**Оглавление**

[**Оглавление** 1](#_Toc56514761)

[**Лекция 5 Жизненный цикл программного продукта** 2](#_Toc56514762)

[**5.1 Введение** 2](#_Toc56514763)

[**5.2 Понятие жизненного цикла программного продукта** 3](#_Toc56514764)

[**5.3 Базовые понятия жизненного цикла ПО в программной инженерии** 8](#_Toc56514765)

[**5.4 История развития стандартов моделей жизненного цикла ПО** 10](#_Toc56514766)

[**5.5 Действующие стандарты на жизненный цикл программного продукта** 11](#_Toc56514767)

[**5.5.1.** **Стандарт ISO/IEC 12207** 11](#_Toc56514768)

[**5.5.2 Стандарт ISO 15504,** 17](#_Toc56514769)

[**5.5.3 Влияние развития информационных технологий на появление новых ИТ-стандартов** 19](#_Toc56514770)

## **Лекция 5 Жизненный цикл программного продукта**

### **5.1 Введение**

Материал предыдущих лекций позволяет сделать несколько выводов:

1. Современные требования управления бизнесом, производством и общественными процессами требуют всё более сложных программных продуктов;
2. Работа над созданием программных продуктов, удовлетворяющих запросам потребителей, стала сложным делом, требующим объединения многих специалистов разного профиля.
3. Соответствующая деятельность в полной мере может считаться проектной. Чтобы она была успешной, к ней надо подходить (и соответственно организовывать по современным канонам) как к проекту.
4. Требования к качеству, стоимости и срокам «изготовления» программного продукта (или шире, к программной системе) непрерывно меняются по мере развития вычислительной техники и подходов к программированию. Соответственно, меняются и подходы к реализации проекта.

Для организации слаженной работы коллективов исполнителей программных проектов, а также при подготовке очередных поколений требовалось наличие методологической[[1]](#footnote-1) основы технологии разработки программного обеспечения и программных систем[[2]](#footnote-2), позволяющей за счёт детального описания процесса разработки системы, позволяет упорядочить этот процесс, особое место в котором занимает планирование. Такой основой является понятие жизненного цикла программного продукта[[3]](#footnote-3) И именно потребность в таком упорядочении породила интерес к изучению понятия *жизненного цикла*.

### **5.2 Понятие жизненного цикла программного продукта**

Понятие жизненного цикла программного обеспечения появилось, когда программистское сообщество осознало необходимость перехода от кустарных ремесленнических методов разработки программ к технологичному промышленному их производству. Как обычно происходит в подобных ситуациях, программисты попытались перенести опыт других индустриальных производств в свою сферу. В частности, было заимствовано понятие жизненного цикла. Жизненный цикл промышленного изделия – это:

* последовательность этапов (фаз, стадий): инициации, проектирования, изготовления образца, организация производства, серийное производство, эксплуатация, ремонт, вывод из эксплуатации;
* состоящих из технологических процессов, действий и операций.

Организация промышленного производства с позиции жизненного цикла позволяет рассматривать все его этапы во взаимосвязи, что ведет к сокращению сроков, стоимости и трудозатрат.

Рассмотрим правомерность применения методологии жизненного цикла к сфере создания программных средств.

Традиционным образцом для *методологий программирования* являются инженерные дисциплины, такие, например, как строительство и машиностроение, где особое внимание уделяется планированию, которое предшествует непосредственному материальному *производству*.

Инженеры разрабатывают целый ряд чертежей, в которых **точно** указывается, что именно должно быть построено и **как соединить** все составляющие в единое целое. Во время работы над чертежами принимается много различных проектных решений. Затем **чертежи передаются другой группе специалистов**, часто вообще в другую компанию, которая будет заниматься собственно строительством. Принято считать, что строители в точности воспроизводят все, что было обозначено на чертежах. В действительности строители сталкиваются с некоторыми проблемами, однако, как правило, они вполне разрешимы.

Есть место в этом процессе и *декомпозиции*, т.е. разбиению задачи материального *производства* на подзадачи. Чертежи, где представлены отдельные элементы строительства, ложатся в основу подробного чертежа, который позволяет определить конкретные задачи и зависимости между ними. А это, в свою очередь, дает возможность рассчитать стоимость и временные рамки строительства в целом. Кроме того, здесь же подробно описывается, каким образом строители должны выполнять свою работу. Благодаря этому работа строителей становится еще менее интеллектуальной, хотя, разумеется, нередко требует очень хороших навыков ручного труда.

А правомерно ли переносить такой поход из сферы материального *производства* на программирование? Если да, то мы должны с самого начала разграничить два вида деятельности в этой области:

* *проектирование*, требующее креативного мышления: разбора вариантов решения, оптимизации и других творческих элементов;
* *производство*, неукоснительно следующее ранее составленному проекту, в котором можно считать осуществленной замену творчества технологией (что аналогично переходу к технологиям от ремесленничества в области материального *производства*).

**Эта заманчивая перспектива, к сожалению, не может быть в полной мере перенесена на область*****разработки* программных изделий, которые с начала и до конца остаются искусственными объектами мыслительной деятельности,*****артефактами*.**

**Не только** ***проектирование*, но и простое кодирование требует от программиста креативного мышления.**

На каждом уровне *развития проекта* приходится разбирать варианты, оптимизировать, создавать новое, **а не просто следовать скрупулезному плану**. К тому же приходится еще **решать задачи проверки** пройденных этапов *разработки* и уже наработанных фрагментов.

Тем не менее потребность в создании сложных программных систем приводит к необходимости регламентации творческого процесса. И каждая ***методология программирования*** пытается построить процесс *разработки* таким образом, чтобы **минимизировать** (**!**) творческий элемент в случаях **рутинной** работы. Иными словами, методологии стремятся сделать так, чтобы сокращалось число ошибок, чтобы как можно раньше переходить если не к *производству*, то хотя бы к тому, что является **аналогом** *производства* при *разработке* программ. Отсюда попытки разграничить план и конструкцию программы, спецификации пользовательской потребности и план, выбор инструментов для работы программиста и саму работу. Это же приводит к появлению регламентов и предписаний, следование которым уменьшает вероятность ошибочных решений.

По существу, любая методология представляет собой набор регламентов и предписаний. **В частности, любая методология выстраивает свою*****модель жизненного цикла* как основу для этих соглашений**.

Мы стараемся показать, что понятие *жизненного цикла* само по себе от методологий не зависит. И в "хаотическом" конструировании ранних программных продуктов, и в современных "жестких" методологиях, и в так называемых "облегченных" (lightweight) методологиях можно указать на *жизненный цикл*. И хотя форма представления *жизненных циклов* в разных случаях различна до неузнаваемости, можно утверждать, что в основе любых представлений *разработки* и *сопровождения* программных изделий лежат общие процессы, которые в конечном итоге ведут проекты от их замыслов к удовлетворению потребностей пользователя. Любая методология предписывает организацию этих общих процессов. Поэтому *модели жизненного цикла* рассматриваются как развитие понятий, связанных с общими процессами *разработки* программных систем.

Последнее замечание. Как известно, любое промышленное изделие зарождается и неизбежно заканчивает свой жизненный путь. ПО – не вещь, в каком же смысле можно говорить о его «кончине»?

Аналогия жизненного цикла программного обеспечения с техническими системами имеет более глубокие корни, чем это может показаться на первый взгляд. Программы не подвержены физическому износу, но в ходе их эксплуатации обнаруживаются ошибки (неисправности), требующие исправления. Ошибки возникают также от изменения **условий использования** программы. Последнее же является принципиальным свойством программного обеспечения, иначе оно теряет свой смысл. Поэтому правомерно говорить о старении программ (не о физическом старении, а о моральном) и даже об «окончании» их жизни.

Необходимость внесения изменений в действующие программы как из-за обнаруживаемых ошибок, так и по причине развития требований приводит по сути дела к тому, что разработка программного обеспечения продолжается после передачи его пользователю и в течение всего времени жизни программ. Деятельность, связанная с решением довольно многочисленных задач такой продолжающейся разработки, получила **название сопровождения программного обеспечения**(рис.5.1).

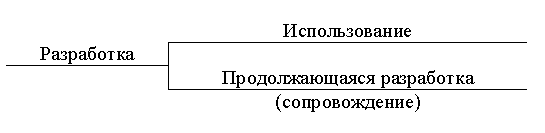


Рис. 5.1. Схема жизненного цикла ПО на этапах разработки, использования и сопровождения.

Исторически развитие концепций жизненного цикла связано с поиском для него адекватных моделей. Как и всякая другая, модель жизненного цикла является абстракцией реального процесса, в которой опущены детали, несущественные с точки зрения назначения модели. Различие назначений применения моделей определяет их разнообразие.

**Основные причины, из-за которых нужно изучать вопросы моделирования жизненного цикла программного обеспечения, можно сформулировать следующим образом**.

**Во-первых**, это знание даже для **непрофессионального** программиста помогает понять, на что можно рассчитывать при заказе или приобретении программного обеспечения и что нереально требовать от него. В частности, неудобные моменты работы с программой, ее ошибки и недоработки обычно устраняются в ходе продолжающейся разработки, и есть основания ожидать, что последующие версии будут лучше. Однако кардинальные изменения концепций программы — задача другого проекта, который совсем необязательно будет во всех отношениях лучше данной системы.

**Во-вторых**, модели жизненного цикла — **основа знания технологий программирования и инструментария**, поддерживающего их. Программист всегда применяет в своей работе инструменты, но **квалифицированный программист знает, где, когда и как их применять**. Именно в этом помогают понятия моделирования жизненного цикла: любая технология базируется на определенных представлениях о жизненном цикле, выстраивает свои методы и инструменты вокруг фаз и этапов жизненного цикла.

**В-третьих**, общие знания того, как развивается программный проект, дают наиболее надежные ориентиры для его **планирования**, позволяют экономнее расходовать ресурсы, добиваться более высокого качества управления. Все это относится к сфере профессиональных обязанностей **руководителя** программного проекта.

**В-четвёртых**, *жизненный цикл* следует рассматривать как **основу деятельности менеджера программного проекта**: с ним связываются и *цели проекта*— окончательные и промежуточные, распределение и *контроль* расходования ресурсов, а также все другие аспекты управления *развитием проекта*. Прежде всего эта привязка обусловлена разбиением *производства* **любой программы на этапы**, которые ассоциируются с определенными видами *работ* или функций, выполняемых разработчиками в тот или иной момент *развития проекта*. Этапы характеризуются направленностью выполняемых функций на достижение локальных (для этапа) целей проекта. Необходимость отслеживания целей приводит к понятию **контрольных точек** — моментов *разработки*, когда осуществляется подведение промежуточных итогов, осмысление достигнутого и ревизия сделанных ранее предположений.

Впервые о жизненном цикле ПО заговорили в 1968 г. в Лондоне, где состоялась встреча 22-х 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.

Вокруг этой концепции было много споров. 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.

### **5.3 Базовые понятия жизненного цикла ПО в программной инженерии**

1. Все **продукты** процессов программной инженерии представляют собой некоторые описания, а именно:
   1. Тексты требований к разработке;
   2. Согласования договорённостей с заказчиком;
   3. Описания архитектуры и структуры данных;
   4. Тексты программ;
   5. Документацию;
   6. Инструкции и т.п.
2. Главными ресурсами разработки ПС являются **сроки, время и стоимост**ь, которые необходимо правильно использовать на процессах ЖЦ;
3. ЖЦ ПО следует представлять в виде четырёх обобщённых фаз:
   1. Концепция (инициация, идентификация, отбор)[[4]](#footnote-4);
   2. Анализ(определение);
   3. Выполнение (практическая реализация или внедрение, производство и развёртывание, проектирование или конструирование, сдача в эксплуатацию);
   4. Закрытие (завершение, включая оценивание).
4. Так как эти фазы определены очень широко, как правило, для каждой категории и подкатегории проекта внутри каждой фазы выделяют несколько подфаз. В общем случае фазы и подфазы не обязательно должны выполняться линейно и последовательно;
5. Для жизненного цикла можно выделить (и применять) понятия модели ЖЦ и методологии (метода):
   1. Модель ЖЦ – это концептуальный взгляд на его организацию, что подразумевает описание фаз и принципы перехода между ними;
   2. Методология ЖЦ задаёт (описывает):
      1. Комплекс работ по фазам,
      2. Детальное содержание этих работ,
      3. Ролевую ответственность специалистов на всех этапах ЖЦ,
      4. Лучшие практики в рамках модели и методологии, позволяющие максимально эффективно воспользоваться ими.

Полезные ссыпки на материалы в сети Интернет:

* <http://www.computer-museum.ru/books/n_collection/models.htm>
* <http://www.intuit.ru/studies/courses/38/38/info>

### **5.4 История развития стандартов моделей жизненного цикла ПО**

Естественно, что такое важное методологическое средство невозможно было не формализовать без разработки соответствующих стандартов. Также ясно, что эти стандарты должны были эволюционировать во времени. Коротко представим историю развития наиболее известных стандартов в этой области[[5]](#footnote-5).

**1985 (уточнен в 1988 г.) DOD-STD-2167 А – Разработка программных средств для систем военного назначения**. Первый формализованный и утвержденный стандарт жизненного цикла для проектирования ПС систем военного назначения по заказам Министерства обороны США. Этим документом регламентированы 8 фаз (этапов) при создании сложных критических ПС и около 250 типовых обязательных требований к процессам и объектам проектирования на этих этапах.

**1994г. MIL-STD-498. Разработка и документирование программного обеспечения**. Принят Министерством обороны США для замены DOD-STD-2167 A и ряда других стандартов. Он предназначен для применения всеми организациями и предприятиями, получающими заказы Министерства обороны США. В 1996 г. утверждено очень подробное (407 стр.) руководство “Применение и рекомендации к стандарту MIL-STD-498”. Основную часть составляют 75 подразделов — рекомендаций по обеспечению и реализации процессов ЖЦ сложных критических ПС высокого качества и надежности, функционирующих в реальном времени.

**1995г. IEEE 1074. Процессы жизненного цикла для развития программного обеспечения**. Охватывает полный жизненный цикл ПС, в котором выделяются **шесть** крупных базовых **процессов**. Эти процессы детализируются **16** частными процессами. В последних имеется еще более мелкая детализация в совокупности на **65 процессов-работ**.

Содержание каждого частного процесса начинается с описания общих его функций, задач и перечня действий — работ при **последующей детализации**. Для каждого процесса в стандарте представлена входная и результирующая информация о его выполнении и краткое описание сущности процесса. В стандарте внимание сосредоточено преимущественно на непосредственном создании ПС и на процессах предварительного проектирования. В приложении представлены четыре варианта адаптации максимального состава компонентов ЖЦ ПС к конкретным особенностям типовых проектов.

Между тем, разработка стандартов ЖЦ и их практическое применение сталкивались с рядом проблем:

* Внедрение стандартов требовало вложения значительных средств, что не всегда окупалось.
* Было неясно, все ли требуемые процессы надо выполнять и в какой мере
* Различные типы ПО (ИС, реального времени, бизнес системы), различные требования
* Высокая динамика отрасли и устаревание стандартов
* Терминологическая неоднозначность различных корпоративных стандартов
* Во многих случаях применение стандартов было вызвано только требованиями заказчиков, хотя на практике превращалось в тормоз и гробило выполнение проектов.

## **5.5 Действующие стандарты на жизненный цикл программного продукта**

### **5.5.1.** **Стандарт ISO/IEC 12207**

Разрешением проблем стандартизации ЖЦ ПО явилась разработка и принятие в 1995 г. стандарта ISO/IEC 12207 - Information Technology - Software Life Cycle Processes (ISO - International Organization of Standardization - Международная организация по стандартизации; IEC - International Electrotechnical Commission - Международная электротехническая комиссия). **В 2000 г. он был принят в России как ГОСТ 12207. Процессы жизненного цикла программных средств**.

Стандарт ISO 12207 разрабатывался с учетом лучшего мирового опыта на основе вышеперечисленных стандартов. Он был задуман как каркас (framework), имеющий чёткие связи с **окружением программной инженерии** – **программным и техническим обеспечением, исполнителями и деловой практикой**.

Основными результатами стандарта ISO 12207 являются:

* Введение единой терминологии по разработке и применению ПО (предназначен не только для разработчиков, но и для заказчиков, пользователей, поставщиками программных и аппаратных средств и других заинтересованных лиц).
* Разделение понятий ЖЦ ПП и модели ЖЦ ПО. ЖЦ ПП в стандарте вводится как полная совокупность всех процессов и действий по созданию и применению ПО, а модель ЖЦ – **конкретный вариант** организации ЖЦ, обоснованно (разумно) выбранный для каждого конкретного случая
* Описание организации ЖЦ и его структуры (процессов)
* Выделение процесса адаптации стандарта для построения конкретных моделей ЖЦ.

Обращаем внимание на следующие важные особенности стандарта:

* подчёркивает различие понятий жизненного цикла программного обеспечения и **моделью** жизненного цикла ПО;
* выделяет процесс адаптации стандарта для **конкретных моделей** ЖЦ;
* Не обязывает использовать определённую модель ЖЦ ПП или конкретную методологию разработки ПП. Поэтому ISO выпускает специальные стандарты и процедуры, дополняющие стандарт 12207, которыми могут (**должны!?**) руководствоваться организации (пользователи).

Рассмотрим некоторые понятия и детали данного стандарта или связанные с ним.

**Программный продукт (software product)**: Набор машинных программ, процедур и, возможно, связанных с ними документации и данных.

**Жизненный цикл программного продукта (software life cycle)** – это непрерывный процесс, который начинается с момента принятия решения о необходимости его создания и заканчивается в момент его полного изъятия из эксплуатации

**Процесс (process) -** Набор взаимосвязанных работ, которые преобразуют **исходные** данные в **выходные** результаты.

Под **Организацией жизненного цикла** – понимается представление его как совокупности взаимодействующих процессов. При этом каждый процесс представляется как набор действий, которые, в свою очередь, могут быть разбиты на отдельные задачи.

**Структура жизненного цикла** – представление процессов ЖЦ в виде иерархического дерева с точки зрения их соподчинённости и важности, на верхнем уровне которого находятся три группы (рис.5.2):

1. Основные;
2. Вспомогательные (поддерживающие),
3. Организационные и
4. Адаптация.

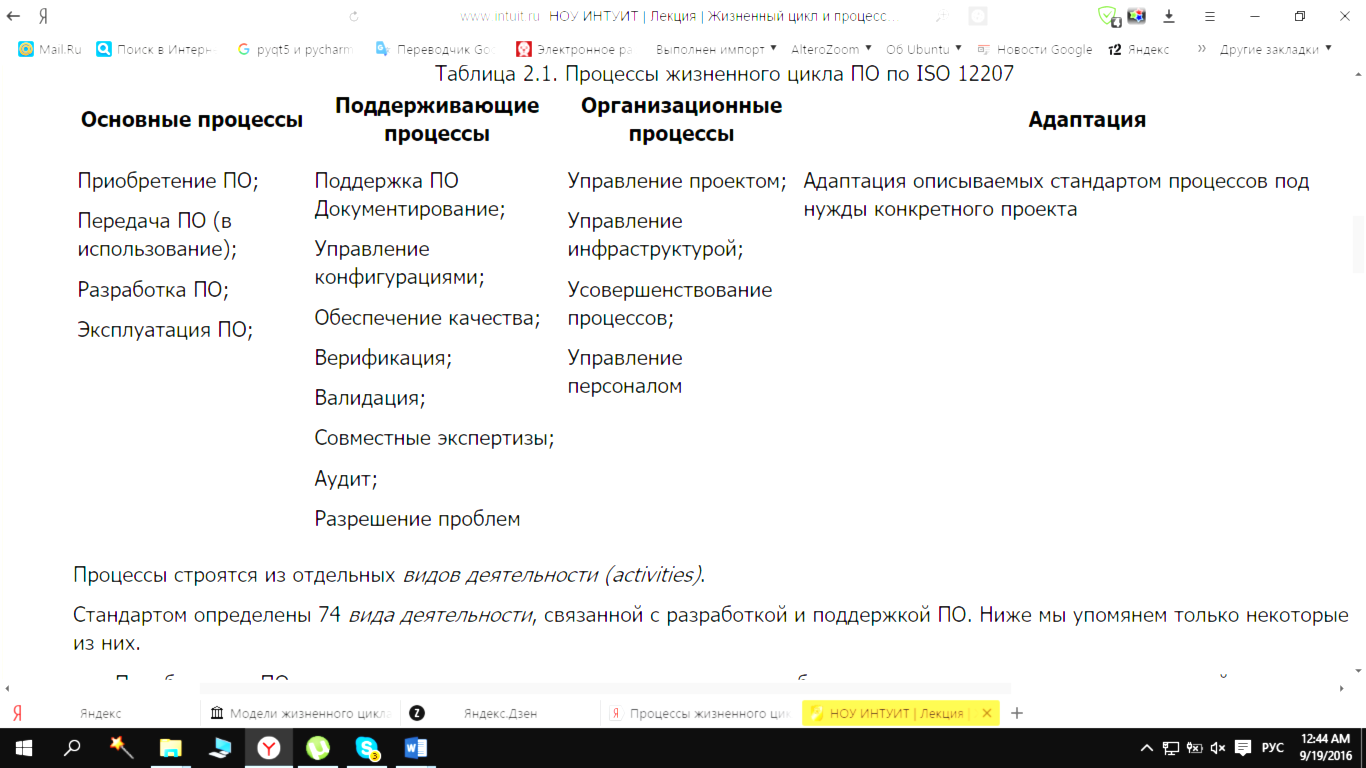


Рис. 5.2а. Классификация процессов ЖЦ согласно стандарта ISO12207

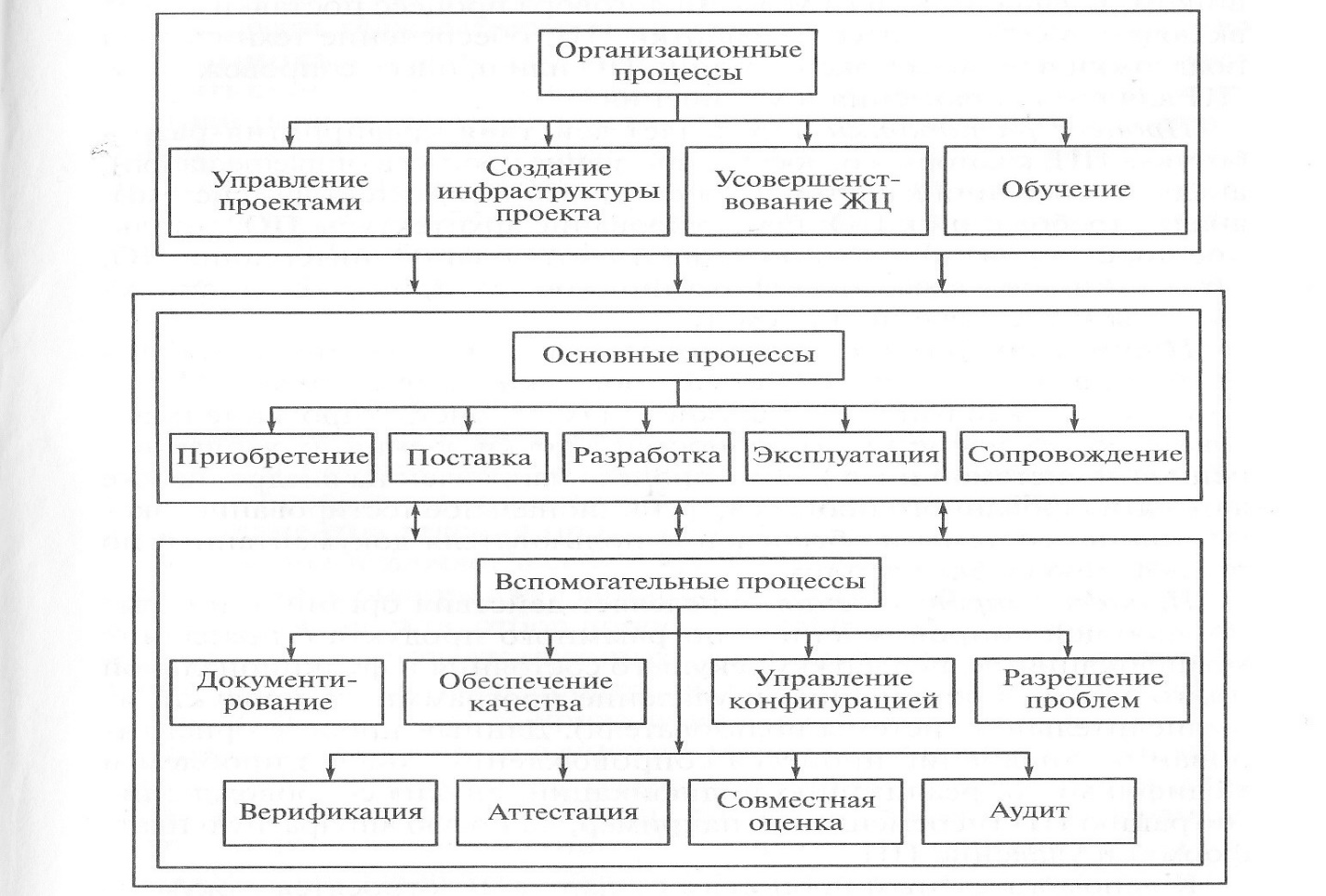


Рис. 5.2.б. Иерархия процессов жизненного цикла (стандарт ISO/IEC 12207)

Стандартом определены 74 *вида деятельности*, связанной с разработкой и поддержкой ПО. Ниже мы упомянем только некоторые из них.

* Приобретение ПО включает такие деятельности, как инициация приобретения, подготовка запроса предложений, подготовка контракта, анализ поставщиков, получение ПО и завершение приобретения.
* Разработка ПО включает развертывание *процесса разработки*, анализ системных требований, проектирование программно-аппаратной системы в целом, анализ требований к ПО, проектирование архитектуры ПО, детальное проектирование, кодирование и отладочное тестирование, интеграцию ПО, квалификационное тестирование ПО, системную интеграцию, квалификационное тестирование системы, развертывание (установку или инсталляцию) ПО, поддержку процесса получения ПО.
* Поддержка ПО включает развертывание процесса поддержки, анализ возникающих проблем и необходимых изменений, внесение изменений, экспертизу и передачу измененного ПО, перенос ПО с одной платформы на другую, изъятие ПО из эксплуатации.
* Управление проектом включает запуск проекта и определение его рамок, планирование, выполнение проекта и надзор за его выполнением, экспертизу и оценку проекта, свертывание проекта.

Каждый *вид деятельности* нацелен на решение одной или нескольких **задач (tasks)**. Всего определено 224 различные задачи. Например:

* Развертывание *процесса разработки* состоит из определения *модели жизненного цикла*, документирования и контроля результатов отдельных работ, выбора используемых стандартов, языков, инструментов и пр.
* Перенос ПО между платформами состоит из разработки плана переноса, оповещения пользователей, выполнения анализа произведенных действий и пр.

#### **5.5.1.1 Подробное описание основных процессов**

1. Процесс приобретения.
2. Процесс поставки.
3. Процесс разработки.
4. Процесс эксплуатации.
5. Процесс сопровождения.

#### **5.5.1.2 Понятие об адаптации стандартов ЖЦ**

Адаптация стандарта подразумевает применение требований стандарта к конкретному проекту. Например, для построения внутрикорпоративных регламентов ведения проектов по созданию ПО.

Также под адаптацией понимается выбор модели (или комбинации моделей) ЖЦ и применение соответствующих методологий, детализирующих процедуры выполнения процессов, работ и задач в рамках заданных границ ЖЦ ПО с учётом организационной структуры и ролевой ответственности, а конкретной организации или проектной группе.

Существует ещё один стандарт ЖЦ, освещающий вопросы организации процессов ЖЦ системного уровня (Live Cycle Process – System - **Процессы жизненного цикла Систем**) и включающий специальный процесс «Tailoring», то есть настройку, адаптацию ЖЦ к конкретным требованиям и ограничениям, существующим или принятым в конкретной организации.

В 2002 году Международная организация по стандартизации и Международная электротехническая комиссия выпустили результат многолетней работы — стандарт ISO/IEC 15288:2002 (см. русскоязычный аналог ГОСТ Р ИСО МЭК 15288-2005). (Системная Инженерия. Процессы жизненного цикла систем. Этапы модели ЖЦ ИТ: планирование, проектирование, разработка и внедрение, эксплуатация, поддержка, утилизация, обновление. Цели этапов жизненного цикла информационной системы (ЖЦ ИС). Шаблон адаптации модели ЖЦ ИС.)[[6]](#footnote-6)

Согласно стандарту, процессы и действия жизненного цикла определяются, соответствующим образом настраиваются и используются в течение стадии жизненного цикла, для полного удовлетворения целей и результатов на этой стадии. В различных стадиях жизненного цикла могут принимать участие разные организации. Не существует единой универсальной модели жизненных циклов систем. Те или иные стадии жизненного цикла могут отсутствовать или присутствовать в зависимости от каждого конкретного случая разработки системы.

В стандарте в качестве примера были приведены следующие стадии жизненного цикла:

1. Стадия замысла.

2. Стадия разработки.

3. Стадия производства.

4. Стадия применения.

5. Стадия поддержки применения.

6. Стадия прекращения применения и списания.

#### **5.5.1.4 Работы по адаптации стандарта ИСО/МЭК 12207**

* Определение условий выполнения проекта
* Запрос исходных данных для адаптации
* Выбор процессов, работ и задач
* Документирование решений по адаптации и их обоснование.

## **5.5.2 Стандарт ISO 15504[[7]](#footnote-7),[[8]](#footnote-8)**

Стандарт ISO 12207 (ссылка на текст - <http://docs.cntd.ru/document/1200076680> ) разрабатывался 9 лет и достаточно быстро устарел. В 1998г. выходит новый стандарт ISO/IEC TR 15504: Information Technology - Software Process Assessment (Оценка процессов разработки ПО). В этом документе рассматриваются вопросы аттестации, определения зрелости и усовершенствования процессов жизненного цикла ПО. Один из разделов документа содержит новую классификацию процессов жизненного цикла, являющуюся развитием стандарта ISO 12207.

Генетическая связь со стандартом ISO 12207 состоит в том, что все процессы стандарта ISO 15504 принадлежат к одной из следующих типов:

* базовый — процесс из 12207;
* расширенный — расширение процесса из 12207;
* новый — процесс, не описанный в 12207;
* составляющий — часть процесса из 12207;
* расширенный составляющий — расширенная часть проц. из 12207

**Классификация процессов в стандарте ISO15504.**

В соответствии с новой классификацией в трех группах процессов вводятся **пять категорий процессов**:

* Основные процессы:
  + CUS: Потребитель-поставщик
  + ENG: Инженерная
* Вспомогательные процессы:
  + SUP: Вспомогательная
* Организационные процессы:
  + MAN Управленческая
  + ORG: Организационная

Рассмотрим подробнее категории группы основных процессов

**Категория Потребитель-поставщик**

Категория состоит из процессов, непосредственно влияющих на потребителя, поддерживающих процесс разработки программного средства и его передачи потребителю и обеспечивающих возможность корректного использования программного средства или услуги.

Включает следующие процессы:

* CUS.1 Процесс приобретения (Acquisition process)
  + CUS.1.1 Процесс подготовки приобретения (Acquisition preparation process)
  + CUS.1.2 Процесс выбора поставщика (Supplier selection process)
  + CUS.1.3 Процесс мониторинга поставщика (Supplier Monitoring process)
  + CUS.1.4 Процесс приемки (Customer Acceptance process)
* CUS.2 Поставки (Supply process)
* CUS.3 Процесс выявления требований (Requirements process)
* CUS.4 Эксплуатации (Operationprocess)
  + CUS.4.1 Процесс эксплуатационного использования (Operational use process)
  + CUS.4.2 Процесс поддержки потребителя(Customer support process)

**Категория Инженерные процессы**

Категория состоит из процессов, которые непосредственно определяют, реализуют или поддерживают программный продукт, его взаимодействие с системой и документацию на него. В тех случаях, когда система целиком состоит из программных средств, инженерные процессы имеют отношение только к созданию и поддержанию этих программных средств.

Включает следующие процессы:

* ENG.1 Процесс разработки (Development process)
  + ENG.1.1 Процесс анализа требований и разработки системы (Systemrequirements analysis and design process)
  + ENG.1.2 Процесс анализа требований к программным средствам (Software requirements analysis process)
  + ENG.1.3 Процесс проектирования программных средств (Software design process)
  + ENG.1.4 Процесс конструирования программных средств (Software construction process)
  + ENG.1.5 Процесс интеграции программных средств (Software integration process)
  + ENG.1.6 Процесс тестирования программных средств (Software testing process)
  + ENG.1.7 Процесс интеграции и тестирования системы (System integration andtesting process)
* ENG.2 Процесс сопровождения системы и программных средств (System and software maintenance process)

### **5.5.3 Влияние развития информационных технологий на появление новых ИТ-стандартов**

Развитие информационных технологий и необходимость межгосударственного взаимодействия с их использование обуславливает разработку силами информационного сообщества всё новых стандартов, учитывающих это развитие.

В частности, это относится к появлению стандартов в области облачных технологий. О том, что это такое можно прочитать на сайтах:

* <https://seo-zona.ru/chajniku-pro-oblachnye-texnologii-2017-04-07.html>
* <https://zen.yandex.ru/media/mcs/chto-takoe-oblachnye-tehnologii-i-pochemu-ih-ispolzuiut-deviat-kompanii-iz-desiati-5eb04d1d49d6c31325adae7c>

С информацией о разработке и использовании облачных стандартов можно ознакомиться в следующих источниках:

* <http://www.alldc.ru/documentation/document/2003.html>
* <http://protect.gost.ru/v.aspx?control=8&baseC=-1&page=0&month=-1&year=-1&search=&RegNum=1&DocOnPageCount=15&id=197607>
* <http://www.cnews.ru/articles/novye_standarty_sozdayut_nauchnuyu_bazu>
* <https://www.ibm.com/developerworks/ru/library/cl-tools-to-ensure-cloud-application-interoperability/>
* <https://habr.com/ru/company/cloud4y/blog/352358/>
* <https://www.osp.ru/lan/2016/04/13049079>

1. **Методология** (от греч. μεθοδολογία — учение о способах; от [др.-греч.](https://ru.wikipedia.org/wiki/Древнегреческий_язык) μέθοδος из [μετά-](https://ru.wikipedia.org/wiki/Мета-" \t "Мета-) + ὁδός, букв. «путь вслед за чем-либо» и [др.-греч.](https://ru.wikipedia.org/wiki/Древнегреческий_язык) λόγος — [мысль, причина](https://ru.wikipedia.org/wiki/Логос)) — учение о [методах](https://ru.wikipedia.org/wiki/Метод), способах и **стратегиях** исследования предмета. [↑](#footnote-ref-1)
2. Программная система – это система, состоящая из ПО и, возможно, компьютерного оборудования для его выполнения. Источник: ГОСТ Р 51904 2002: Программное обеспечение встроенных систем. Общие требования к разработке и документированию. [↑](#footnote-ref-2)
3. В данном случае методологии разработки ПО — это инструмент, с помощью которого создание программного продукта превращается в упорядоченный процесс, а работа программиста становится более прогнозируемой и эффективной. [↑](#footnote-ref-3)
4. В скобках приведены используемые в различных источниках альтернативные термины [↑](#footnote-ref-4)
5. <http://www.unn.ru/pages/issues/aids/2007/16.pdf>

   <http://studopedia.ru/3_20558_standarti-reglamentiruyushchie-zhiznenniy-tsikl-informatsionnih-sistem.html>

   [↑](#footnote-ref-5)
6. <http://www.intuit.ru/studies/courses/64/64/lecture/1868?page=2> [↑](#footnote-ref-6)
7. <http://www.studfiles.ru/preview/6308934/page:5/> [↑](#footnote-ref-7)
8. <http://www.intuit.ru/studies/courses/64/64/lecture/1868?page=2> [↑](#footnote-ref-8)