МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 «Программное обеспечение информационных технологий»

Специализация 1-40 01 01 10 «Программное обеспечение информационных технологий (программирование интернет-приложений)»

**Теоретический материал**

**к практическому проекту на тему:**

«Верификация и аттестация программного обеспечения»

Выполнил студент Сивак Михаил Николаевич

(Ф.И.О.)

Руководитель проекта преп.-стаж. Олефрович А. О.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Пацей Н.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты преп.-стаж. Олефрович А. О.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер преп.-стаж. Олефрович Ф. О.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Минск 2020

**Оглавление**

[**1 Определение цели и темы проекта** 4](#_Toc46521121)

[**2 Определение объёма и общего срока выполнения проекта.** 6](#_Toc46521122)

[**2.1 Определение сроков, ориентированное на проект** 6](#_Toc46521123)

[**2.2 Определение сроков, ориентированное на мощности** 7](#_Toc46521124)

[**3 Разбиение выполнения проекта на части** 8](#_Toc46521125)

[**3.1 Планирование** 10](#_Toc46521126)

[**3.2 Графики работ** 11](#_Toc46521127)

[**4 Процесс создания программного обеспечения** 15](#_Toc46521128)

[**4.1 Модели процесса создания программного обеспечения** 16](#_Toc46521129)

[**4.2 Спецификация программного обеспечения** 17](#_Toc46521130)

[**4.3 Проектирование и реализация программного обеспечения** 19](#_Toc46521131)

[**4.4 Требования к программному обеспечению** 20](#_Toc46521132)

[**4.5 Функциональные и нефункциональные требования** 21](#_Toc46521133)

[**4.6 Системные требования** 22](#_Toc46521134)

[**5 Верификация и аттестация программного обеспечения** 23](#_Toc46521135)

[**5.1 Планирование верификации и аттестации** 27](#_Toc46521136)

[**5.2 Инспектирование программных систем** 28](#_Toc46521137)

[**5.3 Проверка программного обеспечения** 30](#_Toc46521138)

[**5.4 Проверка артефакта или спецификации** 31](#_Toc46521139)

**Введение**

В данном документе изложены теоретические материалы о процессах планирования, создания проекта, о верификации и аттестации программного обеспечения, а также затронуты более тонкие моменты, затрагиваемые на протяжении всего жизненного цикла проекта. Материал построен таким образом, чтобы постепенно перейти к основной теме задания для большего понимания процесса проектирования и создания продукта, начиная с первых шагов, предполагающими выбор темы и аудитории, заканчивая последними шагами, ведущими к выпуску первой версии рабочего проекта. Также в качестве примера будет прилагаться разработанный мной веб-сайт. Бэкенд был написан на платформе Node.js с использованием сторонних модулей (библиотек), в качестве фронтенда использовался движок представлений для express.js – handlebars/hbs, а также bootstrap, css, html, javascript.

# **1 Определение цели и темы проекта**

Самый первый шаг при подготовке проекта в любой сфере: бизнесе, домашнем хозяйстве или образовании – состоит в определении целей и задач проекта. Этот шаг определяет, что мы хотим получить после завершения проекта, и какие действия нам необходимо предпринять для достижения этой цели. Члены проектной команды, включая менеджеров проектов, уделяют недостаточно внимания этому, безусловно, важному процессу, или выполняют его неправильно, что, в конце концов, приводит к неудачному завершению проекта.

Неправильно определенные цели и задачи, или цели без задач, приводят к тому, что в процессе реализации проекта возникают перерасход средств, битвы за территорию, выяснение отношений между членами проектной команды, невыполнение промежуточных контрольных точек и, как следствие, неудовлетворенные клиенты.

Цели и задачи должны быть четкими заявлениями о намерениях. Каждая цель должна иметь собственное стремление, которое влияет на конечный результат проекта. Цели и задачи должны быть измеряемы. Они должны отвечать на вопрос «Что?».

Цели представляют собой общие намерения применительно к проекту. В этом процессе цели отвечают на вопрос «что?». Иными словами, «что» будет выполнять проект? Проекты могут иметь более одной цели, и много задач применительно к одной цели.

Примеры:

* Цель разработки веб-сайта: его посетители будут уверены, что существует угроза глобального потепления.
* Цель страховой компании: отдел медицинского страхования увеличит пакет предлагаемых услуг на 10%.
* Цель работы врачебного кабинета: пациенты не будут ждать дольше 1 часа перед посещением врача.

Задачи должны отвечать на вопрос «Как?»

* Задачи представляют собой конкретные действия, которые приводят к выполнению цели. Каждая цель будет иметь одну или несколько связанных с ней задач. По сути, задача определяет «как» будет выполняться процесс.
* Всегда стоит начать определение задачи с глагола. Это гарантирует, что задача станет измеряемой, и конечный результат проекта будет рассматриваться в рамках действия этой задачи. Одновременно каждая задача станет измеряемым промежуточным этапом выполнения проекта.

Примеры:

1. Цель: посетители сайта будут уверены, что глобальное потепление существует.

* Создать сравнительную таблицу стоимости борьбы с глобальным потеплением сегодня и 100 лет назад.
* Проиллюстрировать последствия глобального потепления в фотогалерее.
* Выявить и проанализировать последствия глобального потепления.

2. Цель: отдел медицинского страхования увеличит пакет предлагаемых услуг на 10%.

* Определить перечень и стоимость услуг, предлагаемых страховыми компаниями.
* Проанализировать запросы потенциальных клиентов, чтобы узнать их требования.
* Сравнить предлагаемые услуги и услуги конкурентов.

3. Цель: пациенты не будут ждать дольше 1 часа перед посещением врача.

* Оценить потребность в персонале.
* Закупить новое программное обеспечение по планированию посещений.
* Разработать систему подтверждения графика.

Теперь перейдём к частному случаю – моему собственному проекту.

Для начала я определял цель проекта и рациональность использования итогового продукта. Целью моего проекта является создание удобного сайта для пользователей интернета, на котором можно совершать покупки автомобилей, а также получить возможность доставки автомобилей в указанное место. Основная цель – сделать более удобной продажу автомобилей для поставщиков и покупку – для клиентов. Почему это хорошая идея? Я считаю, что это поможет сэкономить много времени для клиентов, так как ему предстоит лишь оформить заказ и ожидать доставку автомобиля прямо в его город, либо любое другое место, которое он укажет. Также будет прилагаться возможность просто оставлять заказ и самим приезжать, чтобы его забрать. Для начала мой проект ориентируется на продажу новых автомобилей из автосалонов, далее стоит предусмотреть возможность расширения проекта, чтобы другие клиенты могли выкладывать свои объявления б/у автомобилей.

Теперь посмотрим на задачи, для которых предназначен мой проект. То есть мы дадим некоторое представление того, как проект будет выполнять поставленные задачи. Приложение должно взаимодействовать с клиентом, в моём случае клиентом является браузер. Перечислим список задач:

* Регистрация пользователей.
* Авторизация пользователей.
* Просмотр контента незарегистрированными пользователями.
* Оформление заказа.
* Предоставление списка всех товаров.
* Сортировка товаров по различным критериям.
* Поиск товаров по названию.

# **2 Определение объёма и общего срока выполнения проекта.**

Объём проекта невозможно точно определить на начальном этапе, как и время, необходимое для его выполнения. Проект может разрабатываться отдельно, а может являться частью целого комплекса работ. Тогда вам следует координировать свои работы с другими и принять меры во избежание их дублирования.

В содержании проекта должны быть четко указаны даты начала и завершения, а также что нужно сделать. Например, если проект предусматривает разработку и испытания нового изделия, то надо сразу оговорить, что сюда не входят маркетинговые исследования и запуск его в производство.

При получении какого-либо задания нужно избегать недомолвок. Если вам поручили выполнить, разработать какой-то новый проект, сразу уточните все тонкости и нюансы, стараясь ничего не упускать, к примеру, спросить, должны ли вы проводить исследование рынка для определения характеристик этого продукта.

Всегда нужно давать чёткое описание предстоящих работ. Предположим, проект предусматривает "внедрение новой информационной системы". Под словом "внедрение" люди могут понимать:

* установку нового программного обеспечения;
* обучение персонала работе с новым программным продуктом;
* усовершенствование новой программы;
* устранение ошибок, обнаруженных в новом программном обеспечении;
* все перечисленное;
* что-нибудь еще.

**Нужно убедиться в едином понимании содержания проекта вами, распорядителями и соисполнителями.**

## **2.1 Определение сроков, ориентированное на проект**

Планирование сроков охватывает определение потребности во времени работ и их комплексов сначала без учета ограничений по мощностям, только в зависимости от структуры работ. Проведенное таким образом планирование сроков предполагает возможность выбора необходимых мощностей.

Ориентированное на проект определение сроков - в данном случае методом критического пути (ЦПМ) - начинается с нумерации узлов, поскольку при планировании структуры проекта этого еще не делали. Самое раннее из всех возможных время наступления события FZ} определяется последовательным суммированием длительности работы d", начиная с первой работы всей цепочки работ по самому длительному пути к событию j. Наиболее поздний срок наступления события SZj определяется обратным вычитанием продолжительности работы cL, начиная с последней работы цепочки работ по наиболее длительному пути к событию i.

После определения сроков наступлени событий устанавливают **сроки выполнения работ**FAy, FEy, SAy, SEjj. Только некоторые из этих сроков совпадают со временем наступления событий. Отклонение между сроками обоих видов, называемое **буферным временем,**может возникнуть, если одно событие вызывается несколькими работами или если из одного события вытекает несколько работ

Если работа не имеет общего буферного времени, то она называется критической. Последовательность работ в сетевом плане, в котором все работы критические, называется критическим путем (самый длительный по времени путь). Эти определения верны только в том случае, когда самое позднее время окончания работ одной последовательности совпадает с самым ранним из возможных временем ее окончания: FZn = SZn. Так установлено правилами. В этой ситуации имеется по меньшей мере один критический путь через всю сеть, на котором нет общего буферного времени для работ и событий. Если справедливо FZn < SZn, то критическим путем является такой, на котором есть наименьшее

общее буферное время. Удлинение критического пути по времени ведет к сдвигу конечного срока проекта.

Сроки определяют непосредственно на сетевом плане, нанося на него данные о длительности работ и пояснения к результатам расчетов.

В заключение следует напомнить, что в рамках метода ПЕРТ дается не одна, а три оценки продолжительности работ: пессимистическая, вероятная и оптимистическая. Из этих трех значений рассчитывается ожидаемое значение, которым затем оперируют, как это было показано для критического пути в рамках метода ЦПМ. По крайней мере для этих значений могут быть исчислены продолжительность работ и варианты сроков наступления событий, так что при нахождении критического пути и конечного срока для последовательности работ можно условно использовать вероятную оценку.

## **2.2 Определение сроков, ориентированное на мощности**

Исходя из ориентированных на проект сроков проведения работ сравнивают потребность во времени использования мощностей, необходимых для выполнения работ, с их наличием, осуществляя тем самым планирование загрузки мощностей во времени. Ориентированное на проект планирование сроков проводится при этом с учетом загрузки мощностей уже предварительно включенными в план проектами (заказами).

# **3 Разбиение выполнения проекта на части**

Одним из эффективных инструментов управления проектом является структура разбиения работ. Она позволяет определить, какие работы необходимо выполнить для реализации проекта, и установить единую структуру управления этими работами. Структура разбиения работ позволяет участникам проекта и всем заинтересованным лицам достичь ясного представления о конечной продукции проекта и всех работ, необходимых для создания этой продукции.

После определения цели проекта, анализа объёма полного проекта и определения примерного выполнения всех задач, будет отлично разбить проект на несколько частей, разбить на задачи, потом распределить задачи между разработчиками по специализациям. Это значительно экономит время при создании проекта, а также упрощает процесс верификации программного обеспечения и тестирования.

На самом высоком уровне декомпозиции представлен проект в целом. На самом нижнем уровне находятся единичные работы, иногда называемые пакетом работ или рабочим пакетом.

**Пакет работ** — это единица измерения всей деятельности по проекту. Рабочие пакеты закреплены за определенными лицами, имеют четко обозначенные сроки, стоимость и требования к качеству, отражаются в соответствующей документации. Рабочие пакеты — это единицы управления проектом. Они служат информационной базой, первичным элементом учета управления проектом.

В структуру разбиения работ (далее СРР) входят:

* *структура* — совокупность отношений между элементами системы, необходимых и достаточных для достижения цели проекта;
* *разбиение* — разделение на составные части или категории, на более простые составные части, декомпозиция;
* *работа* — продолжительное физическое или умственное усилие направленное на достижение результата; деятельность обязанность, функция, операция, выполняемая сотрудником или коллективом; часть трудового процесса, требующего затрат времени и ресурсов.

Структура разбиения работ позволяет участникам проекта и всем заинтересованным лицам достичь ясного представления о конечной продукции проекта и всех работ, необходимых для создания этой продукции. Она разделяет проект на иерархически связанные управляемые, простые и контролируемые комплексы и пакеты работ, что позволяет достичь необходимого баланса между потребностями управления и оптимальным представлением информации по проекту.

Ключевой ориентацией структуры разбиения работ являются создаваемые результаты которые можно определить как измеримые, осязаемые и проверяемые выходы, продукты, поручаемые вследствие выполнения работ по проекту или отдельной его части. Создаваемые целевые результаты деятельности по проекту составляют в совокупности продукцию проекта.

Для руководства проекта структура разбиения работ является необходимым инструментом так как она позволяет:

* обеспечить достижение целей проекта путем их сравнения с элементами дерева работ различного уровня;
* разложить сложный по содержанию проект на более простые и управляемые составляющие;
* создать основу для сетевого моделирования, планирования, распределения ответственности;
* более детально обозначить требования к ресурсам, необходимым для выполнения работ;
* определить структуру данных необходимых, для текущей оценки стоимости продолжительности и качества работ;
* создать основу для управления рисками проекта.

Разработка структуры разбиения работ представляет собой декомпозицию проекта на его составляющие. Это осуществляется путей последовательного рассмотрения целей и задач проекта, критериев и ограничений, содержания проекта, технических и потребительских требований и других атрибутов проекта и его продукции. По мере того как появляются дополнительные данные по требованиям к проекту, может быть разработана более детальная структура разбиения работ.

Для создания структуры разбиения работ необходимо придерживаться следующих **принципов:**

* мысленно охватывать проект в целом (рассматривать проект какцелостную систему, разбивающуюся на нижних уровнях на более мелкие составляющие);
* постоянно иметь в виду создаваемые результаты (что требуется сделать?);
* всегда думать о конечном результате (какой вклад осуществляет тот или иной компонент дерева работ в конечные результаты?);
* размышлять в терминах производства или создания результатов (какие выбрать методы? какие возможны специализированные процессы? какие установить требования к качеству? какие предполагаются проверки и инспекции?)

При разработке дерева работ необходима формализация представления о конечном продукте проекта:

* что он представляет из себя?
* из каких составных частей состоит?
* каким образом составные части работают в единой системе?
* на удовлетворение каких потребностей направлено использование продукции?

Общий процесс создания структуры разбиения работ проекта состоит из следующих шагов:

* Шаг 1. Идентификация конечной продукции проекта (что должно быть создано и сдано заказчику для достижения цели проекта?) Тщательное изучение документов содержащих общее описание проекта (например, техническое задание, технический проект, состав работ и пр.)
* Шаг 2. Определение основных производственных результатов проекта, которые могут быть промежуточными результатами, например проектная документация.
* Шаг 3. Декомпозиция основных результатов до уровня, необходимого и достаточного для эффективного контроля за проектом. Такие результаты должны иметь самостоятельные показатели качества и стоимости.
* Шаг 4. Совершенствование дерева работ до тех пор пока оно не будет удовлетворять потребностям всех участников проекта и заинтересованных лиц.

Самая важная часть процесса управления. Это составление списка задач и мероприятий, которые должны быть выполнены в рамках проекта. Также указывается взаимосвязь между задачами и требующиеся ресурсы.

Планы устаревают с момента создания, их нужно постоянно корректировать при появлении более свежей информации. Единоразовое создание плана на старте не гарантирует успешность проекта.

Все современные методологии управления проектами «гибкие» — то есть, планирование, исполнение и контроль происходят циклами:

* Создается план;
* Реализуется его часть;
* Контролируется исполнение;
* Вносятся изменения в план, поскольку модификация отдельных частей сильно влияет на конечный результат.

## **3.1 Планирование**

По американской методологии [PMBOK](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B4_%D0%B7%D0%BD%D0%B0%D0%BD%D0%B8%D0%B9_%D0%BF%D0%BE_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8E_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B0%D0%BC%D0%B8) планирование занимает порядка 50% всего времени, что особенно подчеркивает его значимость.

Планирование лучше начать с разделения проекта на части (Work Breakdown Structure — WBS), где каждая фаза разделена на более мелкие задачи. Это помогает понять объем работ и учесть все нюансы.

На рисунке 3.1 можно увидеть схему, которая представляет собой различные фазы разработки проекта, а также все модули, из которых состоят эти фазы.

[](https://www.cossa.ru/upload/medialibrary/ddf/manage_project-3.jpg)

Рисунок 3.1 – Схема разбиения проекта

## **3.2 Графики работ**

Очень удобный и практичный инструмент для этого — диаграмма Ганта. По сути, это календарный план, который учитывает взаимосвязь между поставленными задачами, а также показывает ресурсы, необходимые для их выполнения.

Самый простой вариант диаграммы можно сделать в Exсel: слева в таблице находится название работ, а все остальные столбцы — даты.

На рисунке 3.2 можно увидеть простую реализацию диаграммы Ганта.

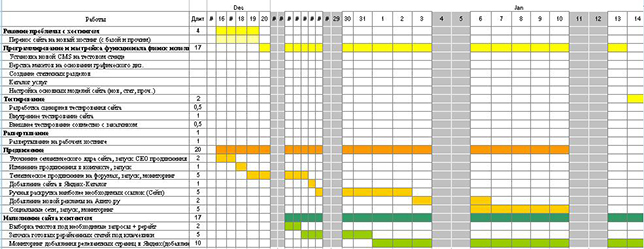
[](https://www.cossa.ru/upload/medialibrary/161/manage_project-4.jpg)

Рисунок 3.2 – Диаграмма Ганта

Если работа выполняется в конкретную дату, она заполнена цветом. Такая схема четко отображает объем работ, сроки и последовательность их выполнения.

При наличии визуальной картины выполняемых задач работа над проектом существенно упрощается, так как они привязаны к календарному графику.

**Как правильно ставить задачи (цели) по проекту**

Рекомендую пользоваться принципом SMART. Как это расшифровывается:

**S (Specific — «конкретный»):** цели проекта определяются четко и конкретно, чтобы все участники и вовлеченные в процесс люди понимали, что и зачем делают. Неконкретную формулировку допускают часто и она плохо влияет на дальнейшее планирование. Например, задача:"сделать хороший сайт" достаточно абстрактна: нужно четко сформулировать, что именно мы хотим получить на выходе.

**M (Measureable — «измеримый»):** чтобы понять, что поставленные цели действительно достигнуты, следует задавать и конечные, и промежуточные критерии оценки (в том числе финансовые). Это поможет проанализировать, насколько процесс продвинулся вперед и соответствует ли срокам. Например, если нужно создать сайт, следует задать критерии измеримости (интернет-магазин на 1000 товаров, бюджет до 30 000 рублей и т. п.).

**A (Achievable — «достижимый»):** нужно требовать, чтобы исполнители приложили усилия, но при этом оглядываться на реальный мир и имеющиеся ресурсы. Не нужно ставить цели, которых не достигнуть, даже если вся команда будет работать 24 часа в сутки. Например, спланировать полет на Марс за 2 месяца нереально — для этого требуется гораздо больше времени.

**R (Relevant — «последовательный»):**все цели должны укладываться в общую концепцию, соотноситься со стратегией развития бизнеса. Например, если вы фокусируетесь на какой-то нише, то нужно развиваться именно в ней. Если не будет результатов, можно рассмотреть и другие ниши, но последовательно.

**T (Timed/Timebounded — «измеримый во времени»):** для целей нужно намечать временные рамки, не только конечные, но и промежуточные. Например, если сайт должен быть готов через месяц, нужно оценить сроки выполнения всех подзадач и четко сформулировать их сотрудникам и подрядчикам.

Как показывает практика, если не задать критерии по SMART, то с большой вероятностью можно как минимум сорвать дедлайн по проекту.

**Чек-лист по планированию проекта**

1. Составить полный список задач (SMART);

2. Распределить ресурсы на каждую задачу, убедиться, что их достаточно;

3. Построить взаимосвязь между задачами — можно ли выполнить их параллельно, последовательно, какие нужно завершить до начала реализации следующих;

4. Определить критический путь, учесть риски по срокам и переносу времени исполнения;

5. Исходя из плана и ресурсов высчитать себестоимость проекта для выставления сметы заказчику.

**Исполнение**

На этом этапе могут возникнуть трудности: например, сложно правильно распределить очередность выполнения задач. Избежать трудностей поможет ряд простых инструментов:

* Матрица Эйзенхауэра;
* Вычеркивание дел;
* Делегирование;
* Тайм-менеджмент (Хронос и Кайрос).

Работают они только тогда, когда вы их правильно и регулярно (!) применяете. Если человек один раз пробует и у него не получается — это не значит, что инструмент не работает.

**Ограничения проектов**

Их существует три:

* По времени;
* По ресурсам (люди, бюджет);
* По результатам (объем выполненных работ, который должен быть на выходе, в финальной версии проекта).

Тройное ограничение относится к фазе исполнения проекта, так как именно на этом этапе чаще всего возникает ситуация, что кто-то из сотрудников что-то не успел сделать. Обычное оправдание: «Хотел сделать хорошо, поэтому не справиться с остальными задачами».

Постарайтесь донести мысль, что некоторые вещи не нужно пытаться довести до идеала. Нужно соблюдать баланс, делая работу максимально качественно, но учитывая ограниченность по времени и ресурсам.

Во ФРИИ мы стараемся следовать методологии Lean Startup. Это значит, что создается минимально работоспособный прототип продукта, на котором проверяется — будет ли работать идея в целом. А дальше, если работает, можно уже допиливать отдельные моменты.

**Контроль выполнения проекта**

Контроль нужно осуществлять на каждом этапе, а не тогда, когда проект близится к завершению. В этом вам помогут системы управления задачами и сотрудниками — «Битрикс24», SyncCloud и другие.

Как мы уже говорили выше, практически все современные методологии характеризуются, как правило, цикличностью (HADI, цикл Деминга, Agile и т. д.) — то есть, сначала что-то планируем и делаем, потом проверяем, вносим изменения и снова возвращается

**Завершение**

В это время оформляются необходимые закрывающие документы, исполнительная документация, исходные коды с комментариями и т. д.

Для успешного завершения необходимы соответствующие регламенты и инструкции. Это позволит обеспечить быстрый и легкий доступ к данным проекта, даже если люди, которые над ним работали, покинут компанию.

**Резюмируем основные моменты, которые помогают проекту стать успешным и продуктивным**

1. Необходим SMART-план по задачам, структура разделения работ на подзадачи, а также план по ресурсам (люди, деньги, оборудование и т.д.).

2. Планы должны отслеживаться и корректироваться.

3. Должен осуществляться контроль ресурсов и требований к проекту.

4. Задачи должны быть выставлены по приоритетам с помощью матрицы Эйзенхауэра.

5. Нужно делегировать задачи и награждать за качество и исполнительность.

6. Нужно все документировать (исходники, корреспонденция, договоры) и хранить эту документацию.

# **4 Процесс создания программного обеспечения**

Процесс создания программного обеспечения – это множество взаимосвязанных процессов и результатов их выполнения, которые ведут к созданию программного продукта. Процесс создания ПО может начинаться с разработки программной системы "с нуля", но чаще новое ПО разрабатывается на основе существующих программных систем путем их модификации.

Процесс создания ПО, как и любая другая интеллектуальная деятельность, основан на человеческих суждениях и умозаключениях, т.е. является творческим. Вследствие этого все попытки автоматизировать процесс создания ПО имеют лишь ограниченный успех. CASE-средства могут помочь в реализации некоторых этапов процесса разработки ПО, но по крайней мере в ближайшие несколько лет не стоит ожидать от них существенного продвижения в автоматизации тех этапов создания ПО, где существенен фактор творческого подхода к разработке ПО.

Одна из причин ограниченного применения автоматизированных средств к процессу создания ПО – огромное многообразие видов деятельности, связанных с разработкой программных продуктов. Кроме того, организации-разработчики используют разные подходы к разработке ПО. Также различаются характеристики и возможности создаваемых систем, что требует особого внимания к определенным сторонам процесса разработки. Поэтому даже в одной организации при создании разных программных систем могут использоваться различные подходы и технологии.

Несмотря на то что наблюдается огромное многообразие подходов, методов и технологий создания ПО, существуют фундаментальные базовые процессы, без реализации которых не может обойтись ни одна технология разработки программных продуктов. Перечислим эти процессы.

1. *Разработка спецификации ПО.*Это фундамент любой программной системы. Спецификация определяет все функции и действия, которые будет выполнять разрабатываемая система.

2. *Проектирование и реализация (производство) ПО.*Это процесс непосредственного создания ПО на основе спецификации.

3. *Аттестация ПО.*Разработанное программное обеспечение должно быть аттестовано на соответствие требованиям заказчика.

4. *Эволюция ПО.*Любые программные системы должны модифицироваться в соответствии с изменениями требований заказчика.

Хотя не существует "идеального" процесса создания ПО, во многих организациях-разработчиках пытаются его усовершенствовать, поскольку он может опираться на устаревшие технологии и не включать лучших методов современной инженерии программного обеспечения. Кроме того, многие организации постоянно используют одни и те же технологии (когда-то ранее хорошо себя зарекомендовавшие) и им также необходимы методы современной инженерии ПО.

Совершенствовать процесс создания программных систем можно разными путями. Например, путем стандартизации, которая уменьшит разнородность используемых в данной организации технологий. Это, в свою очередь, приведет к совершенствованию внутренних коммуникаций в организации, уменьшению времени обучения персонала и сделает экономически выгодным процесс автоматизации разработок. Стандартизация обычно является первым шагом к внедрению новых методов и технологий инженерии ПО.

## **4.1 Модели процесса создания программного обеспечения**

Модель процесса создания программного обеспечения – это общее абстрактное представление данного процесса. Каждая такая модель представляет процесс создания ПО в каком-то своем "разрезе", используя только определенную часть всей информации о процессе. В настоящем разделе представлены обобщенные модели, основанные на архитектурном подходе. В этом случае можно увидеть всю структуру процесса создания ПО, абстрагируясь от частных деталей отдельных составляющих его этапов.

Эти обобщенные модели не содержат точного описания всех стадий процесса создания ПО. Напротив, они являются полезными абстракциями, помогающими "приложить" различные подходы и технологии к процессу разработки. Кроме того, очевидно, что процесс создания больших систем не является единым, а состоит из множества различных процессов, ведущих к созданию отдельных частей большой системы. Модели создания программного обеспечения.

1. *Каскадная модель.*Основные базовые виды деятельности, выполняемые в процессе создания ПО (такие, как разработка спецификации, проектирование и производство, аттестация и модернизация ПО), представляются как отдельные этапы этого процесса.

2. *Эволюционная модель разработки ПО.*Здесь последовательно перемежаются этапы формирования требований, разработки ПО и его аттестации. Первоначальная программная система быстро разрабатывается на основе некоторых абстрактных общих требований. Затем они уточняются и детализируются в соответствии с требованиями заказчика. Далее система дорабатывается и аттестуется в соответствии с новыми уточненными требованиями.

3. *Модель формальной разработки систем.*Основана на разработке формальной математической спецификации программной системы и преобразовании этой спецификации посредством специальных математических методов в исполняемые программы. Проверка соответствия спецификации и системных компонентов также выполняется математическими методами.

4. *Модель разработки ПО на основе ранее созданных компонентов.*Предполагает, что отдельные составные части программной системы уже существуют, т.е. созданы ранее. В этом случае технологический процесс создания ПО основное внимание уделяет интеграции отдельных компонентов в общее целое, а не их созданию.

Каскадная и эволюционная модели разработки широко используются на практике. Модель формальной разработки систем успешно применялась во многих проектах, но количество организаций-разработчиков, постоянно использующих этот метод, невелико. Использование готовых системных компонентов практикуется повсеместно, но большинство организаций не придерживаются в точности модели разработки ПО на основе ранее созданных компонентов. Вместе с тем этот метод должен получить широкое распространение в XXI столетии, поскольку сборка систем из готовых или ранее использованных компонентов значительно ускоряет разработку ПО.

## **4.2 Спецификация программного обеспечения**

В этом разделе рассматриваются основные базовые процессы создания ПО: формирование спецификации, разработка, аттестация и модернизация программных систем. Первый из этих процессов, формирование спецификации, предназначен для определения сервисов, которыми будет обладать проектируемое ПО, а также ограничений, накладываемых на функциональные возможности и разработку программной системы. Этот процесс в настоящее время обычно называют "разработка требований" (requirements engineering). Разработка требований часто является критическим этапом в создании ПО, поскольку ошибки, допущенные на этом этапе, ведут к возникновению проблем на этапах проектирования и разработки.

Схема процесса разработки требований показана на рис. 4.1. Результатом его выполнения является разработка документации, формализующей требования, предъявляемые к системе, т.е. создание системной спецификации. В этой документации требования обычно представлены на двух уровнях детализации. На самом верхнем уровне представлены требования, определяемые конечными пользователями или заказчиками ПО; но для разработчиков необходима более детализированная системная спецификация.

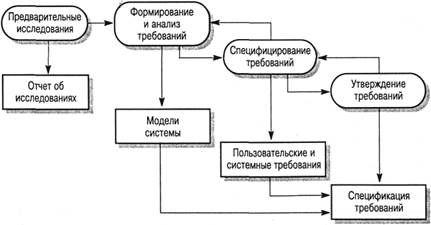


Рисунок 4.1 – Процесс разработки требований

Процесс разработки требований включает четыре основных этапа.

1. *Предварительные исследования.*Оценивается степень удовлетворенности пользователей существующими программными продуктами и аппаратными средствами, а также экономическая эффективность будущей системы и возможность уложиться в существующие бюджетные ограничения при ее разработке. Этот этап должен быть по возможности коротким и дешевым.

2. *Формирование и анализ требований.*Формируются системные требования путем изучения существующих аналогичных систем, обсуждения будущей системы с потенциальными пользователями и заказчиками, анализа задач, которые должна решать система, и т.п. Этот этап может включать разработку нескольких моделей системы и ее прототипов, что помогает сформировать функциональные требования к системе.

3. *Специфицирование требований.*Осуществляется перевод всей совокупности информации, собранной на предыдущем этапе, в документ, определяющий множество требований. Этот документ обычно содержит два типа требований: пользовательские – обобщенные представления заказчиков и конечных пользователей о системе; системные – детальное описание функциональных показателей системы.

4. *Утверждение требований.*Проверяется выполнимость, согласованность и полнота множества требований. В процессе формирования ограничений неизбежно возникновение каких-либо ошибок. На этом этапе они должны быть по возможности выявлены и устранены.

Конечно, процесс разработки требований трудно уложить в описанную последовательность этапов. Например, анализ требований выполняется на протяжении всего процесса их разработки, поэтому внесение новых или изменение уже сформулированных требований возможно на любом этапе. Как правило, этапы разработки требований перекрываются во времени.

## **4.3 Проектирование и реализация программного обеспечения**

Реализация программного обеспечения – это процесс перевода системной спецификации в работоспособную систему. Этап реализации всегда включает процессы проектирования и программирования, но если для разработки ПО применяется эволюционный подход, этап реализации также может включать процесс внесения изменений в системную спецификацию.

На этапе проектирования ПО определяется его структура, данные, которые являются частью системы, интерфейсы взаимодействия системных компонентов и иногда используемые алгоритмы. Проектировщики сразу никогда не получают законченный результат – процесс проектирования обычно проходит через разработку нескольких промежуточных версий ПО. Проектирование предполагает последовательную формализацию и детализацию создаваемого ПО с возможностью внесения изменений в решения, принятые на более ранних стадиях проектирования.

Процесс проектирования может включать разработку нескольких моделей системы различных уровней обобщения. Поскольку проектирование – это процесс декомпозиции, такие модели помогают выявить ошибки, допущенные на ранних стадиях проектирования, а, следовательно, позволяют внести изменения в ранее созданные модели. Все этапы процесса проектирования выполняются последовательно. На практике эти этапы перекрываются вследствие неизбежных обратных связей от одного этапа к предыдущему и повторного выполнения некоторых проектных работ.

Результатом каждого этапа проектирования является спецификация, необходимая для выполнения следующего этапа. Эта спецификация может быть абстрактной и формальной, т.е. такой, какая необходима для детализации системных требований; но она может быть и частью разрабатываемой системы. Так как процесс проектирования непрерывен, спецификации постепенно становятся все более детализированными. Конечными результатами процесса проектирования являются точные спецификации на алгоритмы и *структуры*данных, которые будут реализованы на следующем этапе создания ПО.

Ниже перечислены отдельные этапы процесса проектирования.

1. *Архитектурное проектирование.*Определяются и документируются подсистемы и взаимосвязи между ними.

2. *Обобщенная спецификация.*Для каждой подсистемы разрабатывается обобщенная спецификация на ее сервисы и ограничения.

3. *Проектирование интерфейсов.*Для каждой подсистемы определяется и документируется ее интерфейс. Спецификации на эти интерфейсы должны быть точно выраженными и однозначными, чтобы использование подсистем не требовало знаний о том, как они реализуют свои функции. На этом этапе можно применить методы формальных спецификаций.

4. *Компонентное проектирование.*Проводится распределение системных функций (сервисов) по различным компонентам и их интерфейсам.

5. *Проектирование структур данных.*Детально разрабатываются структуры данных, необходимые для реализации программной системы.

6. *Проектирование алгоритмов.*Детально разрабатываются алгоритмы, предназначенные для реализации системных сервисов.

Описанная схема процесса проектирования является достаточно общей и на практике может (и должна) адаптироваться применительно к разработке конкретного программного продукта. Например, два последних этапа, проектирование структур данных и алгоритмов, могут быть как составными частями процесса проектирования, так и входить в процесс реализации ПО. Если для создания программно4.й системы используются некоторые уже готовые компоненты, это может наложить ограничения на архитектуру системы и интерфейсы системных модулей. Это означает, что количество компонентов, требующих проектирования, значительно уменьшится. Если в процессе проектирования используется метод проб и ошибок, то системные интерфейсы могут разрабатываться после определения структур данных.

## **4.4 Требования к программному обеспечению**

Проблемы, которые приходится решать специалистам в процессе создания программного обеспечения, обычно очень сложны. Природа этих проблем не всегда ясна, особенно если разрабатываемая программная система инновационная. В частности, трудно четко описать те действия, которые должна выполнять система. Описание функциональных возможностей и ограничений, накладываемых на программную систему, называется *требованиями*к этой системе, а сам процесс формирования, анализа, документирования и проверки этих функциональных возможностей и ограничений – *разработкой требований*(requirements engineering). В этой главе внимание концентрируется на самих требованиях и способах их описания. Процесс разработки требований в общих чертах описан в главе 3, а более подробно освещен в следующей главе.

Термин *требования*(к программной системе) может трактоваться по-разному. В некоторых случаях под требованиями понимаются высокоуровневые обобщенные утверждения о функциональных возможностях и ограничениях системы. Другая крайняя ситуация – детализированное математическое формальное описание системных функций. Дэвис (Davis) так объясняет причины этих различий.

Некоторые проблемы, возникающие в процессе разработки требований, порождены отсутствием четкого понимания различия между этими разными уровнями требований. Чтобы различить требования разных уровней, здесь используются термины *пользовательские требования*(user requirements) для обозначения высокоуровневых обобщенных требований и *системные требования*(system requirements) для детализированного описания выполняемых системой функций. Кроме требований этих двух уровней, применяется еще более детализированное описание системы – *проектная системная спецификация*(software design specification), которая может служить мостом между этапом разработки требований и этапом проектирования системы. Три перечисленных вида требований можно определить следующим образом.

1. *Пользовательские требования –*описание на естественном языке (плюс поясняющие диаграммы) функций, выполняемых системой, и ограничений, накладываемых на нее.

2. *Системные требования –*детализированное описание системных функций и ограничений, которое иногда называют функциональной спецификацией. Она служит основой для заключения контракта между покупателем системы и разработчиками ПО.

3. *Проектная системная спецификация*– обобщенное описание структуры программной системы, которое будет основой для более детализированного проектирования системы и ее последующей реализации. Эта спецификация дополняет и детализирует спецификацию системных требований.

Пользовательские требования пишутся для заказчика ПО и для лица, заключающего контракт на разработку программной системы, причем они могут не иметь детальных технических знаний по разрабатываемой системе. Спецификация системных требований предназначена для руководящего технического состава компании-разработчика и для менеджеров проекта. Она также необходима заказчику ПО и субподрядчикам по разработке. Эти оба документа также предназначены для конечных пользователей программной системы. Наконец, проектная системная спецификация является документом, который ориентирован на разработчиков ПО.

## **4.5 Функциональные и нефункциональные требования**

Требования к программной системе часто классифицируются как функциональные, нефункциональные и требования предметной области.

1. *Функциональные требования.*Это перечень сервисов, которые должна выполнять система, причем должно быть указано, как система реагирует на те или иные входные данные, как она ведет себя в определенных ситуациях и т.д. В некоторых случаях указывается, что система не должна делать.

2. *Нефункциональные требования.*Описывают характеристики системы и ее окружения, а не поведение системы. Здесь также может быть приведен перечень ограничений, накладываемых на действия и функции, выполняемые системой. Они включают *временные*ограничения, ограничения на процесс разработки системы, стандарты и т.д.

3. *Требования предметной области.*Характеризуют ту предметную область, где будет эксплуатироваться система. Эти требования могут быть функциональными и нефункциональными.

В действительности четкой границы между этими типами требований не существует. Например, пользовательские требования, касающиеся безопасности системы, можно отнести к нефункциональным. Однако при более детальном рассмотрении такое требование можно отнести к функциональным, поскольку оно порождает необходимость включения в систему средства авторизации пользователя. Поэтому, рассматривая далее эти виды требований, мы должны всегда помнить, что данная классификация в значительной степени искусственна.

## **4.6 Системные требования**

Системные требования – это более детализированное описание пользовательских требований. Они обычно служат основой для заключения контракта на разработку программной системы и поэтому должны представлять максимально полную спецификацию системы в целом. Системные требования также используются в качестве отправной точки на этапе проектирования системы.

Спецификация системных требований может строиться на основе различных системных моделей, таких, как объектная модель или модель потоков данных. Различные модели, используемые при разработке спецификации системных требований.

В принципе системные требования определяют, что должна делать система, не показывая при этом механизма ее реализации. Но, с другой стороны, для полного описания системы требуется детализированная информация о ней, которая по возможности должна включать всю информацию о системной архитектуре. На то существует ряд причин.

1.Первоначальная архитектура системы помогает структурировать спецификацию требований. Системные требования должны описывать подсистемы, из которых состоит разрабатываемая система.

2. В большинстве случаев разрабатываемая система должна взаимодействовать с уже существующими системами. Это накладывает определенные ограничения на архитектуру новой системы.

3. В качестве внешнего системного требования может выступать условие использования для разрабатываемой системы специальной архитектуры.

Спецификации системных требований часто пишутся естественным языком. Использование естественного языка может породить определенные проблемы при написании детализированной спецификации. Применение естественного языка подразумевает, что те, кто пишет спецификацию, и те, кто ее читает, одни и те же слова и выражения понимают одинаково. Однако на самом деле это не так, поскольку естественному языку присуща определенная размытость понятий. Вследствие этого одно и то же требование может трактоваться разными людьми по-разному.

# **5 Верификация и аттестация программного обеспечения**

Верификацией и аттестацией называют процессы проверки и анализа, в ходе которых проверяется соответствие программного обеспечения своей спецификации и требованиям заказчиков. Верификация и аттестация охватывают полный жизненный цикл ПО – они начинаются на этапе анализа требований и завершаются проверкой программного кода на этапе тестирования готовой программной системы.

Верификация и аттестация не одно и то же, хотя их легко перепутать. Кратко различие между ними можно определить следующим образом:

* верификация отвечает на вопрос, правильно ли создана система;
* аттестация отвечает на вопрос, правильно ли работает система.

Согласно этим определениям, верификация проверяет соответствие ПО системной спецификации, в частности функциональным и нефункциональным требованиям. Аттестация - более общий процесс. Во время аттестации необходимо убедиться, что программный продукт соответствует ожиданиям заказчика. Аттестация проводится после верификации, для того чтобы определить, насколько система соответствует не только спецификации, но и ожиданиям заказчика.

На ранних этапах разработки ПО очень важна аттестация системных требований. В требованиях часто встречаются ошибки и упущения; в таких случаях конечный продукт, вероятно, не будет соответствовать ожиданиям заказчика. Но, конечно, аттестация требований не может выявить все проблемы в спецификации требований. Иногда недоработки и ошибки в требованиях обнаруживаются только после завершения реализации системы.

В процессах верификации и аттестации используются две основные методики проверки и анализа систем.

*Инспектирование ПО*. Анализ и проверка различных представлений системы, например документации спецификации требований, архитектурных схем или исходного кода программ. Инспектирование выполняется на всех этапах процесса разработки программной системы. Параллельно с инспектированием может выполняться автоматический анализ исходного кода программ и соответствующих документов. Инспектирование и автоматический анализ - это статические методы верификации и аттестации, поскольку им не требуется исполняемая система.

*Тестирование ПО*. Запуск исполняемого кода с тестовыми данными и исследование выходных данных и рабочих характеристик программного продукта для проверки правильности работы системы. Тестирование - это динамический метод верификации и аттестации, так как применяется к исполняемой системе.

На рис. 5.1 показано место инспектирования и тестирования в процессе разработки ПО. Стрелки указывают на те этапы процесса разработки, на которых можно применять данные методы. Согласно этой схеме, инспектирование можно выполнять на всех этапах процесса разработки системы, а тестирование - в тех случаях, когда создан прототип или исполняемая программа.

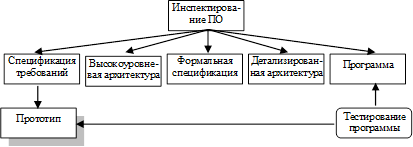


Рисунок 5.1 – Статическая и динамическая верификация и аттестация

К методам инспектирования относятся: инспектирование программ, автоматический анализ исходного кода и формальная верификация. Но статические методы могут прове­рить только соответствие программ спецификации, с их помощью невозможно проверить правильность функционирования системы. Кроме того, статическими методами нельзя проверить такие нефункциональные характеристики, как производительность и надежность. Поэтому для оценивания нефункциональных характеристик проводится тестирование системы.

В настоящее время, несмотря на широкое применение инспектирования ПО, преобладающим методом верификации и аттестации все еще остается тестирование. Тестирование - это проверка работы программ с данными, подобными реальным, которые будут обрабатываться в процессе эксплуатации системы. Наличие в программе дефектов и несоответствий требованиям обнаруживается путем исследования выходных данных и выявления среди них аномальных. Тестирование выполняется на этапе реализации системы (для проверки соответствия системы ожиданиям разработчиков) и после завершения ее реализации.

На разных этапах процесса разработки ПО применяют различные виды тестирования.

*Тестирование дефектов* проводится для обнаружения несоответствий между программой и ее спецификацией, которые обусловлены ошибками или дефектами в программах. Такие тесты разрабатываются для выявления ошибок в системе, а не для имитации ее работы.

*Статистическое тестирование* оценивает производительность и надежность программ, а также работу системы в различных режимах эксплуатации. Тесты разрабатываются так, чтобы имитировать реальную работу системы с реальными входными данными. Надежность функционирования системы оценивается по количеству сбоев, отмеченных в работе программ. Производительность оценивается по результатам измерения полного времени выполнения операций и времени отклика системы при обработке тестовых данных.

Конечно, между этими методами не существует жестких, четко установленных границ. Во время тестирования дефектов испытатель может получить интуитивное, представление о надежности ПО, а во время статистического тестирования есть возможность выявления программных дефектов.

Главная цель верификации и аттестации - удостовериться в том, что система "соответствует своему назначению". Соответствие программной системы своему назначению отнюдь не предполагает, что в ней совершенно не должно быть ошибок. Скорее, система должна достаточно хорошо соответствовать тем целям, для которых планировалась. Уровень необходимой достоверности соответствия зависит от назначения системы, ожиданий пользователей и условий на рынке программных продуктов.

*Назначение ПО*. Уровень достоверности соответствия зависит от того, насколько критическим является разрабатываемое программное обеспечение по тем или иным критериям. Например, уровень достоверности для систем, критическим по обеспечению безопасности, должен быть значительно выше аналогичного уровня достоверности для опытных образцов программных систем, разрабатываемых для демонстрации некоторых новых идей.

*Ожидания пользователей*. Следует с грустью отметить, что в настоящее время у большинства пользователей невысокие требования к программному обеспечению. Пользователи настолько привыкли к отказам, происходящим во время работы программ, что не удивляются этому. Они согласны терпеть сбои в работе системы, если преимущества ее использования компенсируют недостатки. Вместе с тем с начала 1990-х годов терпимость пользователей к отказам в работе программных систем постепенно снижается. В последнее время создание ненадежных систем стало практически неприемлемым, поэтому компаниям, занимающимся разработкой программных продуктов, необходимо все больше внимания уделять верификации и аттестации программного обеспечения.

*Условия рынка программных продуктов*. При оценке программной системы продавец должен знать конкурирующие системы, цену, которую покупатель согласен заплатить за систему, и назначенный срок выхода этой системы на рынок. Если у компании-разработчика несколько конкурентов, необходимо определить дату выхода системы на рынок до окончания полного тестирования и отладки, иначе первыми на рынке могут оказаться конкуренты. Если покупатели не желают приобретать ПО по высокой цене, возможно, они согласны терпеть большее количество отказов в работе системы. При определении расходов на процесс верификации и аттестации необходимо учитывать все эти факторы.

Как правило, в ходе верификации и аттестации в системе обнаруживаются ошибки. Для исправления ошибок в систему вносятся изменения. Этот процесс отладки обычно интегрирован с другими процессами верификации и аттестации. Вместе с тем тестирование (или более обобщенно - верификация и аттестация) и отладка являются разными процессами, которые имеют различные цели.

Верификация и аттестация – процесс обнаружения дефектов в программной системе.

Отладка – процесс локализации дефектов (ошибок) и их исправления (рис. 5.2).

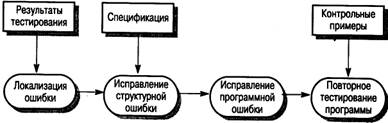


Рисунок 5.2 – Процесс отладки

Простых методов отладки программ не существует. Опытные отладчики обнаруживают ошибки путем сравнения шаблонов тестовых выходных данных с выходными данными тестируемых систем. Чтобы определить местоположение ошибки, необходимы знания о типах ошибок, шаблонах выходных данных, языке программирования и процессе программирования. Очень важны знания о процессе разработке ПО. Отладчикам известны наиболее распространенные ошибки программистов (например, связанные с пошаговым увеличением значения счетчика). Также учитываются ошибки, типичные для определенных языков программирования, например связанные с использованием указателей в языке С.

Определение местонахождения ошибок в программном коде не всегда простой процесс, поскольку ошибка необязательно находится возле того места в коде программы, где произошел сбой. Чтобы локализовать ошибки, программист-отладчик разрабатывает дополнительные программные тесты, которые помогают выявить источник ошибки в программе. Может возникнуть необходимость в ручной трассировке выполнения программы.

Интерактивные средства отладки являются частью набора средств поддержки языка, интегрированных с системой компиляции программного кода. Они обеспечивают специальную среду выполнения программ, посредством которой можно получить доступ к таблице идентификаторов, а оттуда к значениям переменных. Пользователи часто контролируют выполнение программы пошаговым способом, последовательно переходя от оператора к оператору. После выполнения каждого оператора проверяются значения переменных и выявляются возможные ошибки.

Обнаруженная в программе ошибка исправляется, после чего необходимо снова проверить программу. Для этого можно еще раз выполнить инспектирование программы или повторить предыдущее тестирование. Повторное тестирование используется для того, чтобы убедиться, что сделанные в программе изменения не внесли в систему новых ошибок, поскольку на практике высокий процент "исправления ошибок" либо не завершается полностью, либо вносит новые ошибки в программу.

В принципе во время повторного тестирования после каждого исправления необходимо еще раз запускать все тесты, однако на практике такой подход оказывается слишком дорогостоящим. Поэтому при планировании процесса тестирования определяются зависимости между частями системы и назначаются тесты для каждой части. Тогда можно трассировать программные элементы с помощью специальных контрольных примеров (контрольных данных), подобранных для этих элементов. Если результаты трассировки задокументированы, то для проверки измененного программного элемента и зависимых от него компонентов можно использовать только некоторое подмножество всего множества тестовых данных.

## **5.1 Планирование верификации и аттестации**

Верификация и аттестация – дорогостоящий процесс. Для больших систем, например систем реального времени со сложными нефункциональными ограничениями, половина бюджета, выделенного на разработку системы, тратится на процесс верификации и аттестации. Поэтому очевидна необходимость тщательного планирования данного процесса.

Планирование верификации и аттестации, как один из этапов разработки программных систем, должно начинаться как можно раньше. На рисунке 4.3 показана модель разработки ПО, учитывающая процесс планирования испытаний. Здесь планирование начинается еще на этапах создания спецификации и проектирования системы. Данную модель иногда называют V-моделью (чтобы увидеть букву V, необходимо повернуть рис. 5.3 на 90°). На этой схеме также показано разделение процесса верификации и аттестации на несколько этапов, причем на каждом этапе выполняются соответствующие тесты.

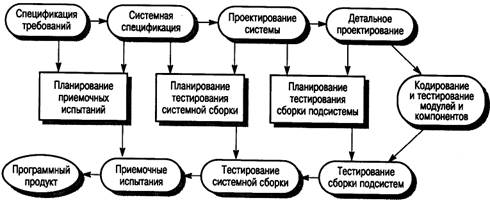


Рисунок 5.3 – Планирование испытаний в процессе разработки и тестирования

В процессе планирования верификации и аттестации необходимо определить соотношение между статическими и динамическими методами проверки системы, определить стандарты и процедуры инспектирования и тестирования ПО, утвердить технологическую карту проверок программ и составить план тестирования программ. Чему уделить больше внимания – инспектированию или тестированию, зависит от типа разрабатываемой системы и опыта организации. Чем более критична система, тем больше внимания необходимо уделить статическим методам верификации.

В плане верификации и аттестации основное внимание уделяется стандартам процесса тестирования, а не описанию конкретных тестов. Этот план предназначен не только для руководства, он в основном предназначен специалистам, занимающимся разработкой и тестированием систем. План дает возможность техническому персоналу получить полную картину испытаний системы и в этом контексте спланировать свою работу. Кроме того, план предоставляет информацию менеджерам, отвечающим за то, чтобы у группы тестирования были все необходимые аппаратные и программные средства.

План испытаний не является неизменным документом. Его следует регулярно пересматривать, так как тестирование зависит от процесса реализации системы. Например, если реализация какой-либо части системы не завершена, то невозможно провести тестирование сборки системы. Поэтому план необходимо периодически пересматривать, чтобы сотрудников, занятых тестированием, можно было использовать на других работах.

## **5.2 Инспектирование программных систем**

Системное тестирование программ требует разработки огромного количества тестов, их выполнения и проверки. Это значит, что данный процесс достаточно трудоемкий и дорогостоящий. Каждый тест позволяет обнаруживать одну, а в лучшем случае несколько ошибок в программе. Причина такого положения заключается в том, что сбои в работе, происходящие из-за ошибок в системе, часто приводят к разрушению данных. Поэтому трудно сказать, какое количество ошибок "ответственно" за сбой в системе.

Инспектирование программ не требует их исполнения, поэтому данный метод можно использовать до завершения полной реализации программ. Во время инспектирования проверяется исходное представление системы. Это может быть модель системы, спецификация или программа, написанная на языке высокого уровня. Для обнаружения ошибок используется знание разрабатываемой системы и семантика ее исходного представления. Каждую ошибку можно рассматривать отдельно, не обращая внимания на то, как она влияет на поведение системы.

Доказано, что инспектирование является эффективным методом обнаружения ошибок. Также немаловажно, что инспектирование значительно дешевле экстенсивного тестирования программ. В экспериментах, сравнивалась эффективность инспектирования и тестирования. Инспектирование программного кода оказалось более эффективным и менее дорогостоящим, чем тестирование. Более 60% ошибок в программах можно обнаружить с помощью неформального исследования (инспектирования) программ. При более формальном подходе, использующем математические методы, в программе можно обнаружить более 90% всех ошибок. Такая проверка используется в процессе разработки систем методом "чистая комната". Процесс инспектирования также может оценить другие качественные характеристики систем: соответствие стандартам, переносимость и удобство сопровождения.

В системных компонентах и подсистемах выявление ошибок путем просмотра и инспектирования обычно более эффективно, чем с помощью тестирования, по двум причинам.

1. За один сеанс инспектирования можно выявить множество разнообразных программных дефектов. Недостатком тестирования является то, что обычно за один сеанс тестирования можно обнаружить только одну ошибку, поскольку ошибки могут привести к отказу системы или их эффекты могут накладываться друг на друга.

2. Инспектирование использует знания о предметной области и языке программирования. Специалист, проводящий инспектирование, должен знать типы ошибок, присущие конкретным языкам программирования и приложениям определенного типа. Поэтому в ходе анализа программ есть возможность сосредоточиться только на конкретных типах ошибок.

Конечно, инспектирование не может полностью заменить тестирование. Инспектирование лучше использовать как начальный процесс верификации для обнаружения большей части программных дефектов. Путем инспектирования проверяют соответствие ПО ее спецификации, однако таким способом нельзя проверить динамическое поведение системы. Более того, нерационально инспектировать законченные системы, собранные из нескольких подсистем. На этом уровне возможно только тестирование. Тестирование также необходимо для оценки надежности и производительности, проверки пользовательского интерфейса и соответствия системы требованиям заказчика.

Инспектирование и тестирование не являются конкурирующими методами верификации и аттестации. Каждому из них присущи свои преимущества и недостатки, поэтому в процессе верификации и аттестации их следует использовать совместно. Одним из наиболее эффективных методов инспектирования является применение контрольных примеров. В этом случае можно обнаружить программные дефекты и разработать более эффективные методы тестирования системы.

Иногда при инспектировании в организации, разрабатывающей традиционное программное обеспечение, возникают трудности. Разработчики, имеющие опыт тестирования программ, неохотно соглашаются с тем, что инспектирование оказывается более эффективным методом выявления ошибок, чем тестирование. Менеджеры относятся к этим технологиям с недоверием, потому что внедрение инспектирования на этапах проектирования и разработки требует дополнительных расходов. Инспектирование всегда требует расходов, причем на начальном этапе разработки ПО, а конечная экономия средств вследствие применения инспектирования достигается только благодаря опыту проводящих его специалистов.

## **5.3 Проверка программного обеспечения**

Это подразумевает проверку соответствия спецификаций при запуске программного обеспечения, но это невозможно (например, как кто-нибудь может узнать, правильно ли реализована архитектура / дизайн / и т. Д. При запуске программного обеспечения?). Только просмотрев связанные с ним артефакты, кто-то может сделать вывод о том, выполнены ли спецификации.

Проверка программного обеспечения проверяет, что программный продукт удовлетворяет или соответствует предполагаемому использованию (высокоуровневая проверка), т. Е. Программное обеспечение соответствует требованиям пользователя, а не артефактам спецификации или потребностям тех, кто будет эксплуатировать только программное обеспечение; но, как и потребности всех заинтересованных сторон (таких как пользователи, операторы, администраторы, менеджеры, инвесторы и т. д.). Существует два способа проверки программного обеспечения: внутренний и внешний. Во время внутренней проверки программного обеспечения предполагается, что цели заинтересованных сторон были правильно поняты и что они были точно и подробно изложены в артефактах требований. Если программное обеспечение соответствует требованиям спецификации, оно прошло внутреннюю проверку. Внешняя проверка происходит, когда она выполняется путем запроса заинтересованных сторон, соответствует ли программное обеспечение их потребностям. Различные методологии разработки программного обеспечения требуют разных уровней участия пользователей и заинтересованных сторон и обратной связи; Итак, внешняя проверка может быть дискретным или непрерывным событием. Успешная окончательная внешняя проверка происходит, когда все заинтересованные стороны принимают программный продукт и заявляют, что он удовлетворяет их потребности. Такая окончательная внешняя проверка требует использования приемочный тест, который является динамическим тестом.

Однако можно также выполнить внутренние статические тесты, чтобы выяснить, соответствует ли оно спецификации требований, но это входит в область статической проверки, поскольку программное обеспечение не работает.

## **5.4 Проверка артефакта или спецификации**

Выходные данные каждого этапа процесса разработки программного обеспечения также могут подлежать проверке при проверке по их входной спецификации.

Примеры проверки артефактов:

* Из спецификации проекта в соответствии со спецификацией требований: правильно ли спецификации архитектурного проекта, детального проектирования и логической модели базы данных соответствуют функциональным и нефункциональным спецификациям требований?
* Из конструктивных артефактов против спецификации проекта: правильно ли исходный код, пользовательские интерфейсы и физическая модель базы данных соответствуют спецификации проекта?

Требования должны быть проверены до того, как программный продукт в целом будет готов (процесс разработки водопада требует, чтобы они были полностью определены до начала проектирования; но итеративные процессы разработки не требуют этого и допускают их постоянное улучшение).

Примеры проверки артефактов:

* Проверка спецификации требований пользователя. Требования пользователя, указанные в документе под названием «Спецификация требований пользователя», проверяются путем проверки, действительно ли они отражают волю и цели заинтересованных сторон. Это можно сделать, опросив их и задав их напрямую (статическое тестирование) или даже выпустив прототипы и попросив пользователей и заинтересованных лиц оценить их (динамическое тестирование).
* Пользовательский ввод проверка: Пользовательский ввод (собранно любой периферийным , такой как клавиатура, био~d-метрики датчик, т.д.) , подтвержденный путем проверки , если вход обеспечивается операторами программного обеспечения или пользователей удовлетворять правила домена и ограничение (например, типа данных, диапазон и формат).

**Список литературы**

1. Портни Стэнли И. – Управление проектами для "чайников"
2. Карпова С.О. – Конспект лекцій з курсу “Основи програмної інженерії”. Херсон: ХФ НУК, 2014.- 285 с
3. Купман, П. "Надежность, безопасность и надежность в повседневных встроенных системах".