ОРГАНИЗАЦИЯ ПРОЦЕССА РАЗРАБОТКИ

Основные понятия программной инженерии

Существуют два понятия «программа» и «программное обеспечение». Государственный стандарт 19781-90 и международный стандарт ISO/ IEC 2382/1-93 определяют, что ПО включает в себя не только программы, но п всю сопутствующую документацию. а также конфигурационные данные, необходимые для правильной работы программ. Чтобы подчеркнуть наличие многих элементов и отдать дань сложности ПО, его часто называют программной системой (ПС). Итак, программные системы состоят из совокупности программ, файлов конфигурации, необходимых для установки этих программ, и документации, которая описывает организацию системы и объясняет пользователям порядок работы с системой.

Программный проект (project) — это временное предприятие, предназначенное для создания уникальных продуктов, услуг или результатов. Временный характер проекта подчеркивает, что у любого проекта есть определенное начало и завершение. Завершение наступает, когда достигнуты цели проекта или признано, что цели проекта не могут быть достигнуты или исчезла необходимость в проекте. Характеристика «временный», как правило, не относится к создаваемому в ходе проекта продукту, услуге или результату. В состав программного проекта входят как люди (разработчики), так и необходимые материальные ресурсы.

Основные понятия программной инженерии

Цель проекта определяет задачи, решаемые в результате реализации проекта, *содержание* — что является результатом проекта. *Управление проектом* определяет, как будет достигнута цель и создан необходимый результат. Управление проектом есть постоянная деятельность, осуществляемая на всем его протяжении по определенной методике. Достижение результата проекта и достижение его цели — разные вещи. Например, результат проекта — разработка информационной системы, цель — автоматизация определенной деятельности. Можно создать информационную систему, не обладающую нужной функциональностью, и цель проекта достигнута не будет, в отличие от результата. Успешным проект будет лишь тогда, когда результат соответствует заданному содержанию и его целям.

Масштаб проекта оценивается по количеству людей, в нем задействованных, и по конечной стоимости. Любой проект, как правило, имеет руководителя (проектного менеджера), который несет ответственность за проект в целом.

Важны следующие аспекты управления проектами:

- ограничения являются следствием приоритетов, расставленных заказчиком с учетом имеющихся ресурсов;
- приоритеты заказчика и исполнителя в общем случае могут не совпадать (противоречить);
- анализ компромиссов определяет баланс приоритетов сторон, заинтересованных в проекте;
- ограничения являются неотъемлемой частью проекта;
- ограничения порождают риски;
- ограничения рассматриваются в контексте уровня детализации проекта.

Основные понятия программной инженерии

Детализация проекта заключается в разбиении (декомпозиции) его на отдельные работы (задачи); она необходима для адекватной оценки его длительности и стоимости при имеющихся требованиях к функциональности и уровню качества.

В области управления проектами существует ряд нормативов и рекомендаций, среди которых можно выделить результаты деятельности следующих организаций:

- PMI — Project Management Institute, Inc. — американский Институт управления проектами;
- IPMA — International Project Management Association — Международная ассоциация управления проектами;
- APM — Association of Project Management — одна из крупнейших независимых профессиональных ассоциаций по проектному менеджменту в Европе;
- Ассоциация по управлению проектами COBNET.

Проект с позиций программной инженерии (IT-проект) — это совокупность действий, необходимых для создания артефактов программного продукта. Проект включает в себя взаимодействие с заказчиком, написание документации, проектирование, кодирование и тестирование.

Основные понятия программной инженерии

Особенностью IT-проекта является еще и то, что все этапы его реализации должны сопровождаться специальными документами:

- **SQAP (Software Quality Assurance Plan)** — *план контроля качества* программного обеспечения, определяющий, каким образом проект должен достигнуть соответствия установленному уровню качества.
- **SCMP (Software Configuration Management Plan)** — ***план управления конфигурациями* программного обеспечения, который определяет**, как и где должны храниться документы, программный код и их версии, устанавливает их взаимное соответствие.
- **SPMP (Software Project Management Plan)** — *план управления программным проектом*, который определяет, каким образом следует управлять проектом.
- **SRS (Software Requirements Plan)** — *спецификация требований* к программному обеспечению, которая определяет требования к приложению, утверждается совместно заказчиком и разработчиком и служит отправной точкой для реализации функциональности программного продукта.
- **SDD (Software Design Document)** — *проектная документация* программного обеспечения, которая представляет архитектуру и детали проектирования приложения с использованием диаграмм.
- **STD (Software Test Documentation)** — *документация по тестированию* программного обеспечения, в которой указывается, каким образом должно проводиться тестирование приложения и его компонентов.

Основные понятия программной инженерии

При реализации IT-проекта важно следующее:

- грамотное *управление персоналом*: адекватный подбор участников проекта с обозначением для каждого его должностных обязанностей, указанием круга взаимодействия с коллегами и зоны ответственности;
- умение *управлять рисками:* все риски должны быть как можно раньше идентифицированы; устранимые риски необходимо предотвратить, например модифицируя требования, а неустранимые — преодолеть, применяя определенные технологии программирования и/или проектирования.

Основные понятия программной инженерии

Термин ***программная инженерия*** или ***инженерия программного обеспечения*** впервые был использован в 1968 году в качестве темы конференции, посвященной вопросам максимальной загрузки самых мощных (по тем временам) компьютеров. Там же было дано определение этого термина, не утратившее своей актуальности и в настоящее время:

программная инженерия (инженерия программного обеспечения) - система инженерных принципов для создания ПО, которое надежно и эффективно работает в реальных компьютерах.

Позже появился международный терминологический стандарт ISO/1EC 2382/1-93, который дает более развернутую формулировку:

программная инженерия - систематическое применение научных и технологических знаний, методов и практического опыта к проектированию, реализации, тестированию и документированию программного обеспечения в целях оптимизации его производства, поддержки и качества.

Основные понятия программной инженерии

Буквальный русский перевод термина ***software engineering*** нельзя назвать шедевром: звучит он как-то не по-русски. Поэтому долгое время в русскоязычном пространстве бытовал термин ***технология разработки ПО***. Однако такой перевод сужал рамки понятия, оставляя за бортом многие аспекты дисциплины. Именно поэтому в национальных стандартах утвердился перевод ***программная инженерия.***

Программная инженерия — сравнительно молодая дисциплина, ее возраст чуть больше сорока лет. Первые двадцать лет (70-80-е годы) в ней доминировал классический, процедурный подход, а во вторые двадцать лет (1990-2000-е годы) — объектно-ориентированный подход к разработке ПО.

Различают методы, средства и процессы программной инженерии.

Основные понятия программной инженерии

Различают методы, средства и процессы программной инженерии:

Методы обеспечивают решение широкого спектра технических задач; например, назовем следующие задачи разработки:

- планирование и оценка программного проекта;
- анализ требований к компьютерной системе в целом и к программному обеспечению в частности;
- проектирование структур программ (и структур данных), входящих в состав ПО;
- конструирование программного текста (другие названия: кодирование, программирование, реализация);
- тестирование (выявление ошибок в созданных программах);
- сопровождение ПО, уже используемого заказчиками.

Средства (утилиты) программной инженерии обеспечивают автоматизированную или автоматическую поддержку методов. В целях совместного применения утилиты могут объединяться в системы автоматизированной разработки ПО. Такие системы принято называть CASE-системами. Аббревиатура CASE расшифровывается как Computer Aided Software Engineering (программная инженерия с компьютерной поддержкой).

Процессы являются «клеем», который соединяет методы и утилиты так, что они обеспечивают непрерывную технологическую цепочку разработки. Процессы определяют:

- порядок применения методов и утилит;
- формирование отчетов, форм по соответствующим требованиям;
- контроль, который помогает обеспечивать качество и координировать изменения;
- формирование «вех», по которым руководители оценивают прогресс.

Основные понятия программной инженерии

Обычно при разработке программного обеспечения придерживаются некоторой последовательности.

В рамках каждого этапа последовательности выделяют виды деятельности (этапа или процесса), действия и задачи.

Деятельность – это процесс ориентированный на достижение весомой цели (например, обеспечение взаимодействия с заинтересованными в проекте лицами) и применятся не зависимо от прикладной области, размера проекта, сложности…Деятельность состоит из действий

Действия – это набор задач, которые производят этапный рабочий продукт (например, модель результатов проектирования)

Задача – небольшой процесс, имеющий хорошо определенную цель, который приводит к ощутимому результату (например, проведение тестирования модуля)

Чаще всего последовательность действий применяемую при разработки ПО называют моделью процесса разработки. Модель процесса разработки – это адаптивное руководство, позволяющее выполнять работу, указывая или выбирая подходящий набор действий и задач. Целю – создавать ПО за приемлемое время и с достаточным качеством.

Основные понятия программной инженерии

Наиболее крупные и распространенные виды деятельности, не зависящие от размеров проекта:

Подготовка. Сотрудничество с заказчиком и другими заинтересованными лицами. Определение целей и задач проекта. Сбор требований…

Планирование. Последовательность шагов (план) в процессе создания ПО. План описывает задачи, которые надо выполнить, факторы риска, расписание работы…

Моделирование. Модели позволяют лучше понять общую картину – эскиз будущего решения. Обычно моделирование включает в себя два этапа: анализ и проектирование. Модель анализа улучшает понимание требований к ПО, модель проектирования показывает эскиз структуры и поведения ПО.

Конструирование. Деятельность по генерации программного кода и тестирование.

Развертывание. Поставка ПО заказчику

Термины и определения

Артефакт — неотъемлемая часть результата и процесса выполнения программного проекта, реализованная в виде документации, программного кода (исходного или скомпилированного) или его части (например, модуля).
Архитектура программного обеспечения — модель программного обеспечения на самом высоком уровне, формализованная теми или иными средствами.
Двоичный код — переведенный на понятный ЭВМ язык (машинный язык) исходный код программы.
Детальное проектирование — создание подробной модели разрабатываемой программной системы (с применением псевдокода, блок-схем и др.), достаточной для написания программного кода.
Инспектирование — коллективное исследование артефактов проекта, направленное на выявление дефектов, осуществляемое, как правило, лицами (разработчиками или их группами), не участвовавшими в их создании.
Исходный код — программный код, написанный на каком-либо языке программирования.
Качество программного обеспечения — совокупность наиболее важных характеристике программного продукта (например, надежность), а также характеристики самого процесса разработки (например, количество дефектов на тысячу строк кода), измеряемые количественно по тем или иным методикам, на основе которых можно сделать заключение о соответствии продукта и/или процесса заранее определенным показателям.
Класс — структура, описывающая группу объектов с одинаковыми свойствами (атрибутами), одинаковым поведением (операциями), ч ипами отношений (относительно объектов других классов) и семантикой.
Модуль — программная единица, обладающая открытой (интерфейс) и закрытой (реализация) частями, выполняющая ряд функций и подключаемая в основной программе в процессе написания кода.

Термины и определения

Модуль — программная единица, обладающая открытой (интерфейс) и закрытой (реализация) частями, выполняющая ряд функций и подключаемая в основной программе в процессе написания кода.

Объектный код —- переведенный на машинный язык исходный код, еше не подверженный особой упаковке, характерной для конкретной операционной системы.

Приложение — готовый программный продукт; как правило, приложением называют программный продукт, разработанный для операционной системы Windows.

Программный проект — процесс реализации комплекса мероприятий, направленных на создание программного продукта.

Программный продукт — результат реализации программного проекта, обладающий заявленной функциональностью и потребительскими характеристиками.

Проектирование программного обеспечения — создание модели программного обеспечения с применением тех или иных средств, языков и стандартов, уровень детализации которой находится между архитектурой программного обеспечения и детальным проектом.

Прототип — программный продукт, по ряду ключевых на данный момент характеристик близкий к разрабатываемому. Прототип предназначен для демонстрации проекта заказчику с целью определения его мнения относительно интерфейса или части реализованной функциональности. Большей частью прототипы используют для своевременной корректировки требований к программному продукту в процессе разработки.

Термины и определения

Сопровождение программного обеспечения — комплекс работ, выполняемых после поставки программного продукта заказчику, направленный, главным образом, на исправление обнаруженных в процессе эксплуатации дефектов и расширение возможностей программного обеспечения.

Тестирование — комплекс мероприятий, направленный на выявление дефектов в готовом программном обеспечении и/или его составляющих.

Требования к программному обеспечению — документ, отражающий, что должно делать разрабатываемое программное обеспечение.

Управление конфигурациями программного обеспечения — поддержание соответствия версий всех артефактов создаваемого программного продукта в процессе разработки.

Понятие жизненного цикла ПП

Появление понятия *жизненного цикла* (ЖЦ) программного продукта (ПП) было связано с кризисом программирования, который наметился в конце 60-х — начале 70-х гг. прошлого века. Суть кризиса состояла в том, что программные проекты все чаще стали выходить из-под контроля: нарушались сроки, превышались запланированные объемы финансирования, результаты не соответствовали требуемым. Многие проекты вообще не доводились до завершения. Ситуация была вызвана ростом сложности проектов, и масштабы ее нарастали, необходимо было принимать меры для радикального усовершенствования принципов и методов разработки ПО с учетом его развития и сопровождения.

Методологическую основу промышленной инженерии составляет понятие жизненного цикла изделия (продукта) как совокупности всех действий, которые надо выполнить на протяжении всей «жизни» изделия.

Каждая программная система (ПС) на протяжении своего существования проходит определенную последовательность процессов (этапов), начиная от постановки задачи до ее воплощения в готовую программу, а затем через эксплуатацию — к изъятию. Такая последовательность этапов называется *жизненным циклом* разработки ПС. На каждом этапе ЖЦ выполняется определенная совокупность процессов и/или подпроцессов, каждый из которых порождает соответствующий промежуточный продукт, используя результаты предыдущего.

Организация промышленного производства с позиции жизненного цикла позволяет рассматривать все его этапы во взаимосвязи, что

Понятие жизненного цикла ПП

Впервые о жизненном цикле ПО заговорили в 1968 г. в Лондоне, где состоялась встреча 22 руководителей проектов по разработке ПО. На встрече анализировались проблемы и перспективы проектирования, разработки, распространения и поддержки программ. Применяющиеся принципы и методы разработки ПО требовали постоянного усовершенствования. Именно на этой встрече была предложена концепция жизненного цикла ПО (SLC — Software Lifetime Cycle) как последовательности шагов-стадий, которые необходимо выполнить в процессе создания и эксплуатации ПО.

В 1970 г. У. У. Ройс (W.W. Royce) произвел идентификацию нескольких стадий в типичном цикле и было высказано предположение, что контроль выполнения стадий приведет к повышению качества ПО и сокращению стоимости разработки.

Все продукты процессов программной инженерии представляют собой некоторые описания, а именно тексты требований к разработке, согласования договоренностей с заказчиком, описания архитектуры и структуры данных, тексты программ, документацию, инструкции по эксплуатации и т.п.

Понятие жизненного цикла ПП

Главными ресурсами разработки ПС в программной инженерии являются сроки, время и стоимость. Правильное использование этих ресурсов на процессах ЖЦ определяет эффективность этой разработки.

Существует общее соглашение о выделении четырех обобщенных фаз жизненного цикла:

- концепция (инициация, идентификация, отбор);
- определение (анализ);
- выполнение (практическая реализация или внедрение, производство и развертывание, проектирование или конструирование, сдача в эксплуатацию, инсталляция, тестирование и т.п.);
- закрытие (завершение, включая оценивание после завершения).

Понятие жизненного цикла ПП

Однако эти фазы столь широко определены, что необходимы более конкретные определения, может быть выделение пяти — десяти основных фаз для каждой категории и подкатегории проекта, возможно с выделением внутри каждой фазы нескольких подфаз.

В общем случае жизненный цикл определяется моделью и описывается в форме методологии (метода). Модель (парадигма) жизненного цикла определяет концептуальный взгляд на его организацию и основные фазы, а также принципы перехода между ними.

Методология жизненного цикла задает комплекс работ, их детальное содержание и ролевую ответственность спецмалистов на всех этапах выбранной модели ЖЦ, обычно определяет и саму модель, а также рекомендует практики (best practices), позволяющие максимально эффективно воспользоваться соответствующей методологией и ее моделью.

Понятие жизненного цикла ПП

В индустрии программного обеспечения можно и необходимо (для обеспечения возможности управления) четко разграничивать фазы проекта (что не подразумевает их линейного и последовательного выполнения).

В определенном контексте «модель» и «методология» могут использоваться взаимозаменяемым образом, например при обсуждении разграничения фаз проекта. Говоря «жизненный цикл», мы в первую очередь подразумеваем «модель жизненного цикла». Несмотря на данное в стандартах определение модели жизненного цикла, все же при употреблении слова «модель» чаще подразумевается именно общий принцип организации ЖЦ, чем детализация соответствующих работ. Соответственно, при определении и выборе модели в первую очередь приходится решать вопросы определенности и стабильности требований, жесткости и детализированности плана работ, а также частоты сборки работающих версий создаваемой программной системы.

Классификация процессов программной инженерии

Классификацию процессов программной инженерии задают международный стандарт ISO/IEC 12207-2008 «Information Technology — Software Life Cycle Processes» и его российский аналог ГОСТ Р ИСО/МЭК 12207-2010.

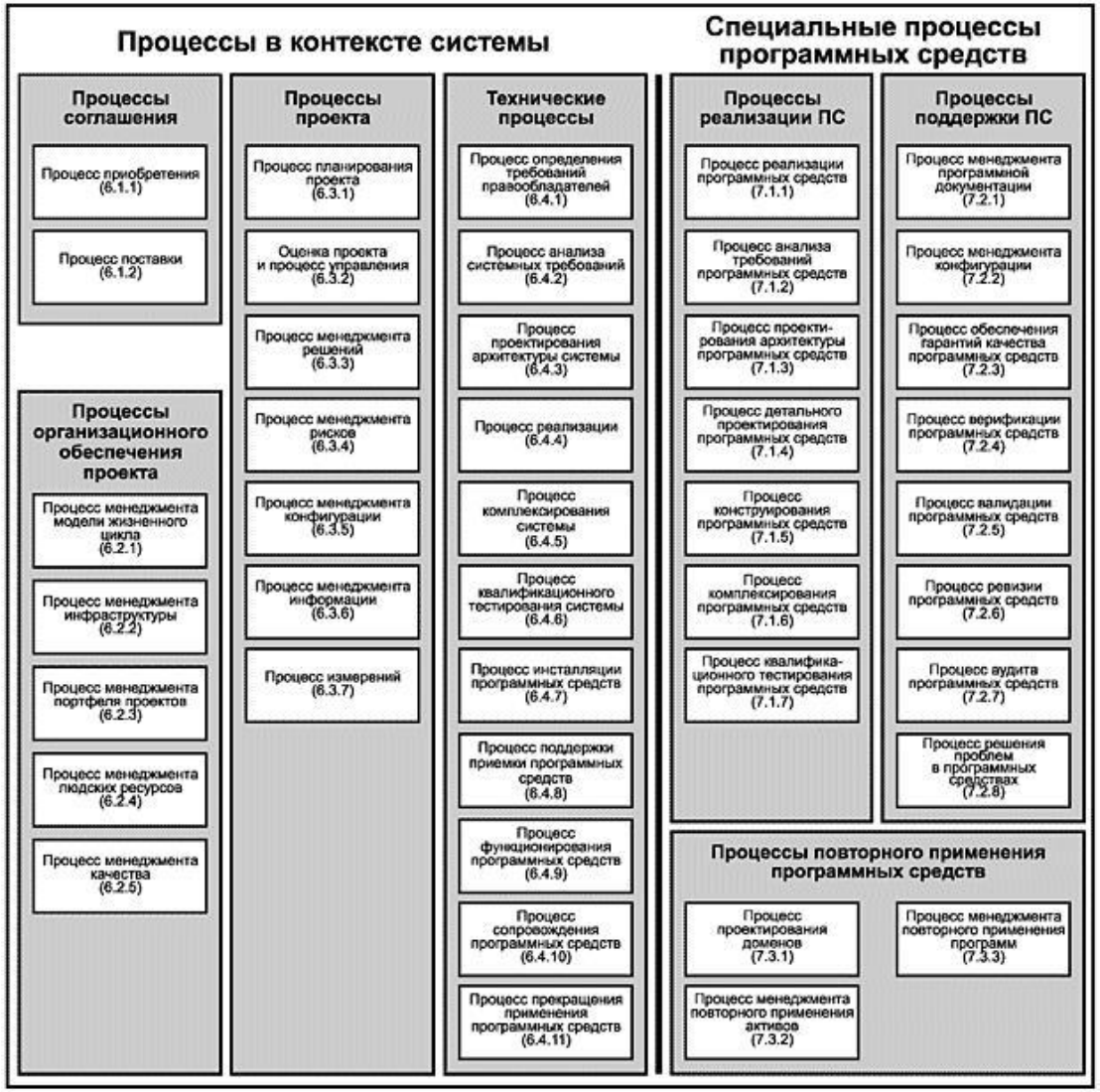
В этих стандартах процессы привязаны к основному понятию программной инженерии — жизненному циклу программного обеспечения (ЖЦ ПО). В авторитетном словаре программной инженерии ISO/IECIEEE 24765-2010 “Systems and Software Engineering - Vocabulary” записано: жизненный цикл программного обеспечения определяется как период времени, который начинается с момента принятия решения о необходимости создания ПО и заканчивается в момент его полного изъятия из эксплуатации.

Действия, которые могут выполняться в ЖЦ ПО, распределены по двум категориям:

Процессы в контексте системы - процессы для работы с автономным программным продуктом

Специальные процессы программных средств – содержит процессы в отношении программного продукта, являющегося частью более крупной системы

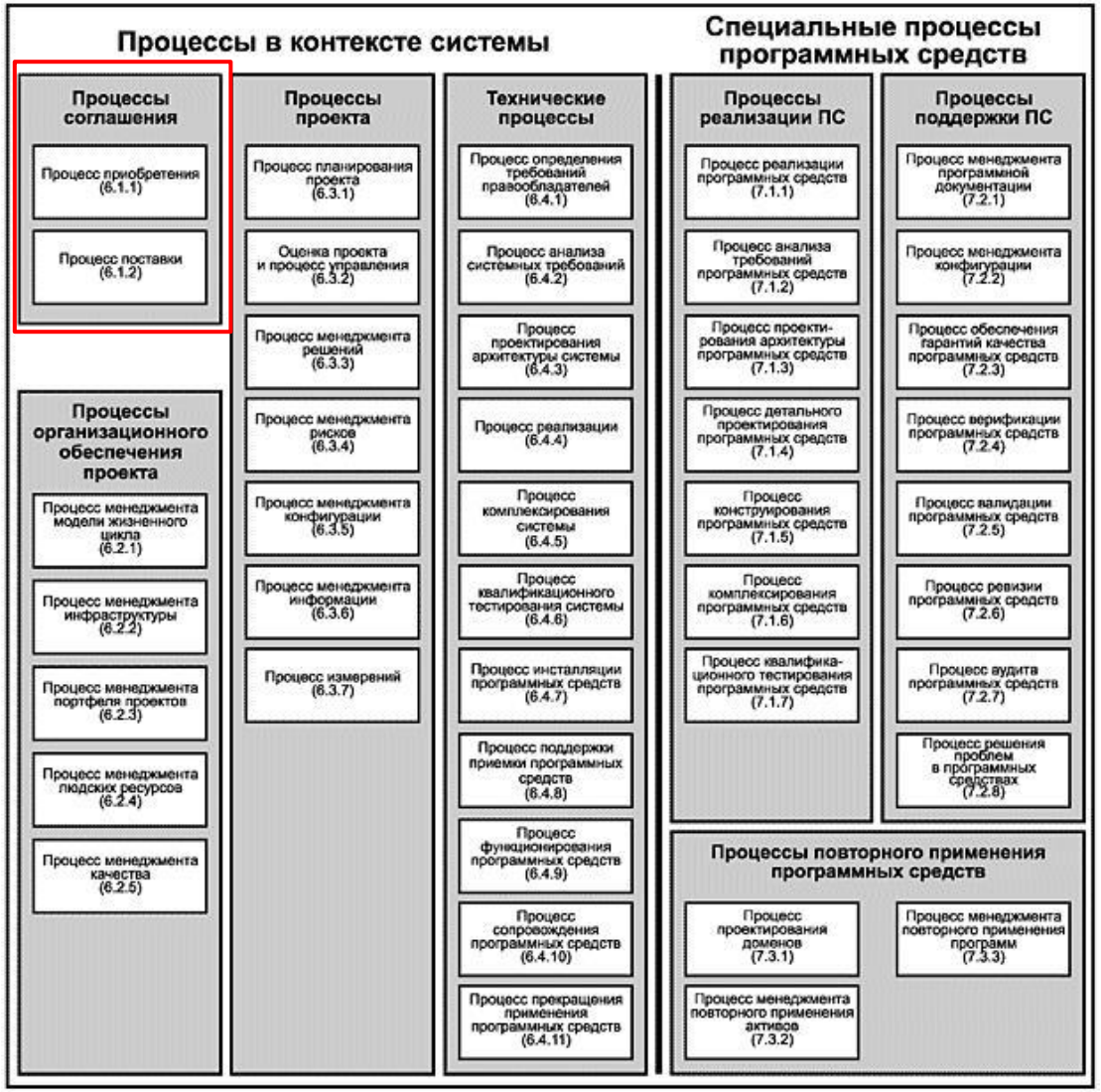
Классификация процессов программной инженерии



Эталонная модель процесса не представляет конкретного подхода к осуществлению процесса, как и не предопределяет модель жизненного цикла системы (программного средства), методологию или технологию. Вместо этого эталонная модель предназначается для принятия организацией и базируется на деловых потребностях организации и области приложений. Определенный организацией процесс принимается в проектах организации в контексте требований заказчиков.

Результаты процесса используются для демонстрации успешного достижения цели процесса, что помогает оценщикам процесса определять возможности реализованного процесса организации и предоставлять исходные материалы для планирования улучшений организационных процессов.

Классификация процессов программной инженерии

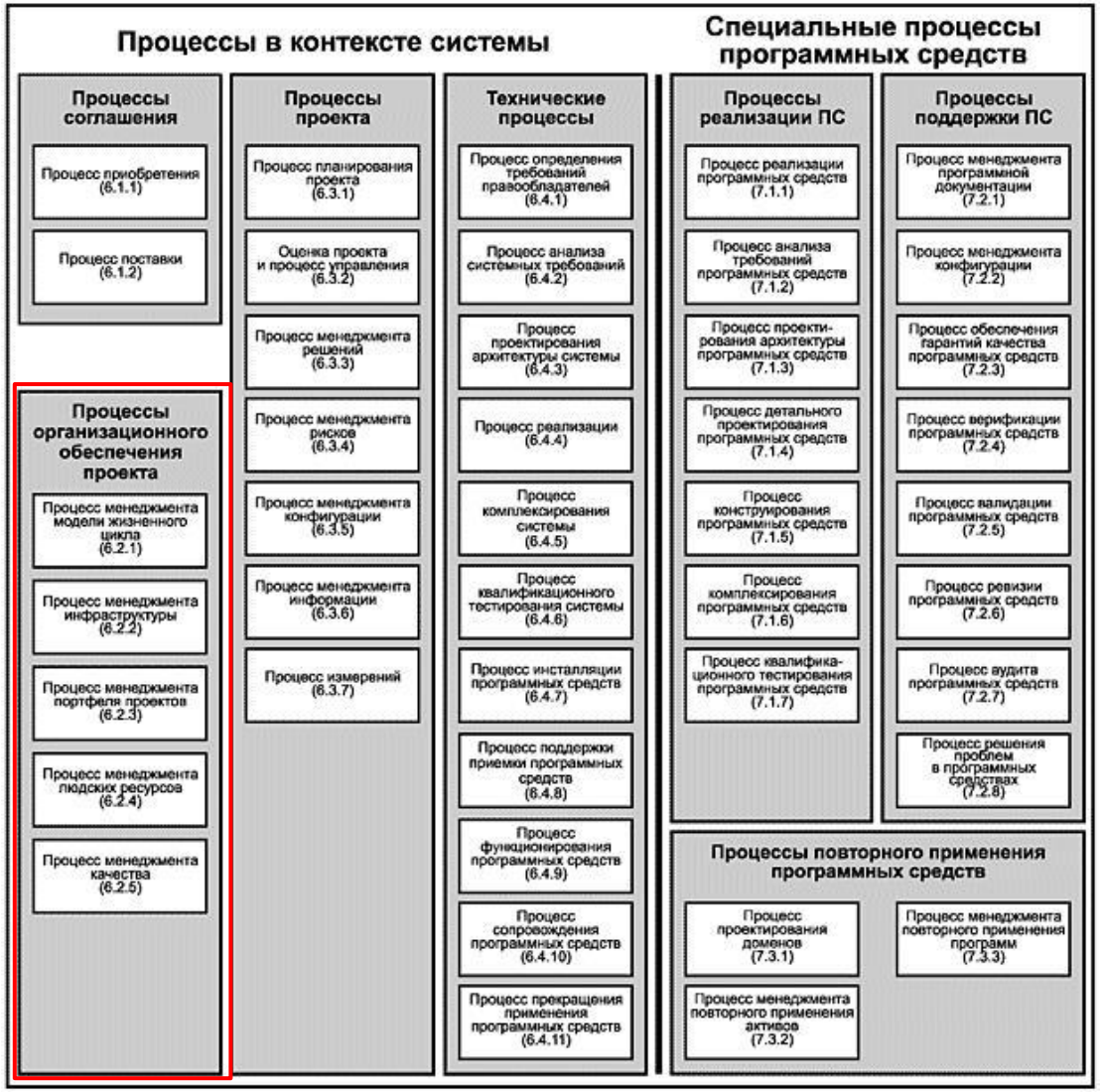


Процессы соглашения.

Процессы соглашения определяют действия, необходимые для выработки соглашений между двумя организациями. Если реализуется процесс приобретения, то он обеспечивает средства для проведения деловой деятельности с поставщиком продуктов, предоставляемых для применения в функционирующей системе, услугах поддержки этой системы или элементах системы, разработанных в рамках проекта. Если реализуется процесс поставки, то он обеспечивает средства для проведения проекта, в котором результатом является продукт или услуга, поставляемые приобретающей стороне.

Таким образом, процессы соглашения, приведенные в настоящем стандарте, ориентированы на программные средства процессами соглашения из **ИСО/МЭК 15288:2007 Системная инженерия. Процессы жизненного цикла систем.**

Классификация процессов программной инженерии

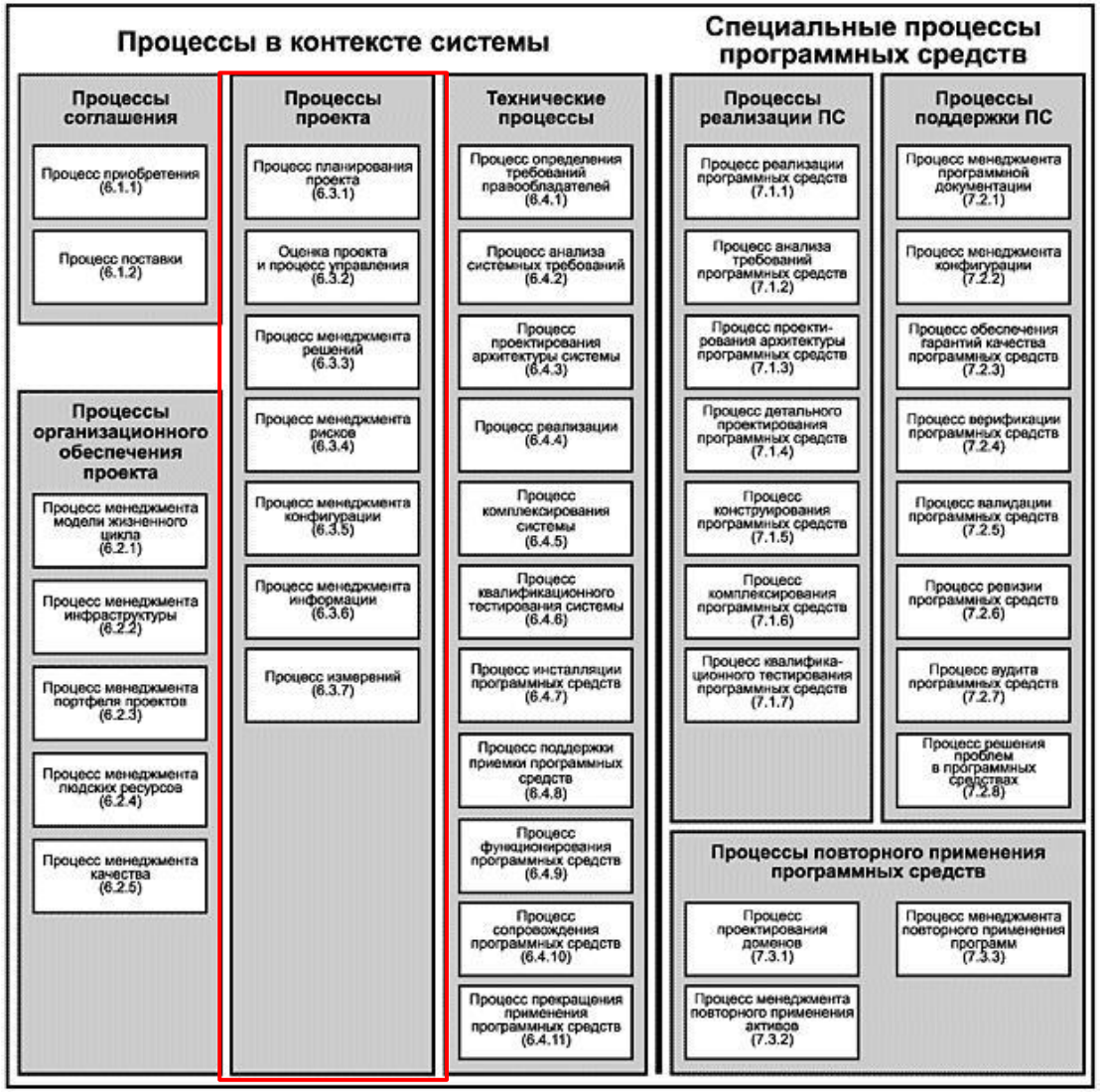


Процессы организационного обеспечения проекта

Процессы организационного обеспечения проекта осуществляют менеджмент возможностей организаций приобретать и поставлять продукты или услуги через инициализацию, поддержку и управление проектами. Эти процессы обеспечивают ресурсы и инфраструктуру, необходимые для поддержки проектов, и гарантируют удовлетворение организационных целей и установленных соглашений. Они не претендуют на роль полной совокупности деловых процессов, реализующих менеджмент деловой деятельности организации. Процессы организационного обеспечения проекта включают в себя:

a) процесс менеджмента модели жизненного цикла;
b) процесс менеджмента инфраструктуры;
c) процесс менеджмента портфеля проектов;
d) процесс менеджмента людских ресурсов;
e) процесс менеджмента качества.

Классификация процессов программной инженерии

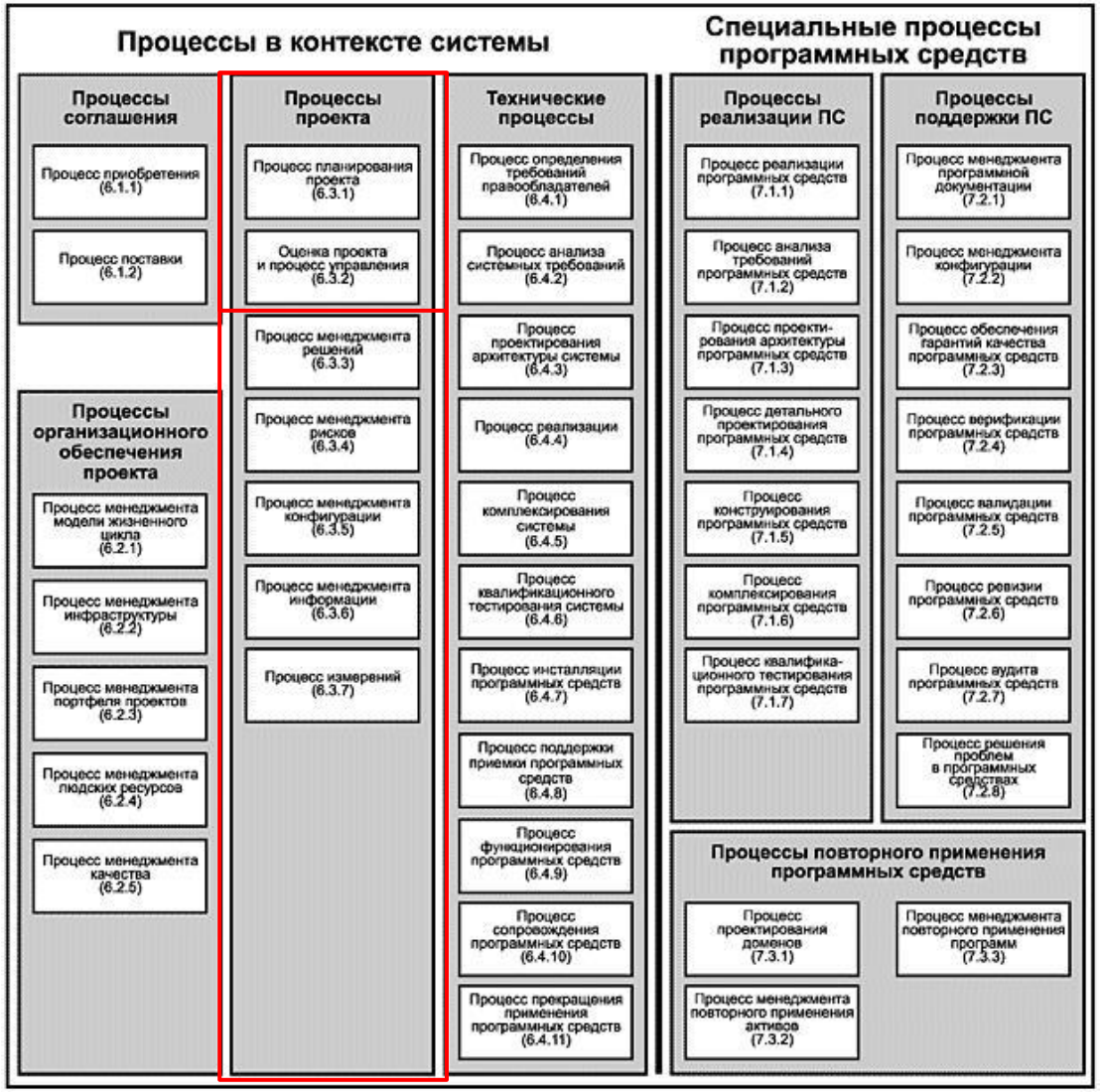


В настоящем стандарте проект выбран как основа для описания процессов, относящихся к планированию, оценке и управлению. Принципы, связанные с этими процессами, могут применяться в любой области менеджмента организаций.

Существуют две категории процессов проекта:

- Процессы менеджмента проекта используются для планирования, выполнения, оценки и управления продвижением проекта.
- Процессы поддержки проекта обеспечивают выполнение специализированных целей менеджмента.

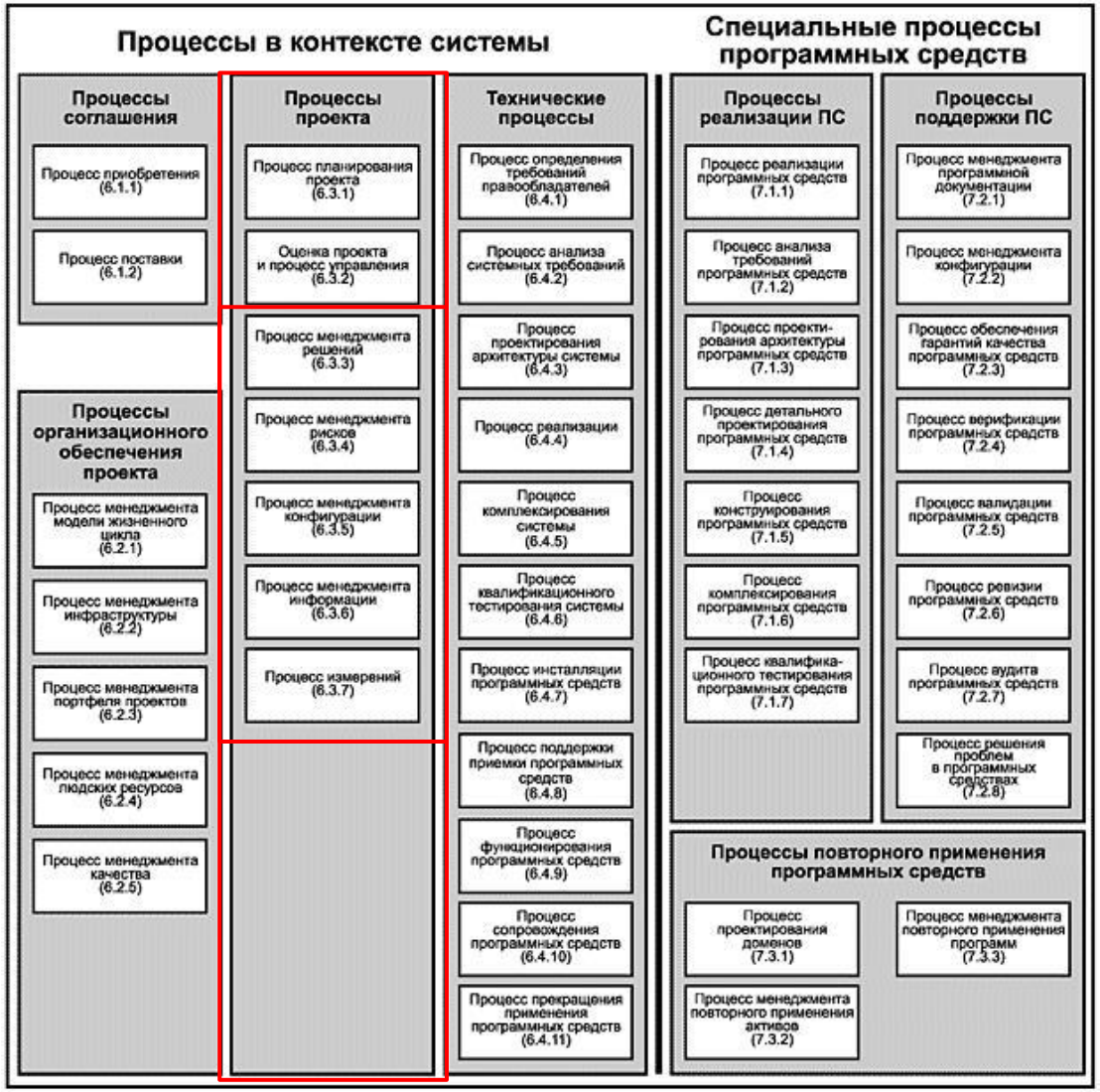
Классификация процессов программной инженерии



Процессы менеджмента проекта применяются для создания и развития планов проекта, оценки фактического выполнения и продвижения относительно плановых заданий и управления выполнением проекта вплоть до полного его завершения. Отдельные процессы менеджмента проекта могут привлекаться в любое время жизненного цикла и на любом уровне иерархии проекта в соответствии с планами проекта или возникновением непредвиденных событий. Процессы менеджмента проекта применяются на уровне строгости и формализации, зависящих от риска и сложности проекта:

- процесс планирования проекта;
- процесс управления и оценки проекта.

Классификация процессов программной инженерии



Процессы поддержки проекта формируют специфическую совокупность задач, ориентированных на выполнение специальных целей менеджмента. Все эти процессы очевидны при осуществлении менеджмента любой инициируемой деятельности, располагаясь по нисходящей от организации в целом вплоть до отдельного процесса жизненного цикла и его задач:

a) процесс менеджмента решений;

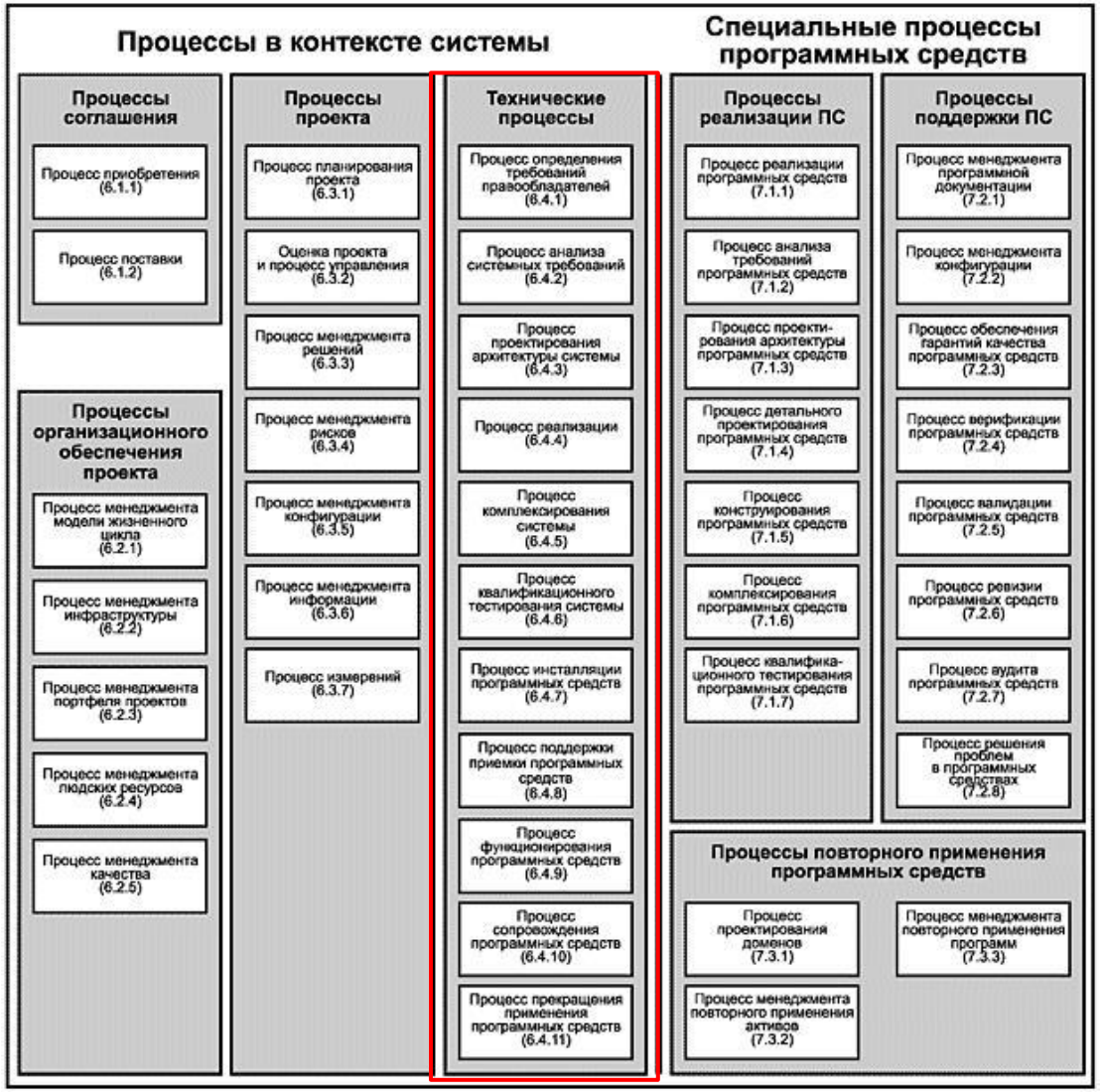
b) процесс менеджмента рисков;

c) процесс менеджмента конфигурации;

d) процесс менеджмента информации;

e)процесс измерений.

Классификация процессов программной инженерии

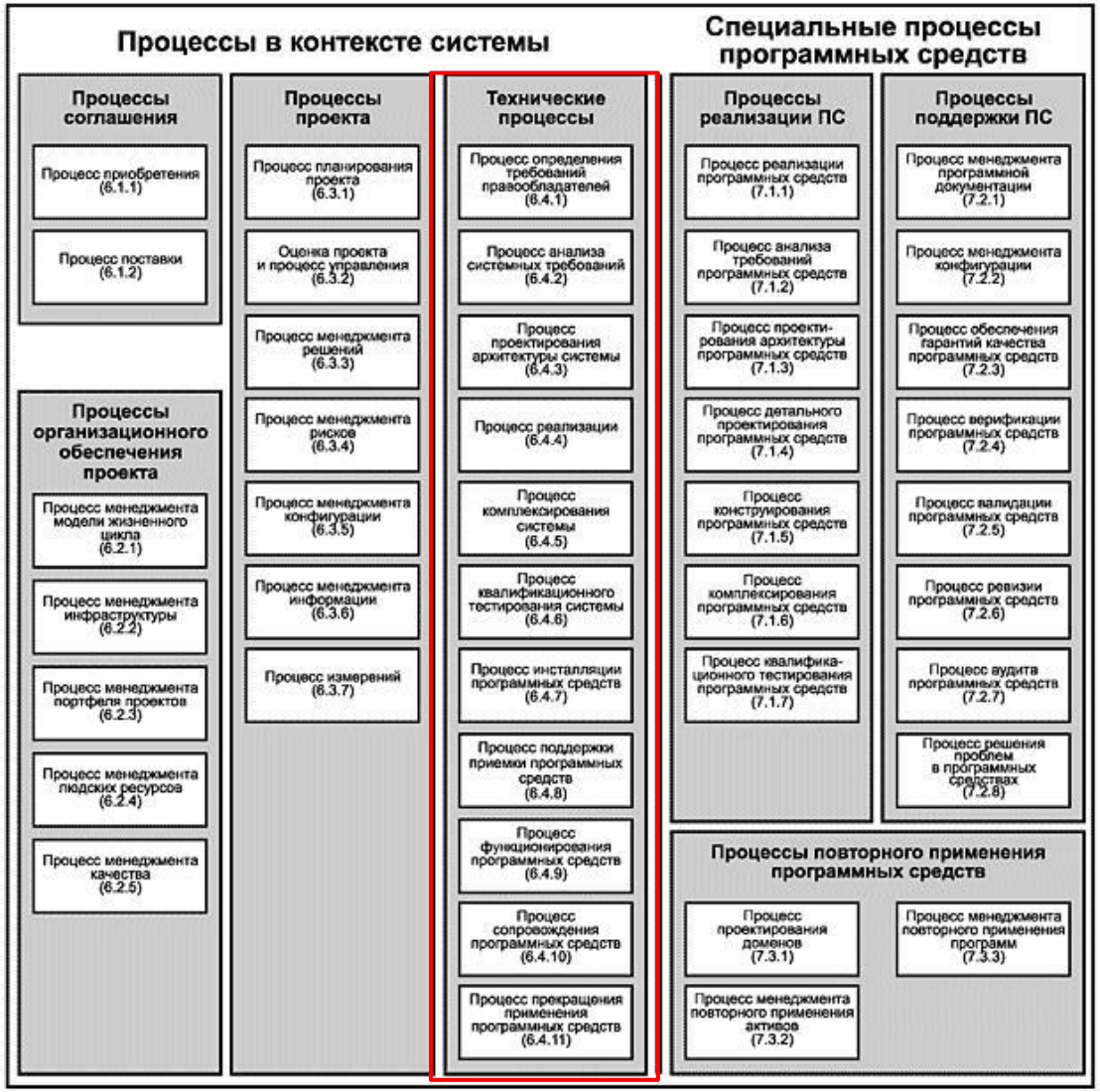


Технические процессы

Технические процессы используются для определения требований к системе, преобразования требований в полезный продукт, для разрешения постоянного копирования продукта (где это необходимо), применения продукта, обеспечения требуемых услуг, поддержания обеспечения этих услуг и изъятия продукта из обращения, если он не используется при оказании услуги.

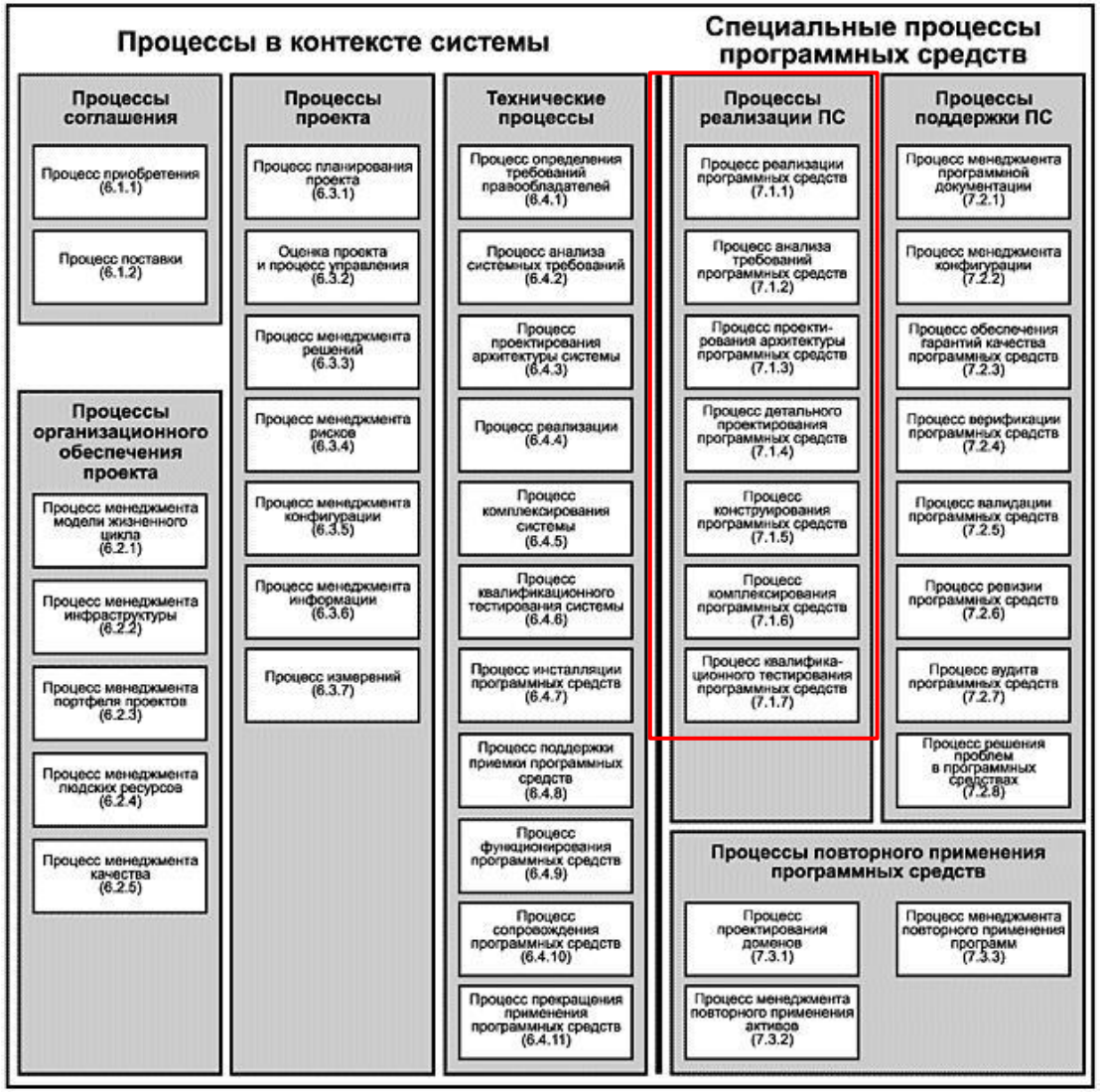
Определяют деятельность, которая дает возможность реализовывать организационные и проектные функции для оптимизации пользы и снижения рисков, являющихся следствием технических решений и действий. Эта деятельность обеспечивает возможность продуктам и услугам обладать такими свойствами, как своевременность и доступность, результативность затрат, а также функциональность, безотказность, ремонтопригодность, продуктивность, приспособленность к применению, и другими качественными характеристиками, требуемыми приобретающими и поддерживающими организациями. Она также предоставляет возможность продуктам и услугам соответствовать ожиданиям или требованиям гражданского законодательства, включая факторы здоровья, безопасности, защищенности и факторы, относящиеся к окружающей среде.

Классификация процессов программной инженерии



Технические процессы состоят из следующих процессов:
a) определение требований правообладателей;
b) анализ системных;
c) проектирование архитектуры;
d) процесс реализации;
e) процесс комплексирования системы;
f) процесс квалификационного тестирования;
g) процесс инсталляции программных средств;
h) процесс поддержки приемки программных средств;
i) процесс функционирования программных средств;
j) процесс сопровождения программных средств;
k) процесс изъятия из обращения программных средств.

Классификация процессов программной инженерии

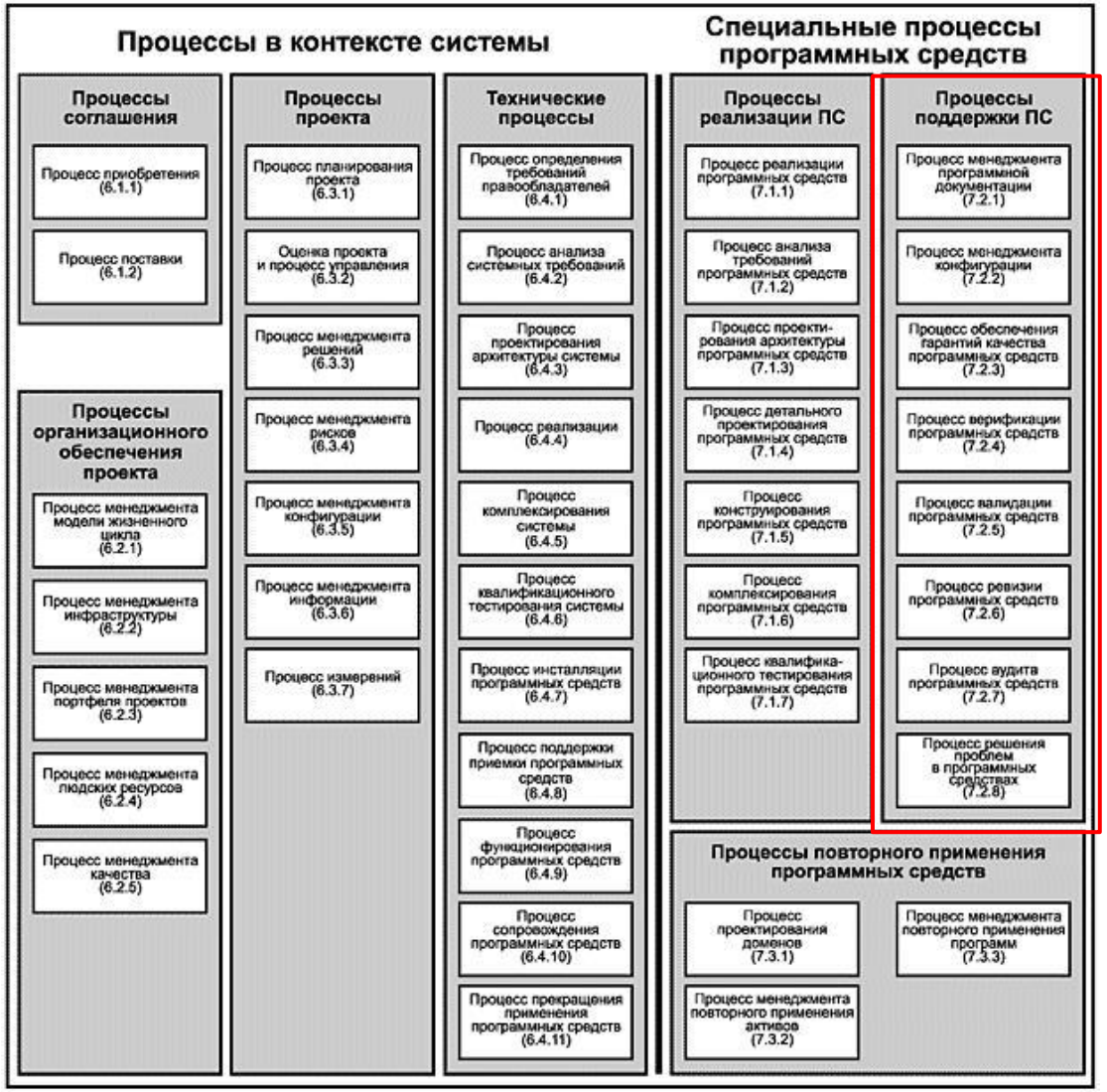


Процессы реализации программных средств используются для создания конкретного элемента системы (составной части), выполненного в виде программного средства. Эти процессы преобразуют заданные характеристики поведения, интерфейсы и ограничения на реализацию в действия, результатом которых становится системный элемент, удовлетворяющий требованиям, вытекающим из системных требований.

Процесс реализации программных средств включает в себя несколько специальных процессов более низкого уровня:

- процесс реализации программных средств;
- процесс анализа требований к программным средствам;
- процесс проектирования архитектуры программных средств;
- процесс детального проектирования программных средств;
- процесс конструирования программных средств;
- процесс комплексирования программных средств;
- процесс квалификационного тестирования программных средств.

Классификация процессов программной инженерии



Процессы поддержки программных средств предусматривают специально сфокусированную совокупность действий, направленных на выполнение специализированного программного процесса. Любой поддерживающий процесс помогает процессу реализации программных средств как единое целое с обособленной целью, внося вклад в успех и качество программного проекта. Существует восемь таких процессов:

a) процесс менеджмента документации программных средств;

b) процесс менеджмента конфигурации программных средств;

c) процесс обеспечения гарантии качества программных средств;

d) процесс верификации программных средств;

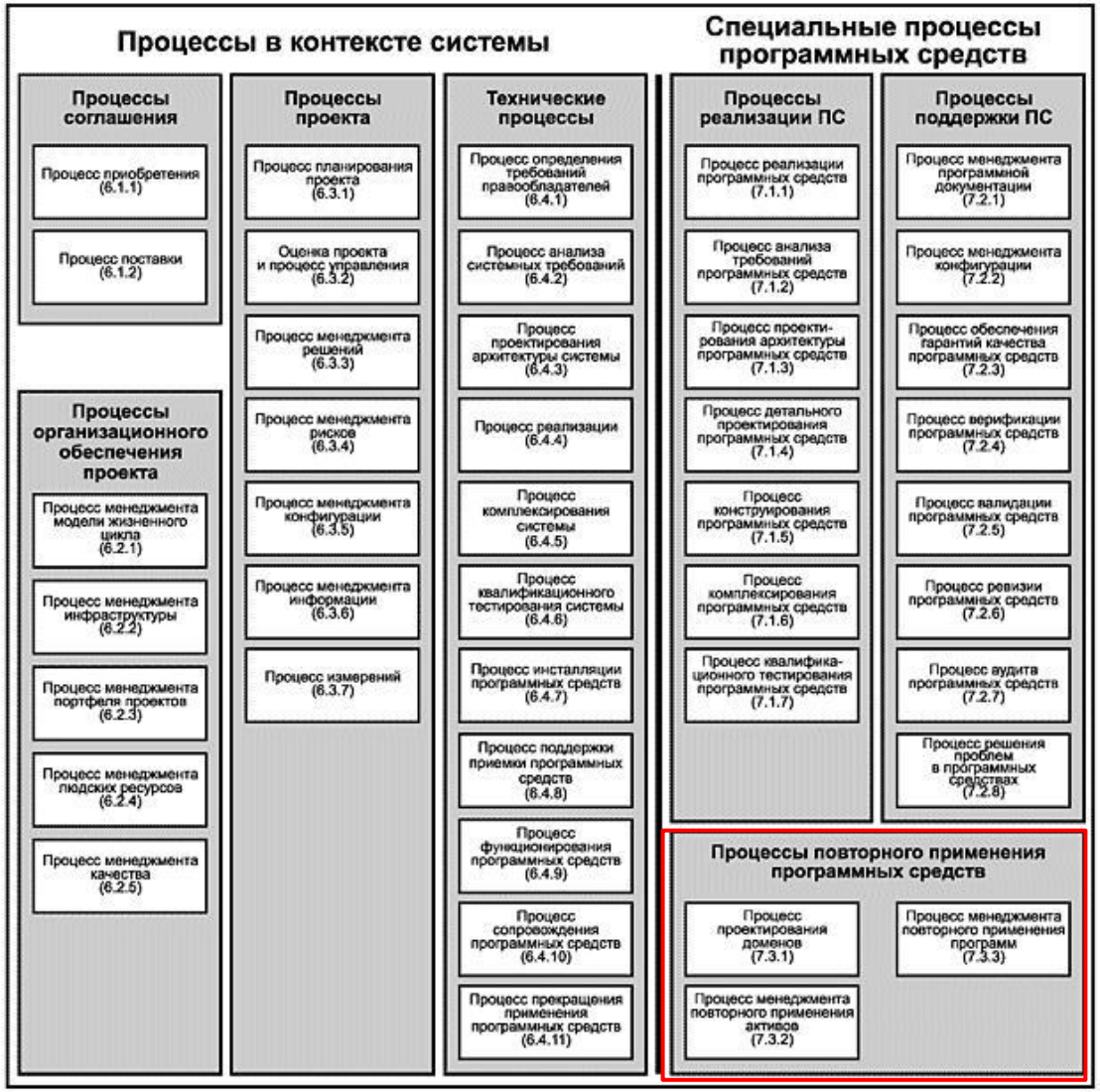
e) процесс валидации программных средств;

f) процесс ревизии программных средств;

g) процесс аудита программных средств;

h) процесс решения проблем в программных средствах.

Классификация процессов программной инженерии



Группа процессов повторного применения программных средств состоит из трех процессов, которые поддерживают возможности организации использовать повторно составные части программных средств за границами проекта. Эти процессы уникальны, поскольку, в соответствии с их природой, они используются вне границ какого-либо конкретного проекта.

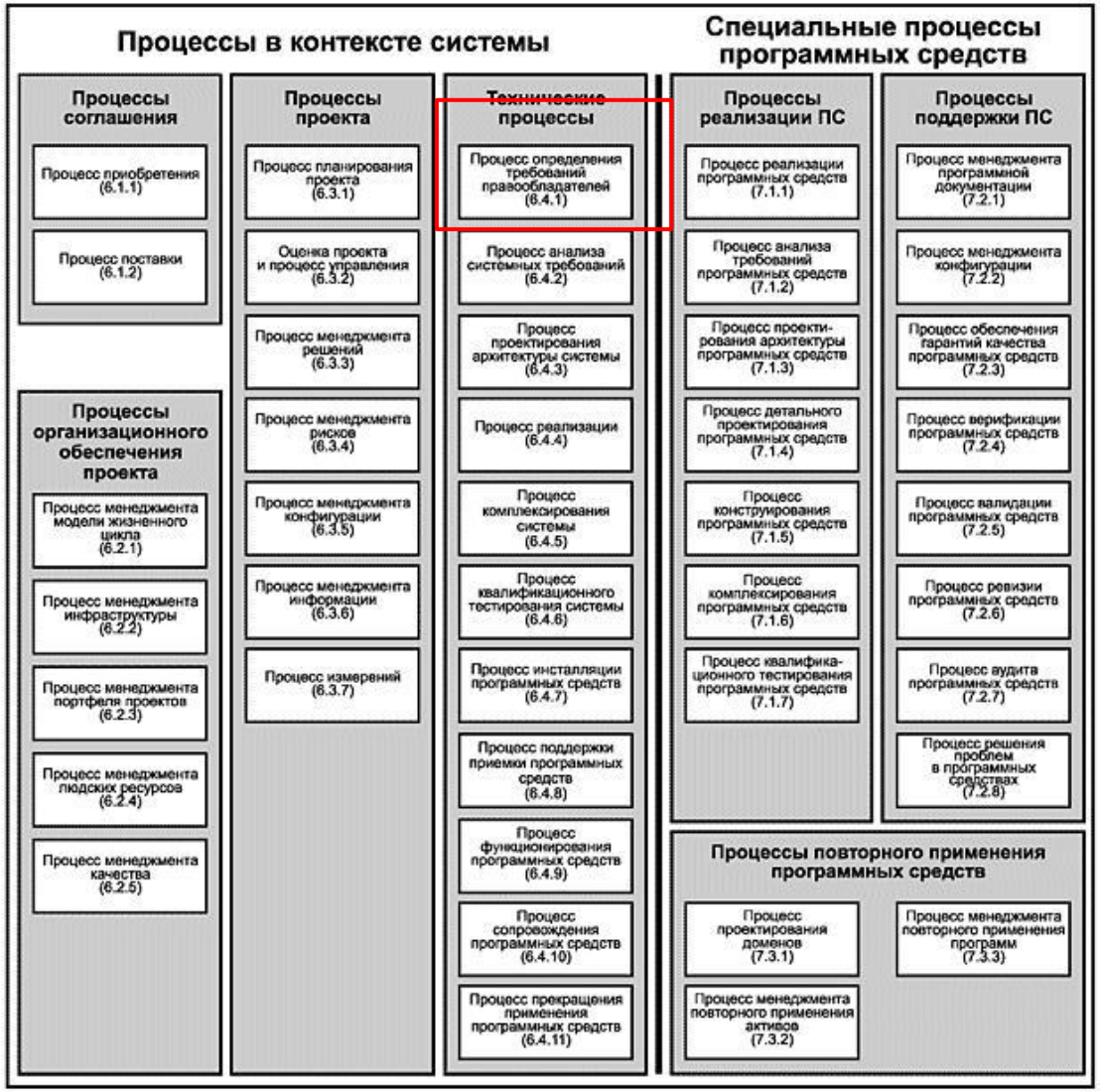
Процессами повторного применения программных средств являются:

a) процесс проектирования доменов;

b) процесс менеджмента повторного применения активов;

c) процесс менеджмента повторного применения программ.

Пример. Процесс определения требований (1)



Цель процесса определения требований правообладателей состоит в выявлении требований к системе, выполнение которых может обеспечивать предоставление услуг, необходимых пользователям и другим правообладателям в заданной среде применения.

Этот процесс позволяет определять правообладателей или классы правообладателей, которые связаны с системой на протяжении всего ее жизненного цикла, а также их потребности и пожелания. В рамках процесса они анализируются и преобразуются в общую совокупность требований правообладателей, которые описывают желаемое поведение системы в процессе взаимодействия со средой применения. Она служит в качестве ссылки, по отношению к которой каждая предоставляемая услуга подвергается валидации для подтверждения того, что система полностью удовлетворяет заявленным требованиям.

Выходы

В результате успешного осуществления процесса определения требований правообладателей:

a) задаются требуемые характеристики и условия использования услуг;

b) определяются ограничения для системных решений;

c) достигается возможность прослеживания от требований правообладателей к правообладателям и их потребностям;

d) описывается основа для определения системных требований;

e) определяется основа для валидации соответствия услуг;

f) формируется основа для ведения переговоров и заключения соглашений о поставке услуги или продукции.

Форма описания процессов

Цель процесса

Выходы (результаты) процесса

Виды деятельности и задачи процесса

Пример. Процесс менеджмента модели жизненного цикла (1)

Цель

Цель процесса менеджмента модели жизненного цикла заключается в определении, сопровождении и обеспечении гарантии наличия политик, процессов жизненного цикла, моделей жизненного цикла и процедур для использования организацией в пределах области применения настоящего стандарта.
Данный процесс предусматривает политики, процессы и процедуры жизненного цикла, согласованные с целями организации, которые определяются, адаптируются, совершенствуются и сопровождаются для поддержки отдельных потребностей проекта в пределах задач и функций организации и которые готовы к применению с использованием эффективных испытанных методов и инструментария.

Выходы

В результате успешного осуществления процесса менеджмента модели жизненного цикла:
a) предоставляются политики и процедуры менеджмента и развертывания моделей и процессов жизненного цикла;
b) определяются обязанности, ответственность и полномочия менеджмента жизненного цикла;
c) определяются, сопровождаются и совершенствуются процессы, модели и процедуры жизненного цикла для применения организацией;
d) осуществляется процесс усовершенствований в порядке установленных приоритетов.

Пример. Процесс менеджмента модели жизненного цикла (2)

Виды деятельности и задачи (Организация должна осуществлять следующие виды деятельности в соответствии с принятыми в организации политиками и процедурами в отношении процесса менеджмента модели жизненного цикла)

Учреждение процессов. Данный вид деятельности состоит из решения следующей задачи: Организация должна учредить совокупность организационных процессов для всех процессов и моделей жизненного цикла программных средств, используемых в деловой деятельности. Эти процессы и их применение в конкретных случаях должны документироваться в публикациях организации. При необходимости следует установить механизм управления процессами при разработке, мониторинге, управлении и совершенствовании процессов.

Примечание - Установление механизма управления процессами включает в себя определение обязанностей, ответственности и полномочий для менеджмента жизненного цикла.

Оценка процессов. Данный вид деятельности состоит из решения следующих задач:

Организации следует разработать, документировать и применять процедуру оценки процессов. Записи об оценках необходимо осуществлять и поддерживать.

Организация должна планировать и осуществлять ревизии процессов через определенные интервалы времени для гарантии того, что процессы остаются пригодными и результативными в свете результатов проведения оценок.

Пример. Процесс менеджмента модели жизненного цикла (3)

Совершенствование процессов. Данный вид деятельности состоит из решения следующих задач:
Организация должна проводить такие улучшения своих процессов, какие она посчитает необходимыми по результатам оценки и ревизии процессов. Процесс документирования следует также обновлять для отражения улучшений в организационных процессах.
Исторические, технические данные, а также данные оценивания следует накапливать и анализировать для усиления понимания сильных и слабых сторон применяемых процессов. Этот анализ следует использовать в качестве обратной связи для совершенствования процессов, выработки рекомендаций по изменениям в текущих или последующих проектах и определения потребностей в передовых технологиях.
Данные о затратах на качество следует накапливать, сопровождать и использовать для совершенствования процессов организации, обусловленных действиями ее руководства. Эти данные должны служить цели установления затрат как на предупреждение, так и на разрешение проблем и несоответствий в программных продуктах и услугах.

Основы процесса моделирования ЖЦ ПП

За десятилетия опыта построения программных систем был наработан ряд типичных схем выполнения работ при проектировании и разработке. Такие схемы получили название моделей ЖЦ.

Модель жизненного цикла — это схема выполнения работ и задач на процессах, обеспечивающих разработку, эксплуатацию и сопровождение программного продукта, отражающая жизнь ПП, начиная от формулировки требований к нему до прекращения его использования. Исторически модель жизненного цикла включает в себя:

- разработку требований или технического задания;
- разработку системы или технического проекта;
- программирование или рабочее проектирование;
- пробную эксплуатацию;
- сопровождение и улучшение;
- снятие с эксплуатации.

Выбор и построение модели ЖЦ ПП базируется на концептуальной идее проектируемой системы, с учетом ее сложности й в соответствии со стандартами, позволяющих формировать схему выполнения работ по усмотрению разработчика и заказчика.

Модель ЖЦ разбивается на процессы реализации, которые должны включать отдельные работы и задачи, реализуемые в данном процессе, и при их завершении осуществлять переход к следующему процессу.

Основы процесса моделирования ЖЦ ПП

Модель ПО:

- Полное описание системы ПО с определенной точки зрения.
- Представляет средства для визуализации, описания, проектирования и документирования системы.

«Моделирование является **центральным** звеном всей деятельности по созданию качественного ПО» Гради Буч

Моделью называется некоторый объект-заместитель (реальный или абстрактный), который в определенных условиях может заменять объект-оригинал, воспроизводя интересующие субъекта свойства и характеристики оригинала при этом имеет определенные преимущества в отношении простоты и удобства взаимодействия субъекта с моделью

Моделирование - это

- (*в узком смысле*): выяснение или воспроизведение свойств какого-либо существующего или создаваемого объекта (процесса, явления) с помощью другого объекта (процесса, явления)
- (*в широком смысле*): научное направление, связанное с построением, совершенствованием, изучением и применением моделей реально существующих или проектируемых объектов (процессов, явлений)

Основы процесса моделирования ЖЦ ПП

Причины использования моделей

Сложность реальных объектов

В данном контексте модель выступает как средство упрощения объекта, в результате которого объем и разнообразие характеризующих его факторов уменьшается до уровня восприимчивости человека

Необходимость проведения экспериментов

Во многих практических ситуациях экспериментальное исследование реальных объектов ограничено высокой стоимостью либо вообще невозможно

Процессы в объектах могут протекать очень быстро (тепловые двигатели, электронные приборы) или очень медленно (геологические, социально-экономические и др. системы)

Необходимость прогнозирования

Необходимость предсказания развития ситуации, оценки последствий принимаемых решений и т.п.

Основы процесса моделирования ЖЦ ПП

При выборе общей схемы модели ЖЦ для конкретной предметной области решаются вопросы включения или невключения отдельных работ, очень важных для создаваемого вида продукта. В настоящее время основой формирования новой модели ЖЦ для конкретной прикладной системы является стандарт ISO/IEC12207, который описывает полный набор процессов, охватывающий все возможные виды работ и задач, связанных с построением ПС.

Из этого стандарта нужно выбрать только те процессы, которые более всего подходят для реализации данного ПС.

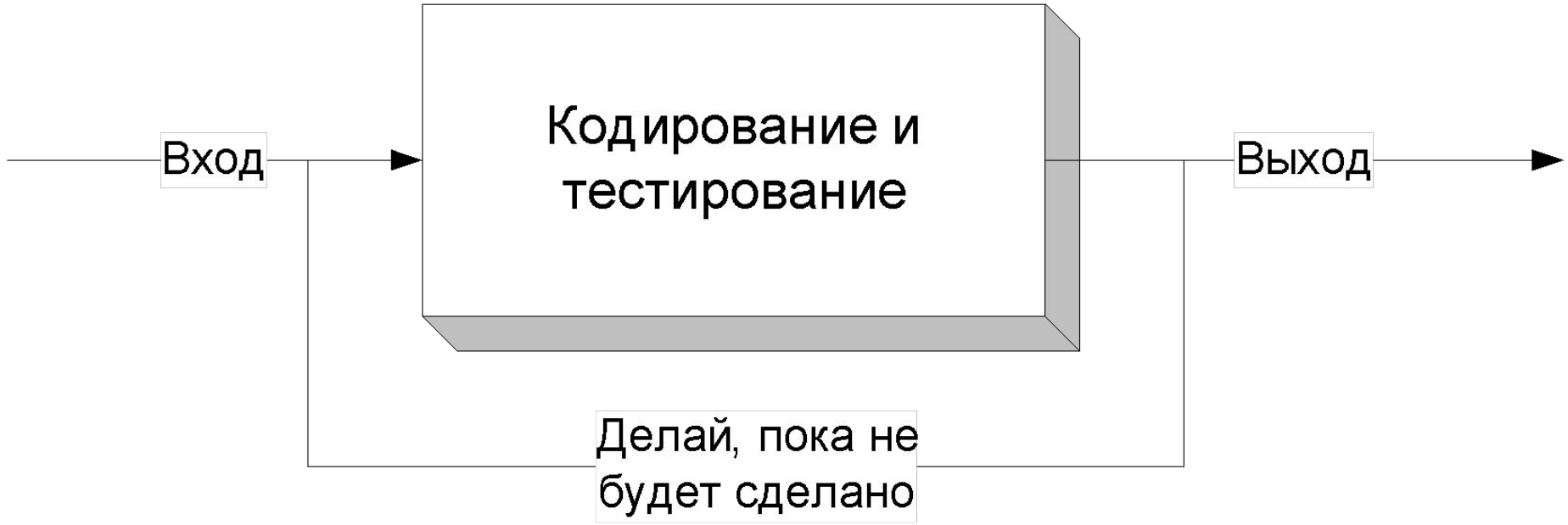
Основы процесса моделирования ЖЦ ПП

Внедрение модели ЖЦ в практическую деятельность по созданию программного продукта позволяет упорядочить взаимоотношения между субъектами процесса и максимально учитывать динамику модификации требований к проекту и системе.

Эти и другие не менее важные вопросы послужили источником формирования различных *видов моделей ЖЦ*, основанных на процессном подходе к разработке программных проектов.

Простейшая модель разработки

- была широко распространена до 1970 гг
- была заменена на каскадную модель
- Обычно используют студенты



Классические виды работ (деятельности) в ЖЦ

Подразумевается, что разработка начинается с заключения контракта с заказчиком (подготовка) и проходит через планирование, моделирование (анализ, проектирование), программирование (кодирование, тестирование) и развертывание (поставка заказчику, сопровождение).

- *Подготовка* обеспечивает активное взаимодействие с потенциальным заказчиком. Здесь собираются и формируются требования, определяющие характеристики и функции будущей программной системы. Фактически эти требования определяют полное задание на разработку.
- *Планирование* На этом этапе решаются задачи планирования программного проекта, определяются объем будущих работ и их риск, необходимые трудозатраты, формируются рабочие задачи и расписание (план-трафик работ).
- *Моделирование* посвящено выполнению двух действий — *анализу требований* и *проектированию*. *Результаты этих действий — модели* - обычно записываются на графическом языке моделирования. языке картинок.

Классические виды работ (деятельности) в ЖЦ

Анализ требований обрабатывает набор требований к ПО, сформированный на этапе подготовки. Уточняются и детализируются функции, характеристики и интерфейс ПО. Все текстовые определения и модели документируются в ***спецификации*** *анализа.*

Проектирование состоит в создании представлений:

- архитектуры ПО;
- структурной и поведенческой организации;
- входных и выходных форм данных.

Классические виды работ (деятельности) в ЖЦ

Архитектура ПО определяет организационную структуру программной системы, разделяет ее на части, связи между этими частями и механизмы взаимодействия, основные руководящие принципы для детализации дальнейшего проектирования

Архитектура это множество значимых решений относительно принципов построения программной системы.

Архитектура фиксирует:

- крупномасштабную организациюструктурных элементов и топологию их связей;
- поведение, описываемое кооперацией этих элементов;
- важные механизмы, доступные во всей системе;
- архитектурный стиль, который управляет организацией элементов системы.

Классические виды работ (деятельности) в ЖЦ

Исходные данные для проектирования содержатся в ***спецификации анализа.*** По сути, в ходе проектирования выполняется трансляция требований к ПО во множество проектных представлений. При решении задач проектирования основное внимание уделяется качеству будущего программного продукта.

Конструирование — этот этап включает в себя действия кодирования и тестирования.
Кодирование, иначе называемое программированием или реализацией, — состоит в переводе результатов проектирования в текст на языке программирования.
Тестирование — выполнение программы для выявления ошибок в функциях, логике и форме реализации программного продукта. Если ошибки выявлены, запускается отладка, цель которой — устранить ошибки.
Развертывание — последний этап классического жизненного цикла нацелен на два действия:

- ***Поставку*** разработанного продукта заказчику и сопровождение процесса эксплуатации этого продукта.
- ***Сопровождение —*** это внесение изменений в эксплуатируемое ПО. Цели изменений:
  - исправление ошибок;
  - адаптация к изменениям внешней для ПО среды:
  - усовершенствование ПО по требованиям заказчика.

Классические виды работ (деятельности) в ЖЦ

Адаптация обычно требуется, если заказчик хочет использовать продукт с другой операционной системой, а усовершенствование состоит или в расширении функ циональности полюбившегося продукта, или в изменении каких-то характеристик (например, ускорения реакции на запросы пользователя).

Согласно статистическим данным, 65% сопровождения связано с усовершенствованием ПО, 18% отводится на адаптацию и 17% связано с исправлением ошибок. Из этого можно заключить, что исправление ошибок не является самой распространенной работой сопровождения. Львиная толика сопровождения ориентирована на модернизацию ПО. Поэтому сопровождение само но себе является естественным продолжением разработки системы со своими этапами подготовки, планирования, моделирования и конструирования.

Сопровождение ПО заключается в повторном применении одного (или нескольких) из предшествующих шагов (этапов) жизненного цикла к существующему ПО, но не в разработке нового ПО. Такая возможность показана на рис. 1.1 стрелками обратных связей.

Стратегии разработки

Существуют три стратегии разработки ПО (были определены в международном стандарте ISO15271) :

- ***однократный проход*** (водопадная стратегия) — линейная последовательность этапов разработки; КЛАССИЧЕСКАЯ МОДЕЛЬ ЖЦ
- ***инкрементная стратегия.*** В начале процесса определяются все пользовательские и системные требования, оставшаяся часть разработки выполняется в виде последовательности версий. Первая версия реализует часть запланированных возможностей, следующая версия реализует дополнительные возможности и т. д., пока не будет получена полная система; ИНКРЕМЕНТНАЯ МОДЕЛЬ ЖЦ
- ***эволюционная стратегия.*** Система также строится в виде последовательности версий, но в начале процесса определены не все требования. Требования уточняются в результате разработки версий. ЭВОЛЮЦИОННАЯ (СПИРАЛЬНАЯ) МОДЕЛЬ ЖЦ

Стратегия разработки	В начале процесса определены все требования?	Множество циклов разработки?	Промежуточное ПО распространяется?
Однократный проход	Да	Нет	Нет
Инкрементная (запланированное улучшение продукта)	Да	Да	Может быть
Эволюционная	Нет	Да	Да

Каскадная модель

ISO 15271

Каскадная модель жизненного цикла по существу реализует принцип однократного выполнения каждого из следующих видов деятельности в их естественных границах:

- установление потребностей пользователя;
- определение требований;
- проектирование системы;
- изготовление системы;
- испытание;
- корректировка;
- поставка или использование.

При применении такого принципа разработки каждого программного объекта соответствующие работы и задачи процесса разработки обычно выполняют последовательно .

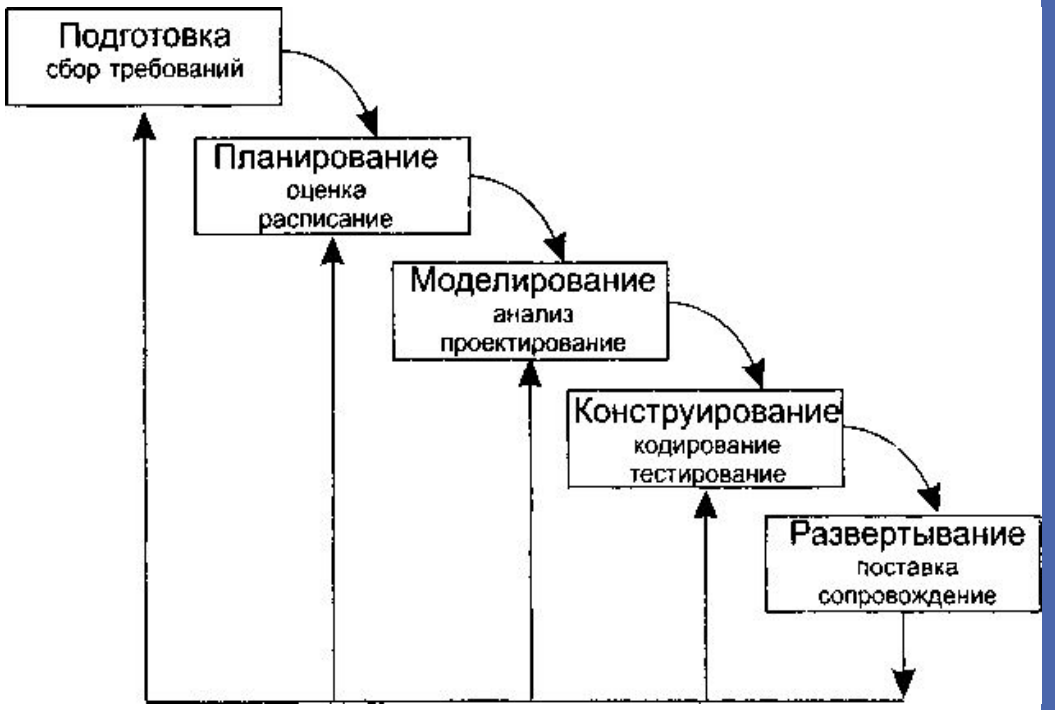
Однако они могут быть частично выполнены параллельно в случаях перекрытия последовательных работ.

Каскадная модель

Старейшей моделью процесса разработки ПО является классический жизненный цикл (автор Уинстон Ройс, 1970) . В 70—80-х гг. XX в. модель была принята как стандарт министерства обороны США.

Очень часто классический жизненный цикл называют каскадной или водопадной моделью, подчеркивая, что разработка рассматривается как последовательность этапов, причем переход на следующий, иерархически нижний этап происходит только после полного завершения работ на текущем этапе.

В разное время в описании каскадной модели включали разное количество фаз (этапов) в зависимости от типа проекта



Каскадная модель

1. исследование концепции: происходит исследование требований, разрабатывается видение продукта и оценивается возможность его реализации;
2. выработка требований: определяются программные требования для информационной предметной области системы, а также предназначение, линия поведения, производительность и интерфейсы;
3. проектирование: разрабатывается и формулируется логически последовательная техническая характеристика программной системы, включая структуру данных, архитектуру ПО, интерфейсные представления и процессуальную (алгоритмическую) детализацию;

4. реализация: эскизное описание ПС превращается в полноценный программный продукт, результатом является исходный код, база данных и документация; в реализации обычно выделяют два этапа: реализацию компонентов ПО и интеграцию компонент в готовый продукт; на обоих этапах выполняется кодирование и тестирование, которые тоже иногда рассматривают как два подэтапа;
5. эксплуатация и поддержка: подразумевает запуск и текущее обеспечение, включая предоставление технической помощи, обсуждение возникших вопросов с пользователем, регистрацию запросов пользователя на модернизацию и внесение изменений, а также корректирование и/или устранение ошибок;
6. сопровождение: устранение программных ошибок, неисправностей, сбоев, модернизация и внесение изменений, что обычно приводит к повторению или итерации отдельных этапов разработки.

Каскадная модель

Основной принцип построения каскадной модели заключается в строго последовательном выполнении фаз, т.е. каждая последующая фаза начинается лишь тогда, когда полностью завершено выполнение предыдущей фазы.

Каждая фаза имеет входные и выходные данные, которые соответствуют определенным критериям входа и выхода. Каждая фаза полностью документируется, переход от одной фазы к другой осуществляется посредством формального обзора с участием заказчика.

Основой модели служат сформулированные требования, которые меняться не должны. Критерием качества результата является соответствие продукта установленным требованиям.

Каскадная модель

Преимущества каскадной модели состоят в следующем. Модель проста, удобна в применении и понятна заказчикам, так как часто используется другими организациями для отслеживания проектов, не связанных с разработкой ПО. Процесс разработки выполняется поэтапно, и ее структурой может руководствоваться даже слабо подготовленный в техническом плане или неопытный персонал. Она способствует осуществлению строгого контроля менеджмента проекта, каждую стадию могут выполнять независимые команды, все документировано, что позволяет достаточно точно планировать сроки и затраты.

Преимущества iso 15271

- однократное представление всех возможностей (характеристик) системы;
- необходимость только единственной фазы перехода от старой системы к новой.

Каскадная модель

При использовании каскадной модели для «неподходящего» проекта могут проявляться следующие ее *недостатки*:

- попытка вернуться на одну или две фазы назад, чтобы исправить какую-либо проблему или недостаток, приведет к значительному увеличению затрат и сбою в графике;
- интеграция компонентов, на которой обычно выявляется большая часть ошибок, выполняется в конце разработки, что сильно увеличивает стоимость устранения ошибок;
- запаздывание с получением результатов (если в процессе выполнения проекта требования изменились, то получится устаревший результат).

Недостатки каскадной модели особо остро проявляются в случае, когда трудно (или невозможно) сформулировать требования или требования могут меняться в процессе разработки.

Недостатки iso 15271

- требования к объектам определены недостаточно четко;
- система обычно слишком велика, чтобы все работы по ее созданию выполнять однократно;
- предполагаемые скорые изменения в технологиях работ;
- возможные текущие изменения требований к системе;
- ограниченность ресурсов, например средств или персонала;
- промежуточный продукт может быть непригоден для использования.

Каскадная модель

Каскадная модель не утратила своей актуальности при решении определенного типа задач, когда требования и их реализация максимально четко определены и понятны или используется неизменяемое определение продукта и вполне понятные технические методики, например при решении задач научно-вычислительного характера (разработка пакетов и библиотек научных программ); при разработке операционных систем и компиляторов, систем реального времени управления конкретными объектами; при повторной разработке типового продукта (автоматизированного бухгалтерского учета, начисления зарплаты); при выпуске новой версии уже существующего продукта, если вносимые изменения вполне определены и управляемы (перенос уже существующего продукта на новую платформу); и наконец, принципы каскадной модели находят применение в элементах моделей других типов.

Итерационные модели

Итерационные модели в целом можно разделить на два класса: *модели с приращениями* (Incremental) и *эволюционные* (Evolutionary). По всем этим моделям программный продукт разрабатывается *итерациями*, и каждая итерация заканчивается выпуском работоспособной версии программного продукта. Основное отличие между ними - в подходах к определению требований. Следует отметить, что названия этих моделей в литературе иногда путают.

Инкрементная модель

ISO 15271

Инкрементная модель жизненного цикла, называемая также запланированным усовершенствованием продукта, начинается с выдачи набора требований и реализует разработку последовательности конструкций. Первая конструкция содержит часть требований, в последующую конструкцию добавляют дополнительные требования и так далее до тех пор, пока не будет закончено создание системы. Для каждой конструкции выполняют необходимые процессы, работы и задачи, например анализ требований и создание архитектуры могут быть выполнены сразу, в то время как разработку технического проекта программного средства, его программирование и тестирование, сборку программных средств и их квалификационные испытания выполняют при создании каждой из последующих конструкций.

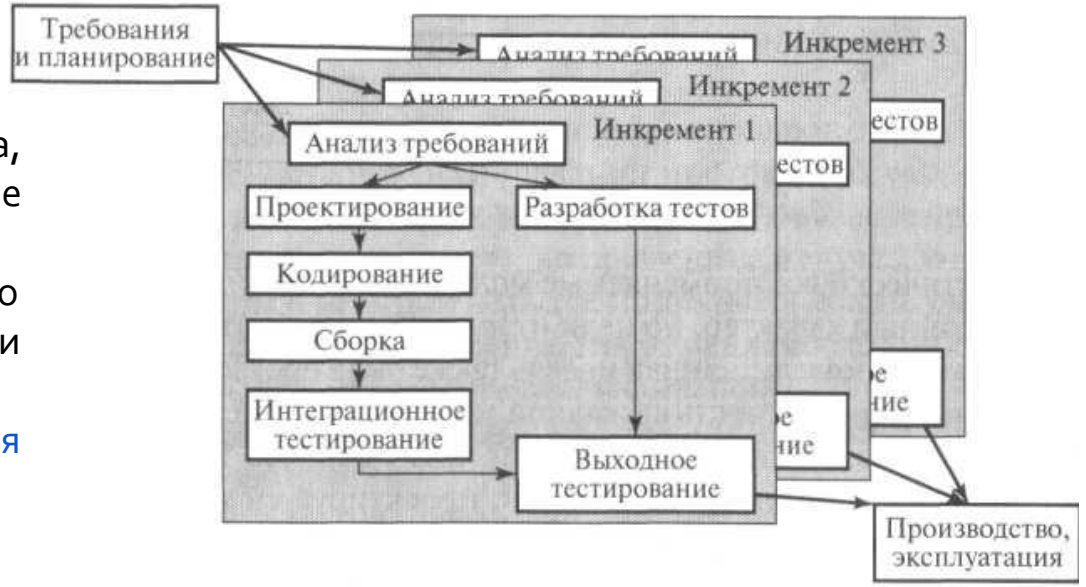
Инкрементная модель

Инкрементная разработка представляет собой процесс пошаговой реализации всей системы и поэтапного наращивания (приращения) функциональных возможностей . На первом шаге (инкремент 1) необходим полный, заранее сформулированный набор требований, которые разделяются по некоторому признаку на группы. Далее выбирается первая группа требований и выполняется полный «проход» по каскадной модели. После того как первый вариант системы, выполняющий первую группу требований, сдан заказчику, разработчики переходят к следующему шагу (инкременту 2) по разработке варианта, выполняющего вторую группу требований. и т.д.

Первый инкремент приводит к получению базового продукта, реализующего базовые требования (правда, многие вспомогательные требования остаются нереализованными).

План следующего инкремента предусматривает модификацию базового продукта, обеспечивающего дополнительные характеристики и функциональность.

По своей природе инкрементный процесс итеративен, инкрементная модель обеспечивает на каждом инкременте работающий продукт.



Инкрементная модель

Особенностью инкрементной модели является разработка приемочных тестов на этапе анализа требований, что упрощает приемку варианта заказчиком и устанавливает четкие пели разработки очередного варианта системы.

Инкрементная модель *особенно эффективна* в случае, когда задача разбивается на несколько относительно независимых подзадач (например, разработка подсистем «Зарплата», «Бухгалтерия», «Склад», «Поставщики»), При этом для внутренней итерации в инкрементной модели можно использовать не только каскадную, но и другие типы моделей.

Современная реализация инкрементного подхода — экстремальное программирование XP (Кент Бек, 1999) . Оно ориентировано на очень малые приращения функциональности.

Инкрементная модель

Данной модели присущи следующие недостатки, которые необходимо учитывать при оценке возможности ее применения:

a) требования к объектам определены недостаточно четко;

b) предусмотрены сразу все возможности системы;

c) предполагаемые скорые изменения в технологиях работ;

d) возможные текущие изменения требований к системе;

e) привлечение ресурсов (средств или персонала) на длительный период ограничено.

Преимущества использования данной модели:

a) необходимость изначального использования характеристик системы;

b) пригодность для использования промежуточного продукта;

c) естественное разделение системы на наращиваемые компоненты (инкременты);

d) возможности наращивания привлекаемого персонала и средств.

Эволюционная (спиральная модель)

На практике при решении достаточно большого количества задач разработка ПО имеет циклический характер, когда после выполнения некоторых стадий приходится возвращаться на предыдущие. Можно указать две основные причины таких возвратов. Во-первых, это ошибки разработчиков, допущенные па ранних стадиях и обнаруженные на более поздних (ошибки анализа, проектирования или кодирования, выявляемые, как правило, на стадии тестирования). Во-вторых, это изменения требований в процессе разработки («ошибки» заказчика). Это или неготовность заказчика сформулировать требования («сказать, что должна делать программа, я смогу только после того, когда увижу, как она работает»), или изменения требований, вызванные изменениями ситуации в процессе разработки (изменения рынка, новые технологии и т.д.).

Циклический характер разработки ПО отражается в спиральной модели ЖЦ, описанной Б. Боэмом в 1988 г. Эта модель, учитывающая повторяющийся характер разработки ПО, была предложена как альтернатива каскадной модели.

Эволюционная (спиральная модель)

ISO15271

В эволюционной модели жизненного цикла систему также разрабатывают в виде отдельных конструкций, но в отличие от инкрементной модели требования изначально не могут быть полностью осознаны и установлены. В данной модели требования устанавливают частично и уточняют в каждой последующей конструкции.

При таком методе для каждой конструкции работы и задачи процесса разработки выполняют последовательно или параллельно с частичным перекрытием.

Работы и задачи процесса разработки обычно выполняют многократно в той же последовательности для всех конструкций. Процессы сопровождения и эксплуатации могут быть реализованы параллельно с процессом разработки. Процессы заказа и поставки, а также вспомогательные и организационные процессы обычно выполняют параллельно с процессом разработки.

Эволюционная (спиральная модель)

Основные принципы спиральной модели можно сформулировать следующим образом.

- Разработка нескольких вариантов продукта, соответствующих различным вариантам требований, с возможностью вернуться к более ранним вариантам.
- Создание прототипов ПО как средства общения с заказчиком для уточнения и выявления требований.
- Планирование следующих вариантов с оценкой альтернатив и анализом рисков, связанных с переходом к следующему варианту.
- Переход к разработке следующего варианта до завершения предыдущего в случае, когда риск завершения очередного варианта/ прототипа становится неоправданно высок.
- Использование каскадной модели как схемы разработки очередного варианта продукта.
- Активное привлечение заказчика к работе над проектом. Заказчик участвует в оценке очередного прототипа, уточнении требований при переходе к следующему, оценке предложенных альтернатив очередного варианта и оценке рисков.

Эволюционная (спиральная модель)

Разработка вариантов продукта в спиральной модели представляется как набор циклов раскручивающейся спирали. Каждому циклу соответствует такое же количество стадий, как и в каскадной модели. При этом начальные стадии, связанные с анализом и планированием, представлены более подробно с добавлением новых элементов.

АР — анализ рисков; П — прототип

В каждом цикле выделяются четыре базовые фазы:

- определение целей, альтернативных вариантов и ограничений;
- оценка альтернативных вариантов, идентификация и разрешение рисков;
- разработка продукта следующего уровня;
- планирование следующей фазы.

Эволюционная (спиральная модель)

«Раскручивание» проекта начинается с анализа общей постановки задачи на разработку ПП. На этой фазе определяются общие цели, устанавливаются предварительные ограничения, определяются возможные альтернативные подходы к решению задачи; на следующей фазе проводится оценка подходов, устанавливаются их риски; и наконец, на фазе разработки создается общая концепция (видение) продукта и путей его создания.

Следующий цикл начинается с планирования требований и деталей ЖЦ продукта для оценки затрат. На фазе определения целей устанавливаются альтернативные варианты требований, связанные с ранжированием требований по важности и стоимости их выполнения. На фазе оценки устанавливаются риски вариантов требований. На фазе разработки — спецификация требований (с указанием рисков и стоимости), готовится демоверсия ПО для анализа требований заказчиком.

Эволюционная (спиральная модель)

Цикл разработки проекта начинается с планирования разработки. На фазе определения целей устанавливаются ограничения проекта (по срокам, объему финансирования, ресурсам), определяются альтернативы проектирования, связанные с альтернативами требований, применяемыми технологиями проектирования, привлечением субподрядчиков. На фазе оценки альтернатив устанавливаются риски вариантов и делается выбор варианта для дальнейшей реализации. На фазе разработки выполняется проектирование и создается демоверсия, отражающая основные проектные решения.

Цикл реализации также начинается с планирования. Альтернативными вариантами реализации могут быть применяемые технологии реализации, привлекаемые ресурсы. Оценка альтернатив и связанных с ними рисков определяется степенью «отработанности» технологий и «качеством» имеющихся ресурсов. Фаза разработки выполняется по каскадной модели с выходом в виде действующего варианта/прототипа продукта.

Следует отметить некоторые *особенности* спиральной модели. До начала разработки ПП есть несколько полных циклов анализа требований и проектирования. Количество циклов (в части анализа, проектирования и реализации) не ограничено и определяется сложностью и объемом задачи. В модели предполагаются возвраты на оставленные варианты при изменении стоимости рисков.

Эволюционная (спиральная модель)

Спиральная модель (по сравнению с каскадной) имеет очевидные *преимущества.* Появляется возможность более тщательного проектирования (несколько начальных итераций) с оценкой результатов проектирования, что позволяет выявить ошибки проектирования на более ранних стадиях. Поэтапно уточняются требования заказчика в процессе выполнения итераций, что позволяет обеспечить более точное их удовлетворение. Заказчик может принимать участие в выполнении проекта с использованием прототипов программы. Заказчик видит, что и как создается, и не выдвигает необоснованных требований, реально оценивает объемы финансирования. Планирование и управление рисками при переходе на следующие итерации позволяют разумно распределять ресурсы и обосновывать финансирование работ. Возможна разработка сложного проекта «по частям» с выделением на первых этапах наиболее значимых требований.

Эволюционная (спиральная модель)

Основные *недостатки* спиральной модели связаны с такими факторами, как:

- сложность анализа и оценки рисков при выборе вариантов;
- сложность поддержания версий продукта (хранение версий, возврат к ранним версиям, комбинация версий);
- сложность оценки точки перехода на следующий цикл;
- «бесконечность» модели (на каждом витке заказчик может выдвигать новые требования, которые приводят к необходимости следующего цикла разработки).

Эволюционная (спиральная модель)

Спиральную модель *целесообразно применять* в следующих случаях:

- когда пользователи не уверены в своих потребностях;
- требования слишком сложны и могут меняться в процессе выполнения проекта, поэтому необходимо прототипирование для анализа и оценки требований;
- достижение успеха не гарантировано и необходима оценка рисков продолжения проекта;
- проект является сложным, дорогостоящим и обоснование его финансирования возможно только в процессе его выполнения;
- когда речь идет о применении новых технологий;
- при выполнении очень больших проектов, которые в силу ограниченности ресурсов можно делать только по частям.

Выводы о классических моделях

Каскадная и спиральная модели устанавливают определенные принципы организации ЖЦ создания программного продукта. Каждая из них имеет преимущества, недостатки и области применимости. Каскадная модель проста, но применима в случае, когда требования известны и меняться не будут. Спиральная модель учитывает такие важные показатели проекта, как изменяемость требований, невозможность оценить заранее объем финансирования, риски выполнения проекта. Но спиральная модель сложна и требует больших затрат на сопровождение.

Существуют и другие модели, которые можно рассматривать как «промежуточные» между каскадной и спиральной. Они используют отдельные преимущества каскадной и спиральной моделей и достигают успеха при решении определенных типов задач.

Разновидности классических моделей

- Каскадная модель с обратной связью
- Каскадная модель с промежуточным контролем
- V-образная модель
- Каскадная модель с прототипированием
- Прототипирование
- Итерационная модель
- Инкрементная модель с перекрытием итераций
- Модель эволюционного прототипирования
- Компонентно-ориентированная модель

Каскадная модель с обратной связью

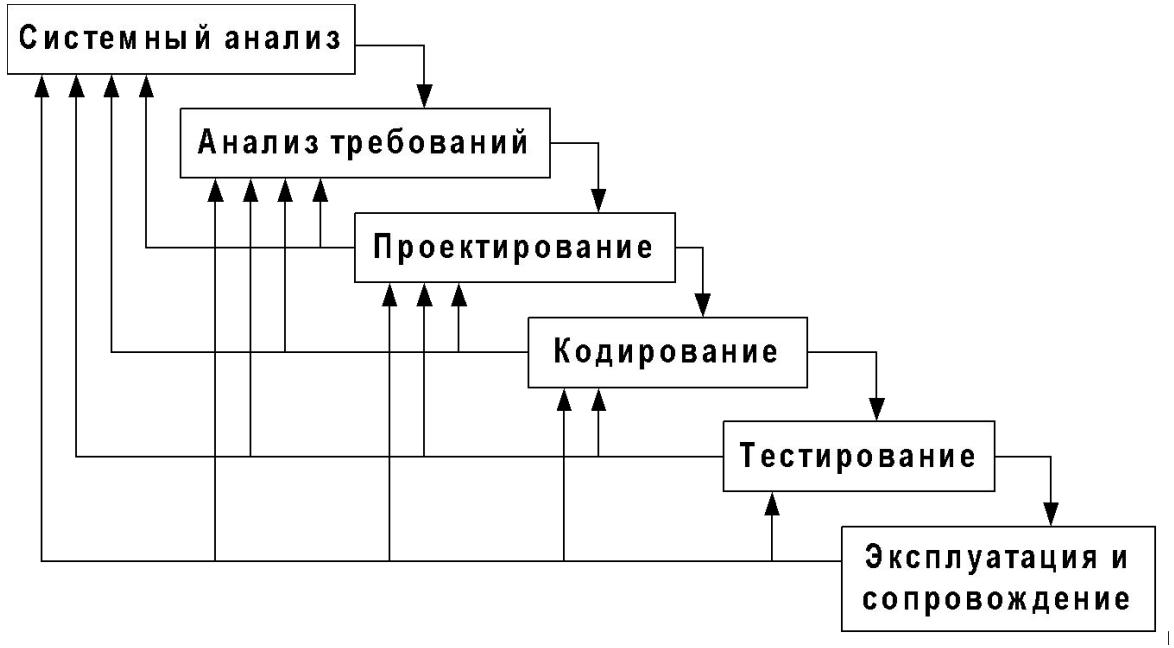
Эта модель расширяет стандартную модель включением в нее циклов обратной связи для возврата на предыдущую стадию при изменении требований, проекта и по результатам инспекций или любых действий по V&V.

Процессы V&V, выполняемые после завершения каждой стадии разработки, играют в этой модели важнейшую роль. В таблице подытожены характеристики и преимущества, обеспечиваемые моделью с обратной связью.

Характеристики	Преимущества
Последовательное упорядочивание стадий	Применение формальных проверок позволяет своевременно выявить дефекты
Формальные проверки по завершении каждой стадии (инспекции, технические обзоры)	Четкие критерии начала и завершения стадий
Наличие документированных требований и проекта	Четкие требования и цели проекта

Каскадная модель с промежуточным контролем

- Требования фиксируются в виде технического задания на все время создания ПО.
- Согласование получаемых результатов с пользователями производится только в точках, планируемых после завершения каждой стадии (возможна корректировка результатов, если они не затрагивают требования, изложенные в техническом задании).
- Пользователи могут внести существенные замечания только после того, как работа над системой будет полностью завершена.
- В случае неточного изложения требований или их изменения в течение длительного периода создания ПО пользователи получают систему, не удовлетворяющую их потребностям.

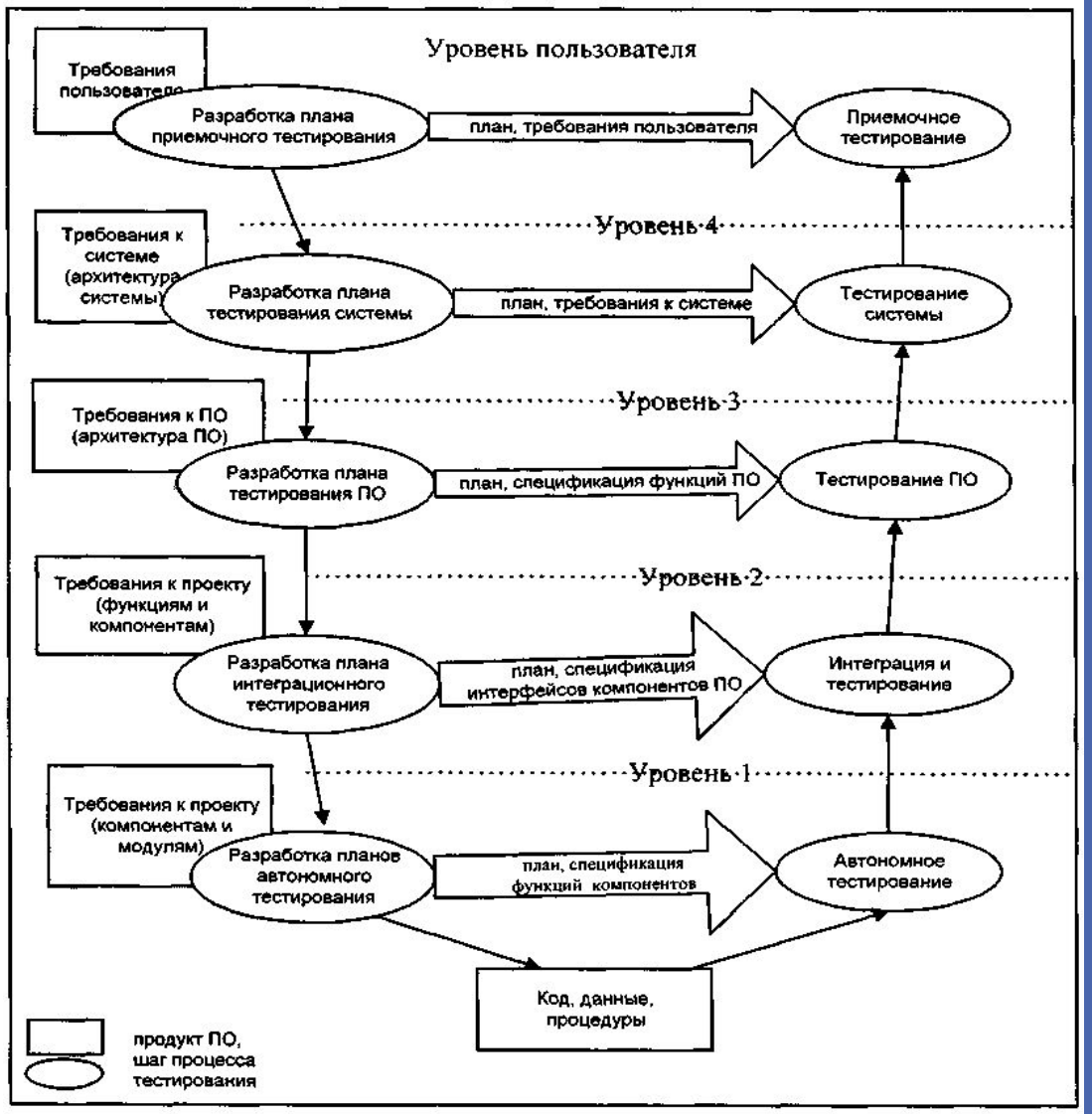


V-образная модель

V-образная (V-shape) модель расширяет каскадную модель включением в нее действий по раннему планированию тестирования.

Взаимосвязь уровней и целей тестирования можно представить в виде модифицированной каскадной модели ЖЦ (так называемой V-образной модели. В этой модели тестирование рассматривается как непрерывный процесс, интегрированный в процесс разработки ПС и включающий два взаимосвязанных подпроцесса - подготовку к тестированию в рамках процессов разработки системы (левая ветвь на рисунке) и проведение тестирования соответствующих объектов (правая ветвь).

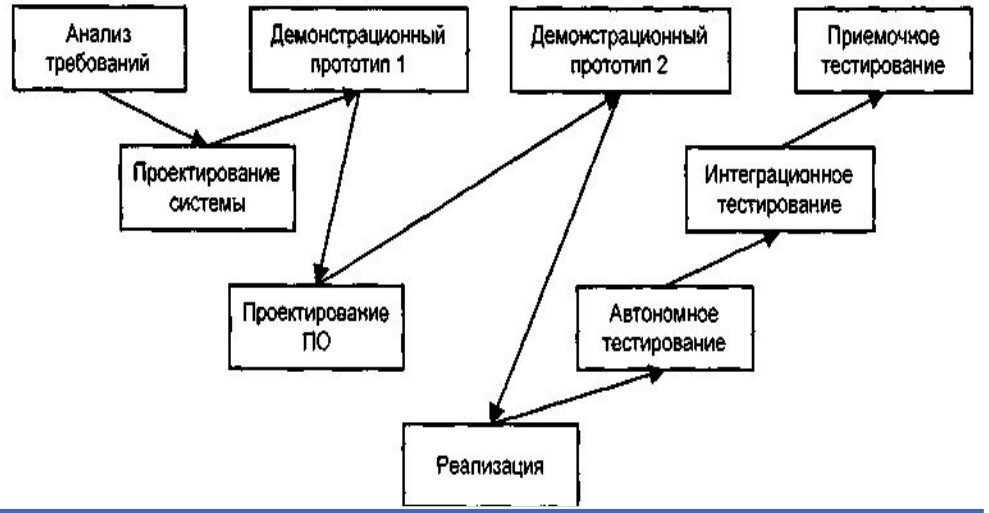
Характеристики	Преимущества
Проверка и оценка тестопригодности требований на ранних стадиях разработки (посредством анализа, выполняемого при планировании тестирования)	Обеспечивает обратную связь с пользователем на ранних стадиях ЖЦ
Наличие документированных тестовых требований	Улучшает планирование и распределение затрат на тестирование
	Четкие документированные цели тестирования



Каскадная модель с прототипированием

Модель является модификацией V-образной модели с включением в нее прототипирования для моделирования требований и проекта

Прототипы имеют демонстрационную цель и после разработки проекта выбрасываются ("прототипирование с выбрасыванием"), а реализация проекта может выполняться в другой среде. В целом разработка выполняется последовательно.



Характеристики	Преимущества
Для анализа и моделирования проектных решений применяется прототипирование с выбрасыванием	Устраняет проблемы связанные с неполнотой и нечеткостью требований

Риски, связанные с выбором модели	Когда лучше применять
Требования не полностью понятны	Требования понятны и не будут существенно изменяться
Система большая чтобы быть реализованным сразу	Разрабатываемая система имеет небольшой размер и сложность
Быстрые изменение в технологии	Все возможности должны быть реализованы сразу
Частой изменение требований	Новая система разрабатывается взамен старой и нужно полностью заменить старую систему
Пользователь не может использовать промежуточные продукты	

Прототипирование = Макетирование

Достаточно часто заказчик не может сформулировать подробные требования по вводу, обработке или выводу данных для будущего программного продукта. В этих случаях целесообразно использовать макетирование.

Основная цель макетирования: снять неопределенности в требованиях заказчика.

Макетирование (прототипирование) — это процесс создания модели требуемого программного продукта.

Модель может принимать одну из трех форм:

- бумажный макет или макет на основе ПК (изображает или рисует человеко- машинный диалог);
- работающий макет (выполняет некоторую часть требуемых функций);
- существующая программа (характеристики которой затем должны быть улучшены).

Макетирование основывается на многократном повторении итераций, в которых участвуют заказчик и разработчик.

Прототипирование = Макетирование

Последовательность действий при макетировании представлена на рис. Макетирование начинается со сбора и уточнения требований к создаваемому ПО. Разработчик и заказчик встречаются и определяют все цели ПО, устанавливают, какие требования известны, а какие предстоит доопределить.

Затем выполняется быстрое проектирование. В нем внимание сосредотачивается на тех характеристиках ПО, которые должны быть видимы пользователю.

Быстрое проектирование приводит к построению макета.

Макет оценивается заказчиком и используется для уточнения требований к ПО.

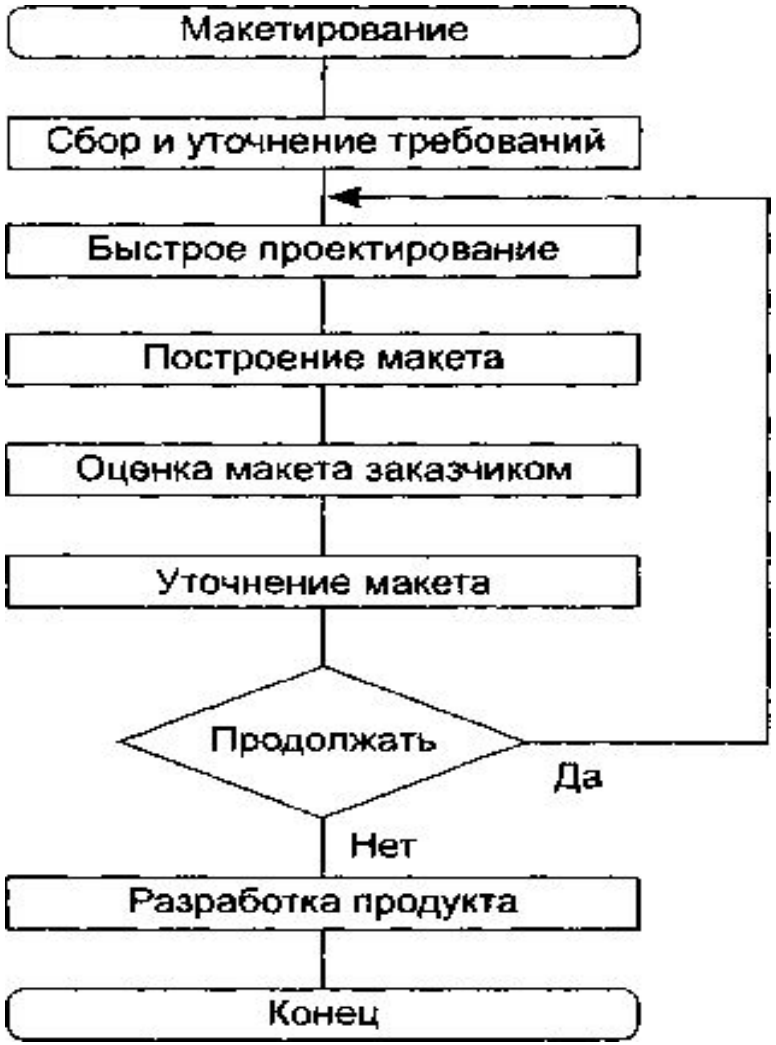
Итерации повторяются до тех пор, пока макет не выявит все требования заказчика и тем самым не даст возможность разработчику понять, что должно быть сделано.

Достоинство макетирования:

- обеспечивает определение полных требований к ПО.

Недостатки макетирования.

- заказчик может принять макет за продукт;
- разработчик может принять макет за продукт.



Прототипирование = Макетирование

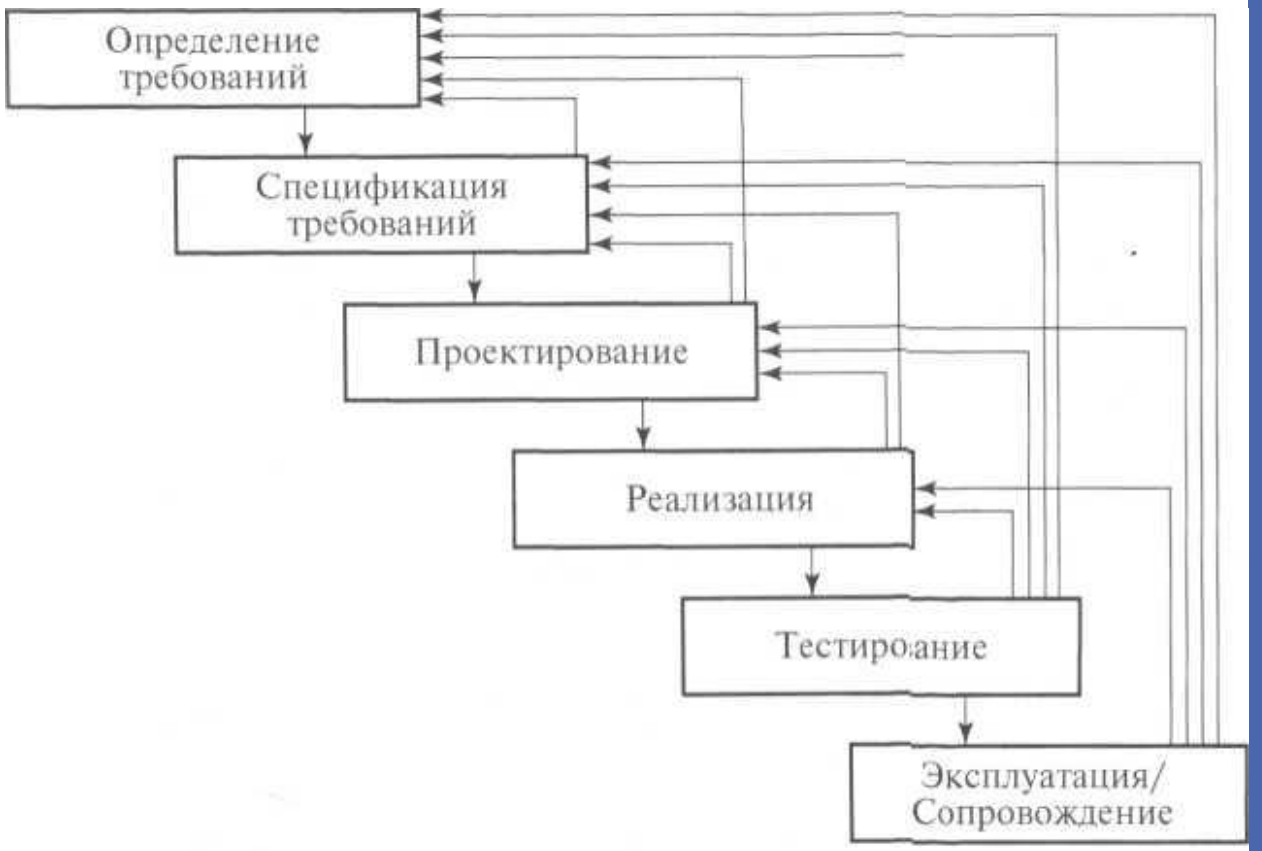
Поясним суть недостатков. Когда заказчик видит работающую версию ПО, он перестает сознавать, что детали макета скреплены «жевательной резинкой и проволокой»; он забывает, что в погоне за работающим вариантом оставлены нерешенными вопросы качества и удобства сопровождения ПО. Когда заказчику говорят, что продукт должен быть перестроен, он начинает возмущаться и требовать, чтобы макет «в три приема» был превращен в рабочий продукт. Очень часто это отрицательно сказывается на управлении разработкой ПО.

С другой стороны, для быстрого получения работающего макета разработчик часто идет на определенные компромиссы. Могут использоваться не самый подходящий язык программирования или операционная система. Для простой демонстрации возможностей может применяться неэффективный алгоритм. Спустя некоторое время разработчик забывает о причинах, по которым эти средства не подходят. В результате далеко не идеальный выбранный вариант интегрируется в систему.

Очевидно, что преодоление этих недостатков требует борьбы с житейским соблазном — принять желаемое за действительное

Итерационная модель

Эта модель жизненного цикла является развитием классической каскадной модели, но предполагает возможность возврата на ранее выполненные этапы. Причинами возврата в классической итерационной модели являются выявленные ошибки, устранение которых и требует возврата на предыдущие этапы в зависимости от типа ошибки (ошибки кодирования, проектирования, спецификации или определения требований). Реально итерационная модель является более жизненной, чем классическая каскадная модель, так как создание ПО всегда связано с устранением ошибок.



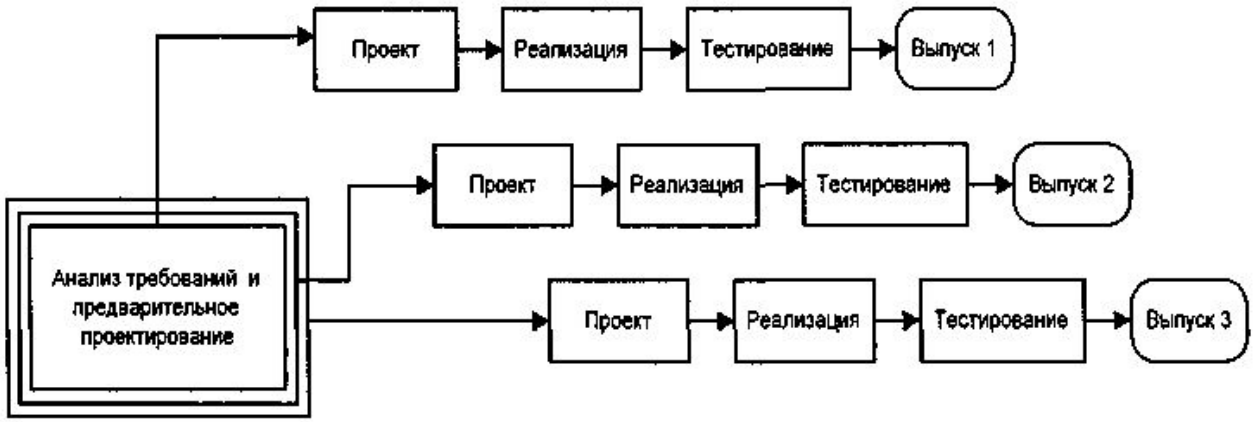
Инкрементная модель с перекрытием итераций

По моделям с приращениями (Incremental) программный продукт разрабатывается итерациями, с добавлением на каждой итерации функциональных возможностей. При этом вначале определяются *все* требования к ПС, а возможно и разрабатывается предварительный проект. Дальнейшая разработка ПС разбивается на итерации. В каждой итерации разработка выполняется последовательно и завершается выпуском работоспособной версии программного продукта (инкремента). В первой итерации реализуется набор основных требований, обеспечивающих базовую функциональность. Остальные итерации реализуются в порядке критичности требований для конечного пользователя. При появлении в середине итерации нового набора требований они откладываются до реализации следующей версии. В реальной жизни это допущение модели может нарушаться и допускается пересмотр требований.

В разных моделях этой группы итерации могут выполняться последовательно или с перекрытием. При последовательном выполнении каждая новая итерация начинается после завершения предыдущей. В модели с перекрытиями новая итерация начинается до завершения предыдущей итерации или когда первая стадия предыдущей итерации (проект) завершена примерно на 80%.

Инкрементная модель с перекрытием итераций

Характеристики	Преимущества
Анализ и проектирование выполняется для всей системы	Критические функции реализуются в первую очередь
Базовые функциональные требования реализуются первыми и тщательно тестируются	Критические функции тестируются более тщательно (т. к. проходят много циклов тестирования)
Остальные требования реализуются в последующих версиях	Наименее критичные задачи реализуются в последнюю очередь, что минимизирует последствия отказов из-за дефектов
Промежуточные версии пригодны для пользователя	Завершение первой версии окончательно утверждает требования и проект, что снижает риск проекта
	Раннее планирование и выполнение тестирования
	Раннее выявление пропущенных дефектов пользователем



Инкрементная модель с перекрытием итераций

Итерационные модели широко применяются для разработки коммерческих программных продуктов, которые развиваются в течение длительного периода времени, или для которых внешние требования изменяются слабо (компиляторы, ОС). Для согласования требований и проекта эти модели могут включать прототипирование.

В методологии Rational Unify Process (RUP) используется *итеративная контролируемая модель с приращениями (прототипированием)*. Каждая итерация этой модели имеет 4 стадии:

- *изучение* - обсуждение и определение набора функций для итерации;
- *развитие* - детальное проектирование;
- *конструирование* - создание полнофункционального продукта, определенного для этой итерации;
- *передача* - выпуск продукта, определенного для этой итерации.

Каждая следующая стадия итерации начинается до завершения предыдущей.

Риски, связанные с выбором модели	Когда лучше применять
Требования не полностью понятны	Требуется быстрая реализация основных возможностей
Требования не стабильны	Если проект системы можно естественным образом разделить на независимые части
Все возможности должны быть реализованы сразу	
Быстрые изменения технологии	

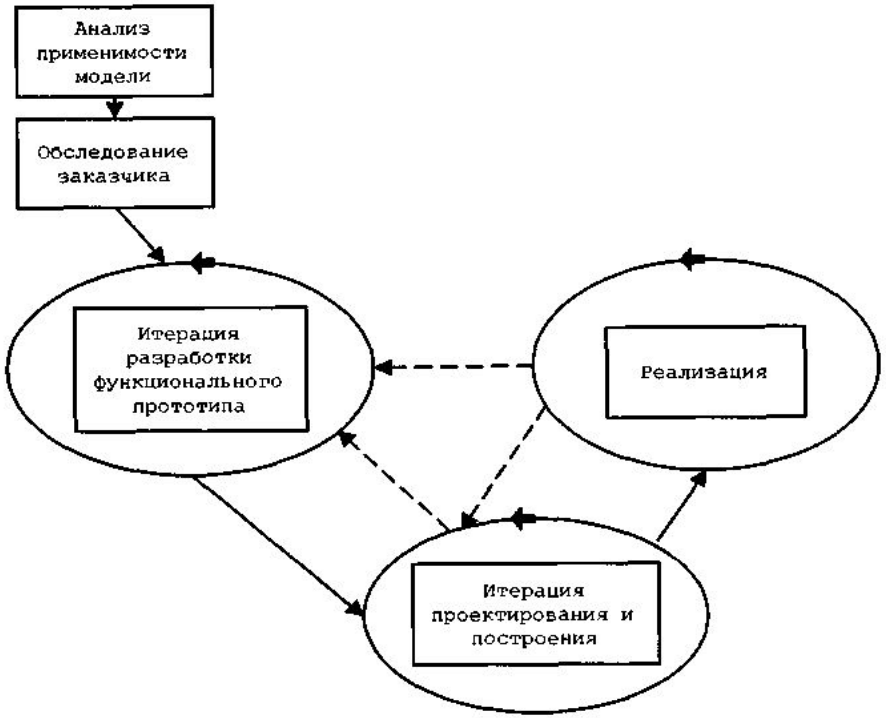
Модель эволюционного прототипирвоания

Эта модель основана на применении *эволюционного прототипирования* в рамках всего ЖЦ разработки (а не только для моделирования требований). В литературе она часто называется моделью *быстрой разработки приложений* (RAD от Rapid Application Development). Моделирование включает следующие шаги:

1. Анализ применимости модели. Изучение возможности применения модели для проекта.
2. Обследование заказчика. Изучение потребностей пользователя и разработка плана создания прототипа.
3. Итерация разработки функционального прототипа. Создание и согласование прототипа интерфейса пользователя, определение нефункциональных требований и стратегии реализации системы.
4. Итерация проектирования и построения. Построение протестированной системы, удовлетворяющей всем функциональным и нефункциональным требованиям. На шагах 3 и 4 разработчики выполняют определение прототипов, согласование сроков разработки, построение и проверку прототипов Эти шаги выполняются итеративно и включают три итерации: начальное ознакомление, уточнение и согласование.
5. Реализация. Установка системы в среде заказчика, разработка документации и обучение.

Модель эволюционного прототипирования

Модель применяется для разработки не критических бизнес-приложений, для которых наиболее важными являются функциональные возможности, а не надежность или производительность, и предполагает тесное взаимодействие с пользователем.



Характеристики	Преимущества
Гибкость, возможность быстро реагировать на изменение и расширение требований	Раннее выявление дефектов в интерфейсе
Приоритеты функциональных характеристик перед техническими (качества))	Быстрая демонстрация функциональных возможностей

Риски, связанные с выбором модели	Когда лучше применять
От разработчика требуется хорошее владение CASE – методами и инструментами.	Пользователи не могут четко сформулировать требования.
Разрабатываемое приложение должно быть не критичным.	Требуется ранняя демонстрация возможностей.
Требуется наличие мощных CASE - средств	

Компонентно-ориентированная модель

Компонентно-ориентированная модель является развитием спиральной модели и тоже основывается на эволюционной стратегии разработки. В этой модели модифицируется содержание квадранта моделирования-конструирования — оно отражает тот факт, что в современных условиях новая разработка должна основываться на повторном использовании существующих программных компонентов .

Программные компоненты, созданные в реализованных программных проектах, хранятся в библиотеке. В новом программном проекте, исходя из требований заказчика, выявляются кандидаты в компоненты. Далее проверяется наличие этих кандидатов в библиотеке. Если они найдены, то компоненты извлекаются из библиотеки и используются повторно. В противном случае создаются новые компоненты, они применяются в проекте и включаются в библиотеку.

Достоинства компонентно-ориентированной модели:

- уменьшает на 30% время разработки программного продукта;
- уменьшает стоимость программной разработки до 70%;
- увеличивает в полтора раза производительность разработки.

Модели процесса разработки ПП

В настоящее время широкое применение получают промышленные технологии создания ПП. Это разработки фирм, накопивших большой опыт создания ПО. Такие технологии представлены описаниями принципов, методов, применяемых процессов и операций и, как правило, поддерживаются набором CASE-средств (Computer- Aided Software Engineering), охватывают все этапы ЖЦ ПП и успешно применяются для решения практических задач. Рассмотрим *особенности моделей* жизненного цикла трех наиболее известных промышленных технологий.

- Microsoft Solution Framework (MSF) — методология разработки программного обеспечения фирмы «Microsoft», предназначенная для решения широкого круга задач. Технология масштабируема, т.е. настраиваема на решение задач любой сложности коллективом любой численности.
- Rational Unified Process (RUP) — разработка фирмы «Rational», долгое время успешно занимавшейся созданием CASE-средств, применяемых на различных этапах жизненного цикла продукта от анализа до тестирования и документирования. Аналогично MSF технология RUP универсальна, масштабируема и настраиваема на применение в конкретных условиях.
- Extreme Programming (XP) — методология экстремального программирования, активно развивающаяся в последнее время и предназначенная для решения относительно небольших задач относительно небольшими коллективами профессиональных разработчиков в условиях жестко ограниченного времени.
- RAD?

Модель Microsoft Solution Framework

Одна из особенностей технологии MSF состоит в том, что она ориентирована не просто на создание ПП, удовлетворяющего перечисленным требованиям, а на поиск решения проблем, стоящих перед заказчиком. Как правило, предъявляемые заказчиком требования направлены на устранение некоторых глубоких проблем; и неточность, неполнота, а также изменение требований в процессе разработки — следствие их недопонимания. Поэтому в технологии MSF большое внимание уделяется анализу проблем заказчика и разработке вариантов системы для поиска их решения.

Модель жизненного цикла MSF является некоторым гибридом каскадной и спиральной моделей, сочетая простоту управления каскадной модели с гибкостью спиральной. Модель жизненного цикла MSF ориентирована на *«вехи»* (milestones), т.е. ключевые точки проекта, характеризующие достижение какого-либо существенного результата. Этот результат может быть оценен и проанализирован, что подразумевает ответ на вопрос: «А достигли ли мы целей, поставленных на этом шаге?». В модели предусматривается наличие основных вех (завершение главных фаз модели) и промежуточных, отражающих внутренние этапы главных фаз.

Модель Microsoft Solution Framework

Основные *фазы модели MSF:*

Создание общей картины приложения (Envisioning). На этом этапе решаются следующие задачи:

- оценка существующей ситуации;
- определение состава команды, структуры проекта, бизнес-целей, требований и профилей пользователей;
- разработка концепции решения и оценка риска.

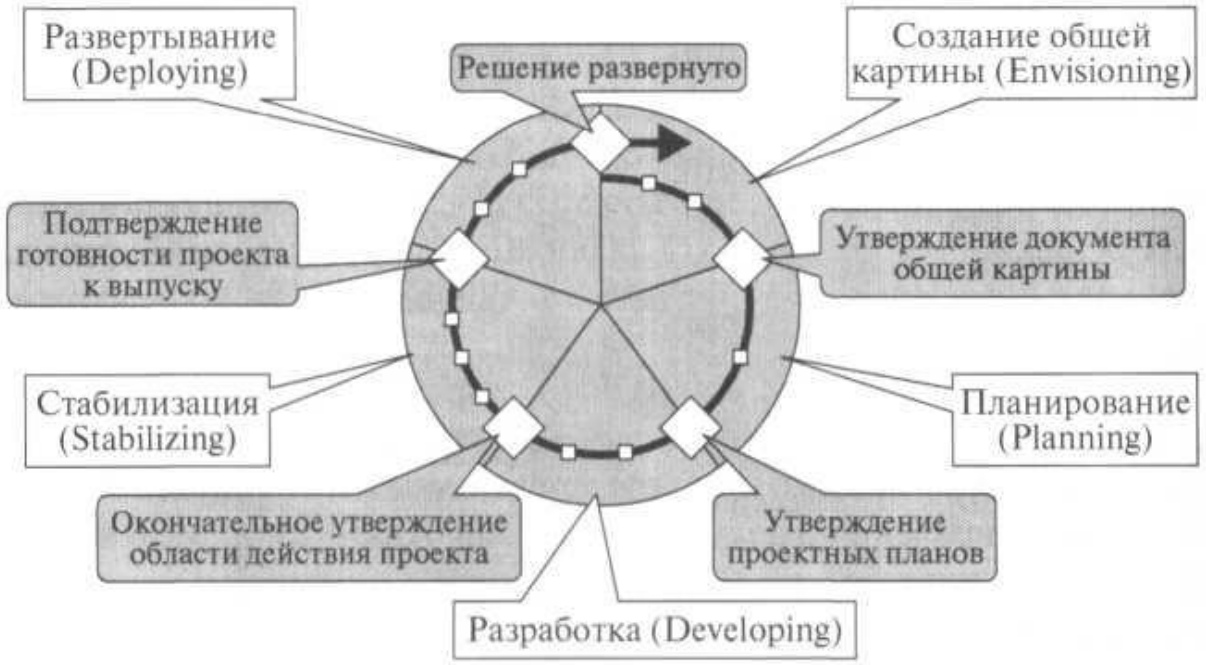
Устанавливаются две промежуточные вехи:

- «Организован костяк команды»
- «Создана общая картина решения».

Планирование (Panning). Включает планирование и проектирование продукта. На основе анализа требований разрабатывается проект и основные архитектурные решения, функциональные спецификации системы, планы и календарные графики; выбираются среды разработки, тестирования и пилотной эксплуатации. Этап состоит ИЗ трех стадий: концептуальное, логическое и физическое проектирование. На стадии *концептуального* проектирования задача рассматривается с точки зрения пользовательских и бизнес-требований и заканчивается определением набора сценариев использования системы. При *логическом* проектировании задача рассматривается с точки зрения проектной команды, решение представляется в виде набора сервисов. И уже на стадии *физического* проектирования задача рассматривается с точки зрения программистов, уточняются используемые технологии и интерфейсы.

Модель Microsoft Solution Framework

Разработка (Developing). Создается вариант решения проблемы в виде кода и документации очередного прототипа, включая спецификации и сценарии тестирования. Основная веха этапа — Окончательное утверждение области действия проекта». Продукт готов к внешнему тестированию и стабилизации. Кроме того, заказчики, пользователи, сотрудники службы поддержки и сопровождения, а также ключевые участники проекта могут предварительно оценить продукт и указать все недостатки, которые нужно устранить до его поставки.



Стабилизация (Stabilizing). Подготовка к выпуску окончательной версии продукта, доводка его до заданного уровня качества. 1нссь выполняется комплекс работ по тестированию (обнаружение И устранение дефектов), проверяется сценарий развертывания системы.

Развертывание (Deploying). Выполняется установка продукта и необходимых компонентов окружения, проводится его стабилизации в промышленных условиях и передача проекта в группу сопровождения, которая анализирует проект в целом на предмет уровня уновлетворенности заказчика.

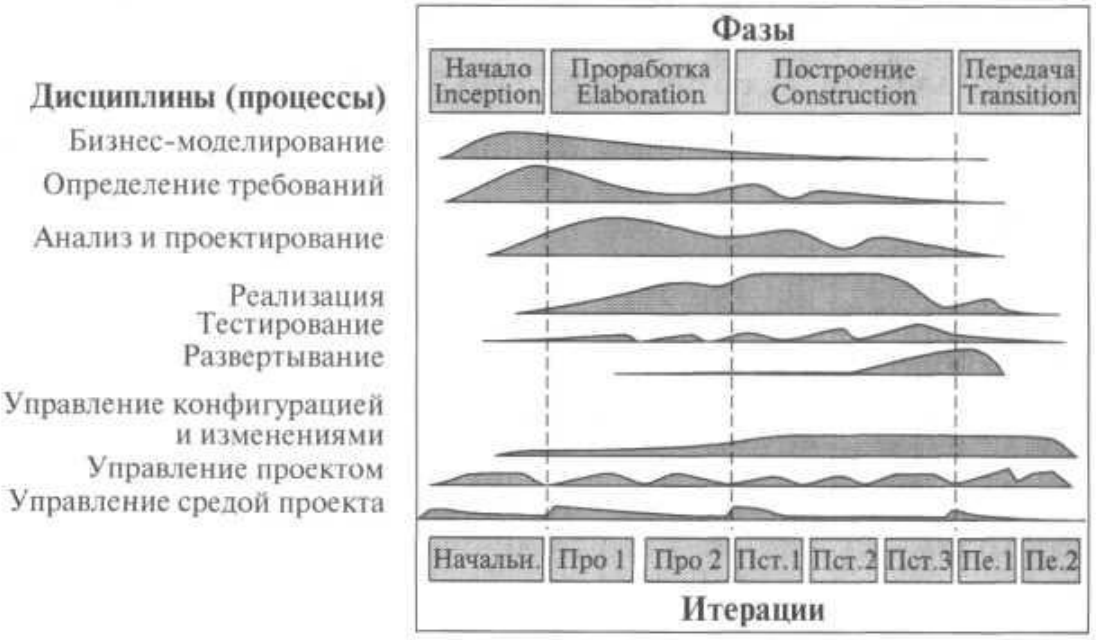
Модель Rational Unified Process

Является довольно сложной, детально проработанной итеративно-инкрементной моделью с элементами каскадной модели. В модели RUP выделяются четыре основные фазы, а также девять видов деятельности (процессов). Кроме того, в модели описывается ряд практик, которые следует применять или руководствоваться для успешного выполнения проекта. RUP ориентирована на поэтапное моделирование создаваемого продукта с помощью UML (Unified Modeling Language — унифицированный язык моделирования).

Основными *фазами модели RUP* являются следующие:

Начало проекта (Inception). Определяются основные цели проекта, его бюджет, основные средства его выполнения: технологии, инструменты, ключевой персонал; составляются предварительные планы проекта. Основная цель этой фазы — достичь компромисса между всеми заинтересованными лицами относительно задач проекта.

Проработка (Elaboration). Основная цель этой фазы — на базе основных, наиболее существенных требований разработать стабильную базовую архитектуру продукта, которая позволит решать поставленные перед системой задачи и в дальнейшем будет использована как основа для разработки системы.



Построение (Construction). Основная цель этой фазы — детальное прояснение требований и разработка системы, удовлетворяющей им, на основе спроектированной ранее архитектуры.

Передача (Transition). Цель фазы — сделать систему полностью доступной конечным пользователям. Здесь происходит окончательное развертывание системы в ее рабочей среде, подгонка мелких деталей под нужды пользователей.

Модель Rational Unified Process

В рамках каждой фазы возможно проведение нескольких *итераций*, количество которых определяется сложностью выполняемого проекта.

Основные процессы (деятельности) RUP делятся на пять рабочих и четыре поддерживающих.

К *рабочим процессам* относятся следующие.

- *Моделирование предметной области* (Business Modeling — бизнес-моделирование). Цель этой деятельности — понять бизнес-контекст, в котором должна будет работать система (и убедиться, что все заинтересованные лица понимают его одинаково), предвидеть возможные проблемы, оценить их возможные решения и последствия для бизнеса организации, в которой будет работать система.
- *Определение требований* (Requirements). Цель — понять, что должна делать система, определить границы системы, основу для планирования проекта и оценок ресурсозатрат в нем.
- *Анализ и проектирование* (Analysis and Design). Выработка архитектуры системы на основе ключевых требований, создание проектной модели, представленной в виде UML-диаграмм, описывающих программный продукт с различных точек зрения.
- *Реализация* (Implementation). Разработка исходного кода компонентов системы, тестирование и интегрирование компонент.
- *Тестирование* (Test). Общая оценка дефектов продукта и его качества в целом, оценка степени соответствия разработанного продукта исходным требованиям.

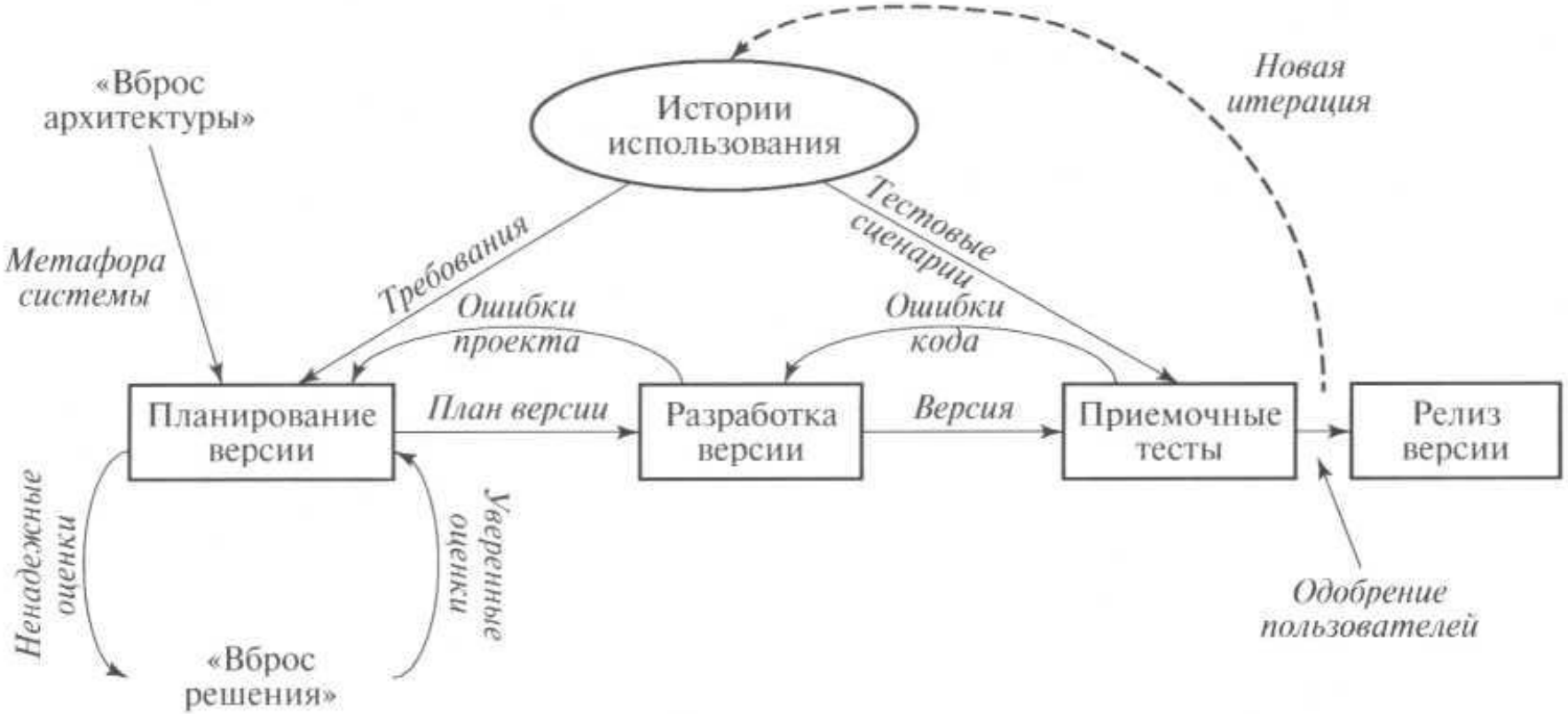
Модель Rational Unified Process

Поддерживающими процессами являются следующие четыре процесса.

- *Развертывание* (Deployment). Цель — развернуть систему в ее рабочем окружении и оценить ее работоспособность.
- *Управление конфигурациями и изменениями* (Configuration and Change Management). Определение элементов, подлежащих хранению, и правил построения из них согласованных конфигураций, поддержание целостности текущего состояния системы, проверка согласованности вносимых изменений.
- *Управление проектом* (Project Management). Включает планирование, управление персоналом, обеспечение связей с другими заинтересованными лицами, управление рисками, отслеживание текущего состояния проекта.
- *Управление средой проекта* (Environment). Настройка процесса под конкретный проект, выбор и смена технологий и инструментов, используемых в данном проекте.

Модель Extreme Programming

Является итерационно-инкрементной моделью быстрого создания и модификации протопопов программного продукта, которые должны удовлетворять очередному требованию (user story).

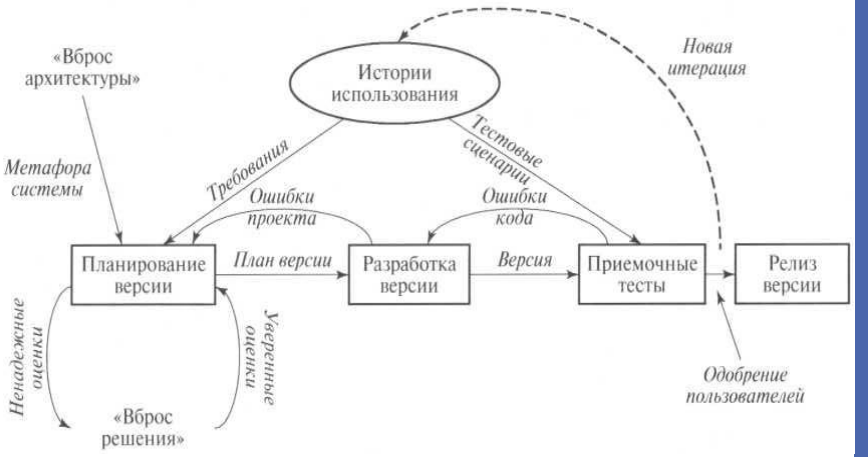


Модель Extreme Programming

Модель XP включает в себя выполнение следующих *основных фаз.*

«Вброс» архитектуры — начальный этап проекта, на котором создается видение продукта, принимаются основные решения по архитектуре и применяемым технологиям. Результатом начального этапа является метафора (metaphor) системы, которая в достаточно простом и понятном команде виде должна описывать основной механизм работы системы.

Истории использования (User Story) — этап сбора требований, записываемых на специальных карточках в виде сценариев выполнения отдельных функций. Истории использования являются требованиями для планирования очередной версии и разработки приемочных тестов (Acceptance tests) для ее проверки.



Планирование версии (релиза). Проводится на собрании с участием заказчика путем выбора User Stories, которые войдут в следующую версию. Одновременно принимаются решения, связанные с реализацией версии. Цель планирования — получение оценок того, что и как можно сделать за 1—3 недели создания следующей версии продукта.

Разработка версии (релиза) проводится в соответствии с планом и включает только те функции, которые были отобраны на этапе планирования.

Тестирование версии (релиза) проводится с участием заказчика, который ранее участвовал в составлении тестов.

Выпуск релиза — разработанная версия передается заказчику для использования или бета-тестирования.

По завершении цикла делается переход на следующую итерацию разработки.

Модель Extreme Programming

Особенности модели жизненного цикла XP проясняют основные принципы этого метода, и прежде всего это *принципы «живой» разработки ПО*, отраженные в *манифесте «живой» разработки ПО:*

- люди и их общение более важны, чем процессы и инструменты;
- работающая программа более важна, чем исчерпывающая документация;
- сотрудничество с заказчиком более важно, чем обсуждение деталей контракта;
- отработка изменений более важна, чем следование планам.

Модель Extreme Programming

Основные правила модели XP также характеризуют ее особенности и основные техники применения:

- живое планирование (planning game), направленное на то, чтобы как можно быстрее определить объем работ, который нужно сделать до разработки следующей версии ПО; решение принимается на основе, в первую очередь, бизнес-приоритетов заказчика и, во-вторую, технических оценок; при этом планы изменяются, как только они начинают расходиться с действительностью или пожеланиями заказчика;
- частая смена версий (small releases): первая работающая версия должна появиться как можно быстрее и тут же должна использоваться, следующие версии подготавливаются через достаточно короткие промежутки времени;
- простые проектные решения (simple design): в каждый момент времени система конструируется так просто, насколько это возможно, новые функции добавляются только после ясной и обоснованной просьбы, вся лишняя сложность удаляется, как только обнаруживается;
- разработка на основе тестирования (test-driven development) означает, что сначала пишутся тесты, демонстрирующие основные возможности системы, чтобы можно было увидеть, что система действительно заработала, а потом уже реализуются модули системы и таким образом, чтобы тесты срабатывали; при этом тесты пишутся заказчиками (заранее);
- постоянная переработка (refactoring) системы для устранения излишней сложности, увеличения понятности кода, повышения его гибкости, при этом предпочтение отдается более элегантным и гибким решениям по сравнению с просто дающими нужный результат;
- программирование парами (pair programming): весь код пишется двумя программистами на одном компьютере, что повышает его качество (отсутствие ошибок, понятность, читаемость);
- постоянная интеграция (continuous integration): система собирается и проходит интеграционное тестирование как можно чаще, по несколько раз в день, каждый раз, когда заканчивается реализация очередной функции.

Методологии управления процессами разработки ПП

Традиционно для упорядочения и ускорения программных разработок предлагались строго упорядочивающие тяжеловесные (heavyweight) процессы. В этих процессах прогнозируется весь объем предстоящих работ, поэтому они называются прогнозирующими (predictive) процессами. Порядок, который должен выполнять при этом человек-разработчик, чрезвычайно строг — «шаг вправо, шаг влево — виртуальный расстрел!» Иными словами, человеческие слабости в расчет не принимаются, а объем необходимой документации способен отнять покой и сон у «совестливого» разработчика.

В последние годы появилась группа новых, облегченных (lightweight) процессов. Теперь их называют подвижными (agile) или гибкими процессами. Они привлекательны отсутствием бюрократизма, характерного для тяжеловесных (прогнозирующих) процессов. Новые процессы должны воплотить в жизнь разумный компромисс между слишком строгой дисциплиной и полным ее отсутствием. Иначе говоря, порядка в них достаточно для того, чтобы получить разумную отдачу разработчиков. *Гибкие процессы требуют меньшего объема документации и ориентированы на человека*. В них явно указано на необходимость использования природных качеств человеческой натуры (а не на применение действий, направленных наперекор этим качествам).

Более того, гибкие процессы учитывают особенности современного заказчика, а именно частые изменения его требований к программному продукту. Известно, что для прогнозирующих процессов частые изменения требований подобны смерти. В отличие от них гибкие процессы адаптируют изменения требований и даже выигрывают от этого. Словом, гибкие процессы имеют адаптивную природу.

Таким образом, в современной инфраструктуре программной инженерии существует два семейства процессов разработки:

- семейство прогнозирующих (тяжеловесных) процессов;
- семейство адаптивных (гибких, облегченных) процессов.

У каждого семейства есть свои достоинства, недостатки и область применения:

- адаптивный процесс используют при частых изменениях требований, малочисленной группе высококвалифицированных разработчиков и грамотном заказчике, который согласен участвовать в разработке;
- прогнозирующий процесс применяют при фиксированных требованиях и многочисленной группе разработчиков разной квалификации.

Гибкие методологии разработки

- RAD Crystal
- Адаптивная разработка
- Scrum
- Kanban
- Fdd
- Dsdm
- Бережливая разработка
- Союз всех гибких + манифест

Второйсеместр!