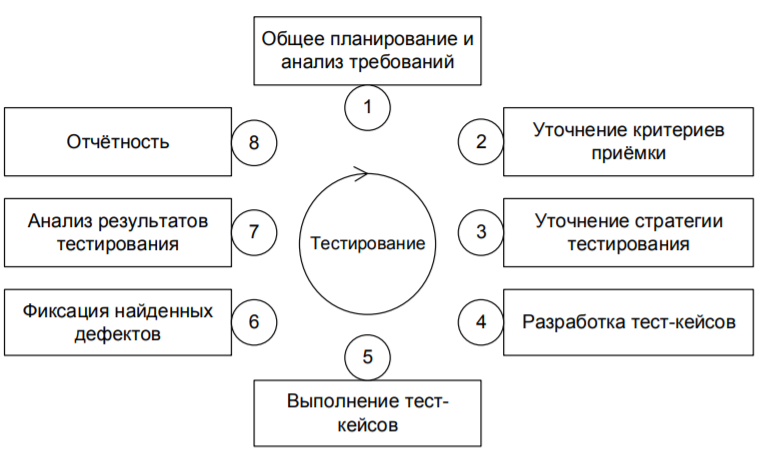


Рисунок 2.6 – Схема последовательности жизненного цикла тестирования

Важно понимать, что длина такой итерации (и, соответственно, степень подробности каждой стадии) может варьироваться в широчайшем диапазоне – от единиц часов до десятков месяцев. Как правило, если речь идёт о длительном промежутке времени, он разбивается на множество относительно коротких итераций, но сам при этом «тяготеет» к той или иной стадии в каждый момент времени (например, в начале проекта больше планирования, в конце – больше отчётности).

## 2.3 Требования к программному обеспечению

Тестирование требований является необходимой и очень важной процедурой, которая в дальнейшем поможет оптимизировать работу команды и избежать недопониманий, а также позволяет понять, можно ли в принципе выполнить данные требования – с точки зрения времени, ресурсов и бюджета.

Требование (*requirement*) – описание того, какие функции и с соблюдением каких условий должно выполнять приложение в процессе решения полезной для пользователя задачи.

В общем случае требования можно разделить на два больших вида в зависимости от времени и места её использования (рисунок 2.7).



Рисунок 2.7 – Соотношение понятий «продуктная документация» и «проектная документация»

Продуктная документация (*product documentation*, *development documentation*) используется проектной командой во время разработки и поддержки продукта. Она включает:

* план проекта (*project management plan*) и в том числе тестовый план (*test plan*);
* требования к программному продукту (*product requirements document*, *PRD*) и функциональные спецификации (*functional specification document*, *FSD*);
* архитектуру и дизайн (*architecture and design*);
* тест-кейсы и наборы тест-кейсов (*test cases*, *test suites*);
* технические спецификации (*technical specifi cations*), такие как схемы баз данных, описания алгоритмов, интерфейсов и т. д.

Проектная документация (*project documentation*) включает в себя как продуктную документацию, так и некоторые дополнительные виды документации и используется не только на стадии разработки, но и на более ранних и поздних стадиях (например, на стадии внедрения и эксплуатации). Она включает:

* пользовательскую и сопроводительную документацию (*user and accompanying documentation*), такую как встроенная помощь, руководство по установке и использованию, лицензионные соглашения и т. д;
* маркетинговую документацию (*market requirements document*, *MRD*), которую представители разработчика или заказчика используют как на начальных этапах (для уточнения сути и концепции проекта), так и на финальных этапах развития проекта (для продвижения продукта на рынке) [4, с. 35-36].

## Описание тест-кейса и дефекта

Тестирование программного обеспечения – это:

* процесс исследования ПО с целью получения информации о качестве продукта;
* процесс проверки соответствия заявленных к продукту требований и реально реализованной функциональности, осуществляемый путем наблюдения за его работой в искусственно созданных ситуациях и на ограниченном наборе тестов, выбранных определенным образом;
* оценка системы с тем, чтобы найти различия между тем, какой система должна быть и какой она есть.

В широком смысле, тестирование – это одна из техник контроля качества (*Quality Control*), которая включает планирование, составление тестов, непосредственно выполнение тестирования и анализ полученных результатов.

Важно понимать, что тестирование ПО включает не только собственно проведение тестов, но и многие другие действия, связанные с процессом обеспечения качества:

* анализ и планирование;
* разработку тестовых сценариев;
* оценку критериев окончания тестирования;
* написание отчетов;
* рецензирование документации (в том числе и исходного кода);
* проведение статического анализа.

Документация – это еще одна составляющая программного продукта любой уважающей себя организации, занимающейся разработкой программного обеспечения. Но не все организации уделяют достаточное количество времени разработке стоящей документации.

Тест-кейс (*test case)* – набор входных данных, условий выполнения и ожидаемых результатов, разработанный с целью проверки того или иного свойства или поведения программного средства.

Структура тест-кейса заключается в «троице», приведенной на рисунке 2.8.



Рисунок 2.8 – Структура тест-кейса

Способ описания тест-кейсов и их структура в каждой компании или команде быть разным: иметь разные глубины описания необходимых действий и результатов, иметь разные структурные составляющие. Но, хорошая структурированность и высокая удобность шаблонов тестовых случаев, может весьма сократить время рутинных заполнений форм и повысить эффективность команды в целом.

Далее подробно рассмотрены основные поля, необходимые при описании дефекта.

Параметры тест-кейса:

* номер (*ID*) тест-кейса – уникальный (например, числовой) идентификатор;
* краткое описание тест-кейса – одна или несколько фраз, из которых ясно, что проверяется данным сценарием;
* ссылка на требования – прямая ссылка или указание названия и версии документа с требованиями;
* автор тест-кейса – тестировщик, разработавший тест-кейс;
* приоритет тест-кейса – насколько важен тест-кейс для проверки данного функционала;
* название\модуль\версия ПО – точное описание тестируемого ПО.

Настройка среды и данные для тестирования:

* данные и параметры для тестирования – исходные данные, необходимые для выполнения проверок;
* предварительная настройка окружения\системы (*pre-conditions*) – подготовка необходимой аппаратной части и\или выполнение программных настроек.

Сценарий тестирования:

* шаги теста – кратко и четко описанное атомарное действие, необходимое для проверки;
* ожидаемый результат – что ждем после этого действия;
* фактический результат – что получаем в реальности (совпадает или нет с ожиданием).

Результаты тестирования:

* статус тест-кейса – текущие состояние теста (например, «разработан», «отправлен в архив»);
* дата выполнения тест-кейса – дата, когда тест проходили последний раз;
* фактический результат - результат выполнения тест-кейса (например, «пройден», «заблокирован»);

Настройка среды:

* возвращение окружения\системы к начальному состоянию (*post-conditions*) – отмена всех сделанных ранее настроек.

Дефект (баг) – это несоответствие фактического результата выполнения программы ожидаемому результату.

Дефекты обнаруживаются на этапе тестирования ПО, когда тестировщик проводит сравнение полученных результатов работы программы (компонента или дизайна) с ожидаемым результатом, описанным в спецификации требований.

На рисунке 2.9 представлена блок-схема, показывающая основные статусы и возможные переходы от статуса к статусу в процессе его существования.



Рисунок 2.9 – Этапы жизненного цикла дефекта

Далее подробно рассмотрено описание данной схемы:

* создан – тестировщик нашел баг, дефект успешно занесен в «*Bug-tracking*» систему;
* открыт – после того, как тестировщик отправил ошибку, она либо автоматически, либо вручную назначается на человека, который должен её проанализировать, в зависимости от решения менеджера проекта, баг может быть: отложен и отклонен;
* назначен – если ошибка актуальна и должна быть исправлена в следующей сборке (*build*), происходит назначение на разработчика, который должен исправить ошибку;
* исправлен – ответственный за исправление бага разработчик заявляет, что устранил дефект;
* проверен – тестировщик проверяет, действительно ли ответственный разработчик исправил дефект, или все-таки разработчик безответственный, если бага больше нет, он получает данный статус;
* переоткрыт – если опасения тестировщика оправданы и баг в новом билде не исправлен – он все так же потребует исправления, поэтому заново открывается;
* закрыт – в результате определенного количества циклов баг все-таки окончательно устранен и больше не потребует внимания команды – он объявляется закрытым.

Классификация дефектов, с точки зрения степени влияния (*Severity*) на работоспособность ПО:

* *blocker* – ошибка, которая приводит программу в нерабочее состояние, дальнейшая работа с программной системой или ее функциями – невозможна;
* *critical* – критический дефект, приводящий некоторый ключевой функционал в нерабочее состояние, так же это может быть существенное отклонение от бизнес логики, неправильная реализация требуемых функций, потеря пользовательских данных и т. д;
* *major* – весьма серьезная ошибка, свидетельствующая об отклонении от бизнес логики или нарушающая работу программы, не имеет критического воздействия на приложение;
* *minor* – незначительный дефект, не нарушающий функционал тестируемого приложения, но который является несоответствием ожидаемому результату;
* *trivial* – баг, не имеющий влияние на функционал или работу программы, но который может быть обнаружен визуально.

Градация дефектов, с точки зрения приоритетности исправления (*Priority*):

* *high* – баг должен быть исправлен как можно быстрее, т. к. он критически влияет на работоспособность программы;
* *medium* – дефект должен быть обязательно исправлен, но он не оказывает критическое воздействие на работу приложения;
* *low* – ошибка должна быть исправлена, но она не имеет критического влияния на программу и устранение может быть отложено, в зависимости от наличия других более приоритетных дефектов.

## 2.5 Покрытие продукта тестами и их прогон

Проект «*WordPizza*» относится реалистичному сеттингу, выбранному исходя из интересов целевой аудитории – кулинария и готовка пиццы, а также путешествие по миру.

На основе проекта «*WordPizza*» были написаны предложения по тест-кейсам и обсуждены с куратором и разработчиками проекта.

Далее была составлена концепция тестирования. На ее основе были созданы два сценария проверки:

* базовый (ежедневно);
* полный (еженедельно).

На рисунке 2.10 представлена концепция тестирования.



Рисунок 2.10 – Концепция тестирования

Для каждого сценария были написаны подробные тест-кейсы в *Excel*. На их основе был покрыт тестами проект «*WordPizza*».

Пример тест кейса приведен в таблице 2.1.

Таблица 2.1 – Пример тест-кейса

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер | Название тест-кейса | Шаги | Ожидаемый результат | Приоритет тест-кейса |
| 1 | Установка и запуск приложения | 1. Установка билда  2. Запуск приложения | Установка билда и запуск приложения. | *critical* |
| 2 | Основная механика | 1. Поиск слов из набора букв  2. Выделение букв кривой линией, следующей за пальцем | Найти все слова из заданного списка слов. Буквы располагаются по кругу, выделяются они кривой линией, следующей за пальцем. | *high* |

Тест-кейсы из *Excel* были импортированы в *TestRail*. Тесты модифицируются и пополняются в процессе тестирования и в настоящее время.

На основе тест-кейсов создаются прогоны, которые могут включать в себя определенный набор тест-кейсов (например, все тест-кейсы для базовой проверки или в рамках полной проверки тест-кейсы только для высокого разрешения). В прогоны для набора тестов можно добавлять конфигурации. В качестве конфигураций могут выступать проекции, сервера.

По результатам прогонов формируются отчеты, которые рассылаются заинтересованным лицам в виде архивов с файлами с расширением .*html*.

Более подробный просмотр результатов прогона непосредственно в *TestRail* представлен на рисунке 2.11.

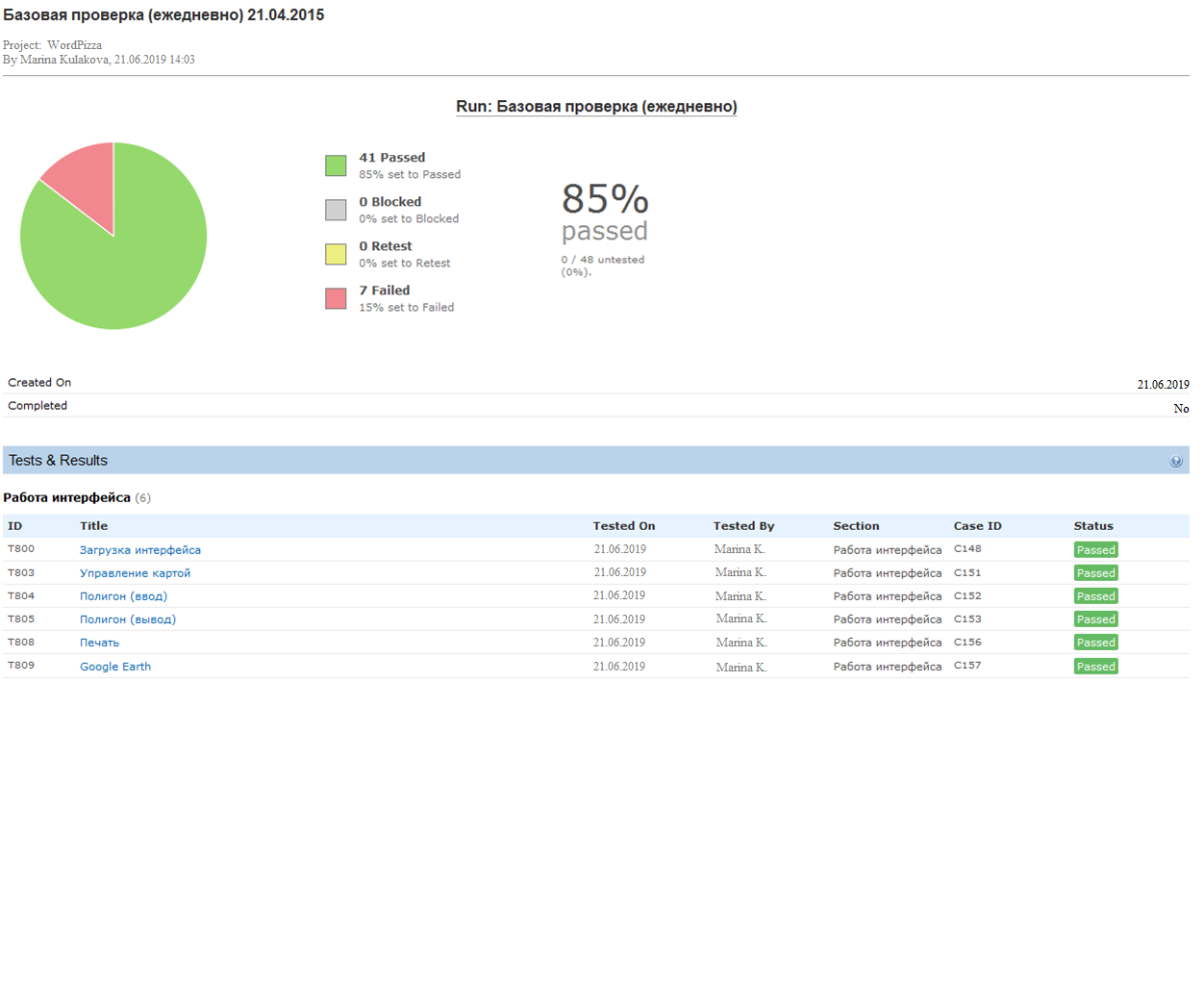


Рисунок 2.11 – Результаты прогона в *TestRail*

В результате работы было найдено и зафиксировано 20 сбоев и 7 предложений. Фактически, сбоев было найдено больше. На рисунке 2.12 представлена диаграмма сбоев.

Рисунок 2.12 – Диаграмма сбоев

Были написано несколько тривиальных автоматических тестов на загрузку интерфейсов, проверку наличия элементов, включенных по умолчанию параметров.

# ЗАКЛЮЧЕНИЕ

Технологическая практика – этап обучения студента, где проявляются навыки выполнения самостоятельной научно-исследовательской работы и овладения методикой исследования и эксперимента при решении актуальной задачи в области, избранной студентом. Любая практика работа является неотъемлемой частью обучения студентов.

В рамках прохождения технологической практика был разработан рабочий процесс в мобильном геймдеве. После изучения документации по проекту были разработаны и написаны тесты к проекту. Приложение проходит все тесты, что говорит о корректности его работы.

Об организации можно сделать следующий вывод: «ОпенМайГейм» – это небольшая, но успешная компания широкого профиля. Сотрудниками компании являются высококвалифицированные специалисты в области информационных технологий. Компания занимает одно из лидирующих мест в сфере информационных технологий в Беларуси, что не может не говорить о престижности и перспективности данного предприятия.

# Список использованных источников

1. Koizimi, M. Japanese Video Game Industry: History of Its Growth and Current State [Электронный ресурс]. – Режим доступа: https://link.springer.com/chapter/ 10.1007/978-3-319-43820-7\_2. – Дата доступа: 03.07.2019.
2. Шефф, Д. Game Over: как Nintendo завоевала мир / Д. Шефф. – М.: Белое Яблоко, 2017. – 384 с.
3. История Nintendo [Электронный ресурс]. – Режим доступа: https:// www.nintendo.ru/-/-Nintendo/-Nintendo-625945.html. – Дата доступа: 03.07.2019.
4. Japanese videogame market analysis [Электронный ресурс]. – Режим доступа: https://newzoo.com/ insights/countries/japan. – Дата доступа: 03.07.2019.
5. Cool Japan Strategy [Электронный ресурс]. – Режим доступа: <http://www> .meti.go.jp/english/policy/mono\_info\_service/creative\_industries/pdf/120116\_01a.pdf. – Дата доступа: 03.07.2019.
6. Куликов, С.C. Тестирование программного обеспечения. Базовый курс / С.С. Куликов. – Минск: Четыре четверти, 2017. – 312 с.
7. Как создавать безопасные условия труда для *IT*-специалиста [Электронный ресурс]. – Режим доступа: http://soutsar.ru/posts/1258120. – Дата доступа: 04.07.2019.
8. Должностная инструкция инженера-программиста [Электронный ресурс]. – Режим доступа: http://all-docs.ru/index.php?page=26&vi1=1476. – Дата доступа: 04.07.2019.
9. Должностная инструкция инженера-программиста на производстве [Электронный ресурс]. – Режим доступа: http://byru.info/baza02/blan1352.shtm. – Дата доступа: 04.07.2019.

# ПРИЛОЖЕНИЕ А

(обязательное)

**Охрана труда в организации и техника безопасности на рабочем месте программиста**

*IT*-специалисты, как и любые другие работники, должны пройти обучение и проверку знаний по охране труда либо в учебном центре, либо в самой организации. Если в ней есть комиссия по проверке знаний по охране труда из не менее трех человек, аттестованных в специализированном учебном центре.

Обучение охране труда в организации проводят по самостоятельно разработанным программам. Их составляют, опираясь на типовые программы, а также учитывая особенности отрасли, в которой работает организация. Учебные планы и программы по охране труда не требуется согласовывать с органами власти.

Кроме того, *IT*-специалистам нужно присвоить группу по электробезопасности:

* I – неэлектротехническому персоналу (тем, кто напрямую не связан с электроустановками, но имеют риск поражения электрическим током);
* II и выше – электротехническому и электротехнологическому персоналу (тем, кто непосредственно занят эксплуатацией или наладкой электрооборудования).

Чтобы присвоить работникам группу I по электробезопасности, не нужно их специально обучать – достаточно инструктажа. Провести его должен ответственный за электрохозяйство компании сотрудник с группой по электробезопасности не ниже III. Инструктаж должен повторяться не реже одного раза в год.

Обучение на группы II – V по электробезопасности *IT*-специалисты также могут проходить как внутри организации, так и в учебном центре. Для проверки знаний в компании должна быть комиссия из пяти человек, как минимум трое из которых были аттестованы.

Закон обязывает работодателя направлять на предварительный и периодические медосмотры сотрудников, которые проводят за компьютером более 50 процентов рабочего времени. Труд *IT*-специалистов почти непрерывно связан с компьютером. Таким образом, медосмотры им нужны.

Микроклимат офисных помещений должен соответствовать нормам СанПин 2.2.4.548-96. Если работа за компьютером является вспомогательной, то температура воздуха должна быть допустимой (20 – 21,9 °С), если работа является основной – то оптимальной (22 – 24 °С).

В офисе нужно провести замеры температуры сертифицированным поверенным прибором. Если она меньше оптимальной или допустимой, необходимо сообщить об этом руководству и потребовать привести условия труда в норму.

Важно создать сбалансированный режим труда и отдыха, предварительно проанализировав загрузку *IT*-специалистов. Сотрудники, работающие за компьютером, должны делать регулярные перерывы. Помещение, оборудованное компьютерами, нужно проветривать каждый час, а специалистам рекомендуется чередовать работу за компьютером и без него. Если такой возможности нет, лучше делать 10 – 15-минутные перерывы через каждые 45–60 минут работы.

Как правило, *IT*-специалисты трудятся по пятидневной рабочей неделе с двумя выходными днями. Но если производство непрерывное, такие сотрудники могут трудиться и посменно. В таком случае о графике работы сотрудников оповещают не менее, чем за месяц. Если специалиста нужно привлечь к работе не по графику (например, до начала смены или после, в выходные дни), должен соблюдаться порядок, установленный Трудовым кодексом.

*IT*-специалистам нередко приходится сталкиваться с авралами на работе. Если сотрудники подолгу задерживаются в офисе в период сдачи срочного проекта, нужно, верно, оформить переработку и оплатить ее.

Работа будет сверхурочной, только если сотрудник выполняет ее по инициативе руководителя по окончании трудового дня (смены) или сверх нормального числа рабочих часов за учетный период. Переработку компенсируют за:

* первые два часа работы – не менее чем в полуторном размере;
* последующие часы – не менее чем в двойном размере.

По желанию работника денежную компенсацию можно заменить дополнительным отдыхом продолжительностью не менее отработанного времени.

Чтобы привлечь сотрудника к работе в выходной или праздничный день, нужно оформить следующие документы:

* уведомление о работе в нерабочий праздничный день (с обязательным указанием на право сотрудника отказаться от такой работы);
* приказ о привлечении к работе в нерабочий праздничный день и предоставлении дополнительных дней отдыха за работу в нерабочий праздничный день;
* заявление работника о предоставлении другого дня для отдыха.

Трудовой режим работника должен быть четко прописан в его трудовом договоре, должностной инструкции и локальных нормативных актах организации (например, в Правилах трудового распорядка). За их нарушения к работнику могут применяться дисциплинарные взыскания.

Также можно создать систему мотивации (как материальную, так и нематериальную). Что касается материальной составляющей, то можно внедрить систему премирования по итогам работы в области охраны труда, экологии, промышленной безопасности, например, за «Достижения в области охраны труда, экологии, промышленной безопасности». А нематериальная мотивация может выражаться в оценке достижений в виде почетных грамот, похвальных листов [7].

# ПРИЛОЖЕНИЕ Б

(обязательное)

**Должностная инструкция инженера-программиста**

Общие положения:

1. инженер-программист относится к категории специалистов, принимается на работу и увольняется с работы приказом директора предприятия;
2. на должность инженера-программиста назначается лицо, имеющее высшее техническое или инженерно-экономическое образование без предъявления требований к стажу работы или среднее специальное образование и стаж работы в должности техника I категории не менее 3 лет либо на других должностях, замещаемых специалистами со средним специальным образованием, не менее 5 лет.

На должность инженера-программиста II категории назначается лицо, имеющее высшее техническое или инженерно-экономическое образование и стаж работы в должности инженера-программиста или на других инженерно-технических должностях, замещаемых специалистами с высшим образованием, не менее 3 лет.

На должность инженера-программиста I категории назначается лицо, имеющее высшее техническое или инженерно-экономическое образование и стаж работы в должности инженера-программиста II категории не менее 3 лет.

1. в своей деятельности инженер-программист руководствуется:
2. нормативными документами по вопросам выполняемой работы;
3. методическими материалами, касающимися соответствующих вопросов;
4. уставом предприятия;
5. правилами трудового распорядка;
6. приказами и распоряжениями директора предприятия (непосредственного руководителя);
7. настоящей должностной инструкцией;
8. инженер-программист должен знать:
9. постановления, распоряжения, приказы и другие руководящие и нормативные документы, касающиеся методов программирования и использования вычислительной техники при обработке информации;
10. технико-эксплуатационные характеристики, конструктивные особенности, назначение и режимы работы оборудования, правила его технической эксплуатации;
11. технологию механизированной обработки информации;
12. виды технических носителей информации;
13. методы классификации и кодирования информации;
14. формализованные языки программирования;
15. действующие стандарты, системы счислений, шифров и кодов;
16. методы программирования;
17. порядок оформления технической документации;
18. передовой отечественный и зарубежный опыт программирования и использования вычислительной техники;
19. основы экономики, организации труда и организации производства;
20. правила и нормы охраны труда, техники безопасности, производственной санитарии и противопожарной защиты;
21. Во время отсутствия инженера-программиста его обязанности выполняет в установленном порядке назначаемый заместитель, несущий полную ответственность за надлежащее исполнение возложенных на него обязанностей [8].

На инженера-программиста возлагаются следующие функции:

* разработка программ, направленных на решение экономических и иных задач;
* отладка программ;
* сопровождение внедренных программ и программных средств;
* участие в разработке форм документов, подлежащих машинной обработке.

Для выполнения возложенных на него функций инженер-программист обязан:

* на основе анализа математических моделей и алгоритмов разрабатывать программы, реализующие решение экономических и других задач;
* разрабатывать технологию, этапы и последовательность решения;
* осуществлять выбор языка программирования и перевод на него используемых моделей и алгоритмов задач;
* определять информацию, подлежащую обработке на ЭВМ, ее объемы, структуру, макеты и схемы ввода, обработки, хранения и выдачи информации, методы ее контроля;
* определять объем и содержание данных тестовых примеров, обеспечивающих наиболее полную проверку соответствия программ их функциональному назначению;
* выполнять работу по подготовке программ к отладке и проводить отладку;
* разрабатывать инструкции по работе с программами, оформлять необходимую техническую документацию;
* определять возможность использования готовых программных средств;
* осуществлять сопровождение внедренных программ и программных средств;
* проводить камеральную проверку программ на основе логического анализа;
* определять совокупность данных, обеспечивающих решение максимального числа условий, включенных в программу, выполнять работу по ее подготовке к отладке;
* проводить отладку разработанных программ, корректировать их в процессе доработки;
* разрабатывать и внедрять методы автоматизации программирования, типовые и стандартные программы, программирующие программы, трансляторы, входные алгоритмические языки;
* выполнять работу по унификации и типизации вычислительных процессов;
* принимать участие в создании каталогов и картотек стандартных программ, в разработке форм документов, подлежащих машинной обработке, в проектных работах по расширению области применения вычислительной техники.

Инженер-программист имеет право:

* знакомиться с проектами решений руководства предприятия, касающимися его деятельности;
* получать от руководителей структурных подразделений, специалистов информацию и документы, необходимые для выполнения своих должностных обязанностей;
* привлекать специалистов всех структурных подразделений предприятия для решения возложенных на него обязанностей (если это предусмотрено положениями о структурных подразделениях, если нет – с разрешения руководителя предприятия);
* требовать от руководства предприятия оказания содействия в исполнении своих должностных обязанностей и прав.

Инженер-программист несет ответственность:

* за неисполнение (ненадлежащее исполнение) своих должностных обязанностей, предусмотренных настоящей должностной инструкцией, в пределах, определенных действующим трудовым законодательством Республики Беларусь;
* за совершенные в процессе осуществления своей деятельности правонарушения – в пределах, определенных действующим административным, уголовным и граждански законодательством Республики Беларусь;

за причинение материального ущерба – в пределах, определенных действующим трудовым, уголовным и гражданским законодательством Республики Беларусь [9].