GeekBrains

Архитектор Информационных систем

Подготовка архитектурного описания и проектной документации

На этом уроке

1. Студенты овладеют знаниями и навыками в создании проектной документации в соответствии с принятыми стандартами индустрии.
2. Будет дано представление о видах документации, стандартах, инструментах, жизненном цикле документов создаваемых и сопровождаемых архитектором при разработке информационных систем.
3. Будет дано представление об общепринятых рекомендациях и принципах при проектировании и разработки ПО.

Оглавление

Документация на программное обеспечение/программный продукт/информационную систему — совокупность материалов (артефактов), сопровождающих разработку на всём жизненном цикле, к которым относятся документы управления и планирования, проектно-конструкторская документация, схемы, листинги программного кода, программы и методики испытаний, руководства пользователя, диалоговая (оперативная) документация и справочный текст, описывающие, как пользоваться программным продуктом.

Документ (артефакт) — элемент документации: целевая информация, предназначенная для конкретной аудитории, размещенная на конкретном носителе (например, в книге, на диске, в краткой справочной карте) в заданном формате.

Программная документация — документы, содержащие в зависимости от назначения данные, необходимые для разработки, производства, эксплуатации, сопровождения программы или программного средства.

Типы документации

Существуют следующие основные типы документации на жизненном цикле программных продуктов:

- **Проектная/управленческая** – для планирования, управления производством, выпуском, экономикой проекта по разработке программных продуктов (информационной системы ИС) и принятия управленческих решений, может включать в себя исходные данные и технические задания;

- **Архитектурная/конструкторская** — для описание технических, технологических, композиционных решений, включающая описание рабочей среды, инфраструктуры, зависимостей, методов тестирования, испытаний и приёмки, а также других принципов, которые должны быть использованы при создании программного обеспечения или программных продуктов (ИС);

- **Техническая/технологическая** — подробная документация на принятые архитектурные решения, алгоритмы, интерфейсы, API, программный код,

- **Пользовательская** — руководства для конечных пользователей, администраторов и другого персонала сопровождающего развитие программного продукта (ИС);

- **Маркетинговая** — материалы для обеспечения коммерческой деятельности по продвижению и продажам программных продуктов (ИС).

1. Документирование архитектуры программного обеспечения

1.1. Общие сведения о процессе документирования

В небольших коллективах разработчиков процессу документирования архитектуры не уделяется должного внимания. Обычно детали архитектуры передаются в устной форме. Для обсуждения архитектур используются маркерные доски. Проблема документирования архитектур программных систем возникает с необходимостью объяснять архитектуру все большей и большей аудитории. При документировании архитектуры часто требуют ответа следующие вопросы:

- Что именно нужно документировать?
- Каким образом создавать документацию? Какой шаблон документа использовать для описания архитектуры?
- Какие нотации использовать?
- Насколько детальным должно быть описание?

Цели документирования:

1) Образовательная. С помощью документации участники проекта имеют возможность ознакомиться с продуктом. Часто документацией пользуются новые участники проекта: архитекторы, аналитики, разработчики и др.
2) Средство коммуникации между заинтересованными сторонами. Участники проекта общаются между собой используя элементы документации. Специалисты по сопровождению определяют стоимость внесения изменений, тестировщики и интеграторы определяют взаимосвязанность компонентов системы.
3) Основа системного анализа. В нем документация выступает в роли входных данных.

1.2. Этапы создания документации

1. Определение потребностей заинтересованных сторон. Если заинтересованные стороны и их потребности не выявлены, то возможно создание документации, которая будет бесполезна для всех участников проекта.
2. Сбор и документирование архитектурной информации в виде группы представлений и специального блока с общей информацией для всех представлений.

3. Проверка того, что созданная документация удовлетворяет требованиям заинтересованных сторон.

4. Подготовка архитектурной документации в вид, пригодный для той или иной заинтересованной стороны.

1.3. Правила документирования программной архитектуры:

1. Документация должна создаваться с точки зрения ее читателя.
2. Отсутствие повторений.
3. Отсутствие двусмысленности.
4. Объяснение нотаций, сокращений и прочих обозначений.
5. Следование стандартам организации и оформления.
6. Поддержка актуальности. В процессе проектирования часто вносятся изменения в документацию.

Созданная документация может быть прямым указанием для кодировщиков, спецификаций для автоматической кодогенерации, входными данными для планирования проекта.

1.4. Функции программной документации

Для эффективного управления документированием программного обеспечения важно осознавать различные функции, выполняемые документацией.

Программную документацию можно рассматривать как имеющую шесть основных функций:

1) информация для управления;
2) связь между задачами;
3) обеспечение качества;
4) инструкции и справки;
5) сопровождение программного обеспечения;
6) исторические справки.

1) Информация для управления

Во время разработки программного обеспечения администрации необходимо оценивать ход работы, возникающие проблемы и вероятности развития процесса. Периодические отчеты, согласно которым проверяют ход работ по графику и представляют планы на следующий период, обеспечивают контрольные механизмы и обзор проекта.

2) Связь между задачами

Большинство проектов разработки программного обеспечения разделяется на задачи, зачастую выполняемые различными группами:

- **специалисты в предметной области** начинают проект;
- **аналитики** формируют требования к системе;
- **архитекторы** разрабатывают системный и программный проекты;
- **сопровождающие программисты** улучшают эксплуатируемое программное обеспечение и разрабатывают его изменения или расширения.
- **специалисты по документации** создают пользовательскую документацию в соответствии со стратегией и стандартами по документированию;
- **специалисты по обеспечению качества** оценивают общую полноту и качество функционирования программного обеспечения;

Этим людям необходимы средства общения друг с другом, обеспечивающие информацию, которую можно, при необходимости, воспроизводить, распространять и на которую можно ссылаться.

Большинство методологий разработки программного обеспечения устанавливают официальные документы для связи между задачами. Например, аналитики представляют официальные спецификации требований для проектировщиков, а проектировщики выдают официальные проектные спецификации для программистов.

3) Обеспечение качества

Требуется документация разработки и документация продукции для выполнения задач, связанных с обязанностями по обеспечению качества программного обеспечения.

4) Инструкции и справки

Документация, требующаяся операторам, пользователям, руководителям и другим заинтересованным лицам для того, чтобы понимать и использовать программную продукцию.

5) Сопровождение программного обеспечения

Сопровождающим программистам требуется детальное описание программного обеспечения, такое, чтобы они могли локализовать и корректировать ошибки и модернизировать или изменять программное обеспечение соответствующим образом.

6) Исторические справки

Документация, требуемая в качестве исторической справки по проекту. Данная документация может помочь в переносе и переводе программного обеспечения в новое окружение.

1.5. Установление стратегии документирования

Стратегия должна поддерживать основные элементы эффективного документирования:

- требования документации охватывают весь жизненный цикл программного обеспечения.
- документирование должно быть управляемым.
- документация должна соответствовать ее читательской аудитории.

- работы по документированию должны быть объединены в общий процесс разработки программного обеспечения.
- должны быть определены и использованы стандарты по документированию.
- должны быть определены средства поддержки.

1.6. Внутри организации должны быть приняты стандарты и руководства для

- модели жизненного цикла программного обеспечения;
- типов и взаимосвязей документов;
- содержания документа;
- качества документа;
- форматов документа;
- обозначения документа.

1.7. Программные документы можно представить разделенными на три категории

1) документация разработки;

2) документация продукции;

3) документация управления проектом.

1.8. Документация разработки

Документы, описывающие процесс разработки программного обеспечения, определяют требования, которым должно удовлетворять программное обеспечение, определяют проект программного обеспечения; определяют, как его контролируют и как обеспечивают его качество. Документация разработки также включает в себя подробное техническое описание программного обеспечения (программную логику, программные взаимосвязи, форматы и хранение данных и т.д.).

Разработка документов преследует пять целей:

1) они являются **средством связи** между всеми вовлеченными в процесс разработки. Они описывают подробности решений, принятых относительно требований к программному обеспечению, проекту, программированию и тестированию;

2) они описывают **обязанности** группы разработки. Они определяют, кто, что и когда делает, учитывая роль программного обеспечения, предмета работ,

документации, персонала, обеспечивающего качество, и каждого вовлеченного в процесс разработки;

3) они выступают как **контрольные пункты**, которые позволяют руководителям оценивать ход разработки. Если документы разработки отсутствуют, неполны или устарели, руководители теряют важное средство для отслеживания и контроля проекта программного обеспечения;

4) они образуют основу **документации сопровождения программного обеспечения**, требуемой сопровождающими программистами как часть документации продукции;

5) они описывают **историю** разработки программного обеспечения.

Типовыми документами разработки являются:

- анализы осуществимости и исходные заявки;
- спецификации требований;
- спецификации функций;
- проектные спецификации, включая спецификации программ и данных;
- планы разработки;
- планы сборки и тестирования программного обеспечения;
- планы обеспечения качества, стандарты и графики;
- защитная и тестовая информация.

1.9. Выбор модели жизненного цикла программного обеспечения

Существует ряд моделей жизненного цикла программного обеспечения с отличающейся терминологией для различных стадий. С точки зрения программной документации, не имеет значения, какая модель выбрана, до тех пор, пока стадии и соответствующая им документация четко определены, спланированы и выполняемы для любого конкретного программного проекта. Руководители должны, поэтому, выбрать соответствующую модель жизненного цикла программного обеспечения и гарантировать, чтобы ее применяли в данной организации.

Руководители должны убедиться, что выбранные стадии и соответствующие задачи помогут им в контроле за ходом любого программного проекта. Создание документации, связанной с конкретной стадией, может, например, быть использовано как контрольный пункт для проверки, приемки и завершения стадии до начала следующей.

Должны быть установлены процедуры для применяемых в организациях стратегий документирования.

Процедуры определяют последовательность документирования:

- планирование;

- подготовка;
- конфигурационное управление;
- проверка;
- утверждение;
- производство;
- хранение;
- дублирование;
- распространение и модернизация;
- продажа.

2. Стандарты жизненного цикла ПО

Информационные системы должны удовлетворять интересам бизнеса, а также быть легко модифицируемыми и недорогими. Плохо спроектированная система, в конечном счете, требует больших затрат и времени для ее содержания и обновления.

Одним из базовых понятий методологии проектирования ИС является понятие жизненного цикла ее программного обеспечения (ЖЦ ПО).

Жизненный цикл ПО – это непрерывный процесс, который начинается с момента принятия решения о необходимости его создания и заканчивается в момент его полного изъятия из эксплуатации.

Существует целый ряд стандартов, регламентирующих ЖЦ ПО, а в некоторых случаях и процессы разработки.

Среди наиболее известных стандартов можно выделить следующие:

ISO/IEC 12207: 2010 — стандарт на процессы и организацию жизненного цикла. Распространяется на все виды заказного ПО.

ГОСТ 34.601-90 — распространяется на автоматизированные системы и устанавливает стадии и этапы их создания. Кроме того, в стандарте содержится описание содержания работ на каждом этапе. Стадии и этапы работы, закрепленные в стандарте, в большей степени соответствуют каскадной модели жизненного цикла.

Custom Development Method (методика Oracle) — по разработке прикладных информационных систем - технологический материал, детализированный до уровня заготовок проектных документов, рассчитанных на использование в проектах с применением Oracle. Применяется CDM для классической модели ЖЦ (предусмотрены все работы/задачи и этапы), а также для технологий "быстрой разработки" (Fast Track) или "облегченного подхода", рекомендуемых в случае малых проектов.

Rational Unified Process (RUP) — предлагает итеративную модель разработки, включающую четыре фазы: начало, исследование, построение и внедрение. Каждая фаза может быть разбита на этапы (итерации), в результате которых выпускается версия для внутреннего или внешнего использования. Прохождение через четыре основные фазы называется циклом разработки, каждый цикл завершается генерацией версии системы. Если после этого работа над проектом не прекращается, то полученный продукт продолжает развиваться и снова минует те же

фазы. Суть работы в рамках RUP - это создание и сопровождение моделей на базе UML.

Microsoft Solution Framework (MSF) — сходна с RUP, так же включает четыре фазы: анализ, проектирование, разработка, стабилизация, является итерационной, предполагает использование объектно-ориентированного моделирования. MSF в сравнении с RUP в большей степени ориентирована на разработку бизнес-приложений.

Extreme Programming (XP) — Экстремальное программирование сформировалось в 1996 году. В основе методологии командная работа, эффективная коммуникация между заказчиком и исполнителем в течение всего проекта по разработке ИС, а разработка ведется с использованием последовательно дорабатываемых прототипов.

Agile — диалоговый принцип разработки, при котором программы разрабатывают поэтапно, с начальной ступени проекта. Это ее основное отличие от каскадных принципов, где код становится доступным только по окончании цикла работы.

Scrum — это фреймворк гибкой разработки ПО, который считается методологией «по умолчанию». Для многих является синонимом Agile. Процесс управления состоит из фиксированных коротких итераций — спринтов (sprints).Используя методологию Scrum, представитель заказчика плотно работает с командой разработки, расставляя приоритеты в списке требований к продукту (Product Backlog). Этот список состоит из баг-фиксов, функциональных и нефункциональных требований — из всего, что нужно сделать для реализации рабочего приложения.

2.1. Процессы жизненного цикла программной системы

Процессы жизненного цикла программной системы базируются на принципах модульности (максимальная связанность функций процесса и минимальная связь между процессами) и ответственности (процесс ассоциируется с ответственностью за его реализацию). Функции, выполняемые этими процессами, определяются специфическими целями, требуемыми результатами и наборами деятельностей, которые составляют этот процесс.

Процессы жизненного цикла программной системы объединяются в четыре группы, согласно таблице 1.

Таблица 1. Процессы жизненного цикла программной системы

Группа	Наименование процесса
Процессы договора	Процесс приобретения

	Процесс поставки
Процессы предприятия	Процесс управления средой предприятия
	Процесс управления инвестициями
	Процесс управления процессами жизненного цикла программной системы
	Процесс управления ресурсами
	Процесс управления качеством
Процессы проекта	Процесс планирования проекта
	Процесс оценки проекта
	Процесс контроля проекта
	Процесс принятия решений
	Процесс управления рисками
	Процесс управления конфигурацией
	Процесс управления информацией
Технические процессы	Процесс концептуального моделирования
	Процесс прикладного моделирования
	Процесс определения требований заинтересованного лица
	Процесс анализа требований
	Процесс структурного проектирования
	Процесс воплощения
	Процесс интеграции
	Процесс проверки
	Процесс утверждения
	Процесс перехода
	Процесс эксплуатации
	Процесс сопровождения
	Процесс изъятия

2.2. Модель жизненного цикла ПО согласно Стандарту ISO/IEC 12207

Стандарт ISO/IEC 12207 не предлагает конкретную модель ЖЦ и методы разработки ПО (под моделью ЖЦ понимается структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач, выполняемых на протяжении ЖЦ. Модель ЖЦ зависит от специфики ИС и условий, в которых последняя создается и функционирует). Его регламенты являются общими для любых моделей ЖЦ, методологий и технологий разработки. Стандарт ISO/IEC 12207 описывает структуру процессов ЖЦ ПО, но не конкретизирует в деталях, как реализовать или выполнить действия и задачи, включенные в эти процессы.

В изначально существовавших однородных ИС каждое приложение представляло собой единое целое. Для разработки такого типа приложений применялся каскадный способ. Его основной характеристикой является разбиение всей разработки на этапы, причем переход с одного этапа на следующий происходит только после того, как будет полностью завершена работа на текущем. каждый этап

завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков.

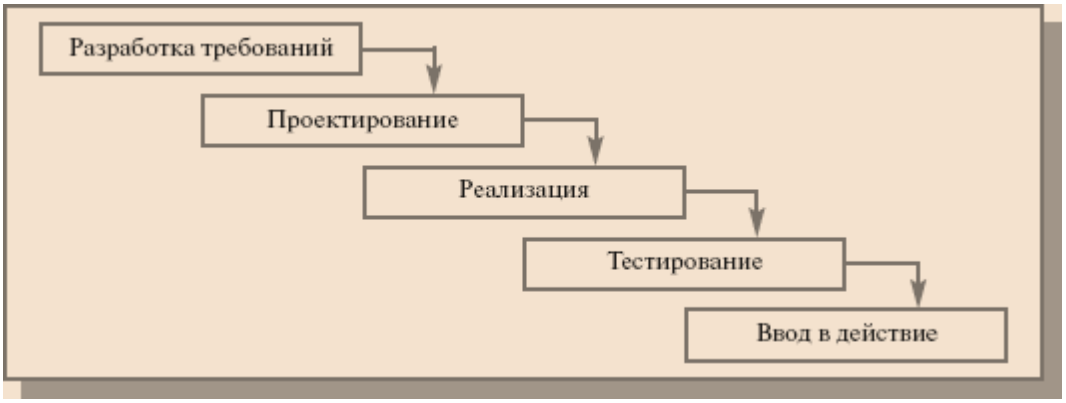


Рисунок 1. Каскадная схема разработки ПО

Положительные стороны применения каскадного подхода заключаются в следующем:

- на каждом этапе формируется законченный набор проектной документации, отвечающий критериям полноты и согласованности;
- выполняемые в логичной последовательности этапы работ позволяют планировать сроки завершения всех работ и соответствующие затраты.

Каскадный подход хорошо зарекомендовал себя при построении ИС, для которых в самом начале разработки можно достаточно точно и полно сформулировать все требования, с тем, чтобы предоставить разработчикам свободу реализовать их как можно лучше с технической точки зрения. В эту категорию попадают сложные расчетные системы, системы реального времени и другие подобные задачи. Однако в процессе использования этого подхода обнаружился ряд его недостатков, вызванных прежде всего тем, что реальный процесс создания ПО никогда полностью не укладывался в такую жесткую схему. В процессе создания ПО постоянно возникала потребность в возврате к предыдущим этапам и уточнении или пересмотре ранее принятых решений. В результате реальный процесс создания ПО принимал вид, представленный на рис. 2.

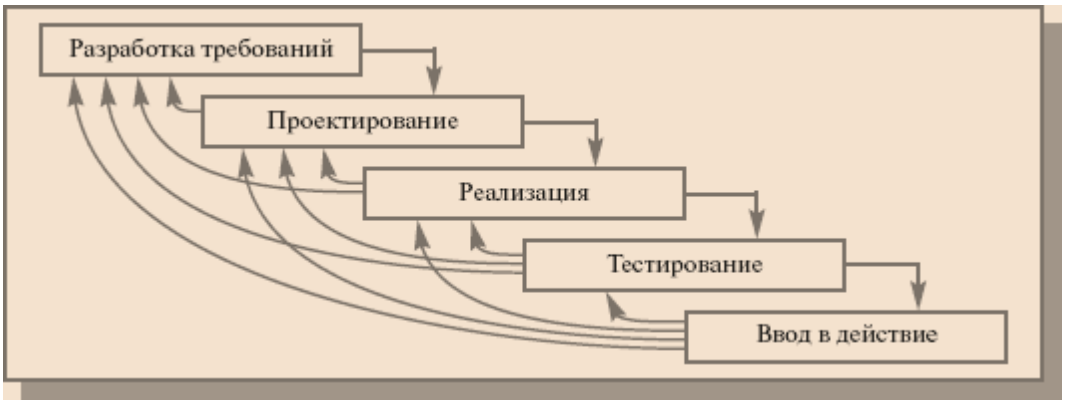


Рисунок 2. Реальный процесс разработки ПО по каскадной схеме

2.2.1. Основные процессы жизненного цикла ISO/IEC 12207

В соответствии с базовым международным стандартом ISO/IEC 12207 все процессы ЖЦ ПО делятся на три группы:

1. Основные процессы:
   - приобретение;
   - поставка;
   - разработка;
   - эксплуатация;
   - сопровождение.

2. Вспомогательные процессы:
   - документирование;
   - управление конфигурацией;
   - обеспечение качества;
   - разрешение проблем;
   - аудит;
   - аттестация;
   - совместная оценка;
   - верификация.

3. Организационные процессы:
   - создание инфраструктуры;
   - управление;
   - обучение;
   - усовершенствование.

2.2.2. Категории процессов жизненного цикла

Стандарт ISO/IEC 12207 группирует различные виды деятельности, которые могут выполняться в течение жизненного цикла программных систем, в семь групп процессов. Каждый из процессов жизненного цикла в пределах этих групп описывается в терминах цели и желаемых выходов, списков действий и задач,

которые необходимо выполнять для достижения этих результатов.

1) процессы соглашения - два процесса
2) процессы организационного обеспечения проекта - пять процессов;
3) процессы проекта - семь процессов;
4) технические процессы - одиннадцать процессов;
5) процессы реализации программных средств - семь процессов;
6) процессы поддержки программных средств - восемь процессов;
7) процессы повторного применения программных средств - три процесса.

Цели и результаты процессов жизненного цикла образуют эталонную модель процессов.

Группы процессов жизненного цикла представлены на рисунке 3.

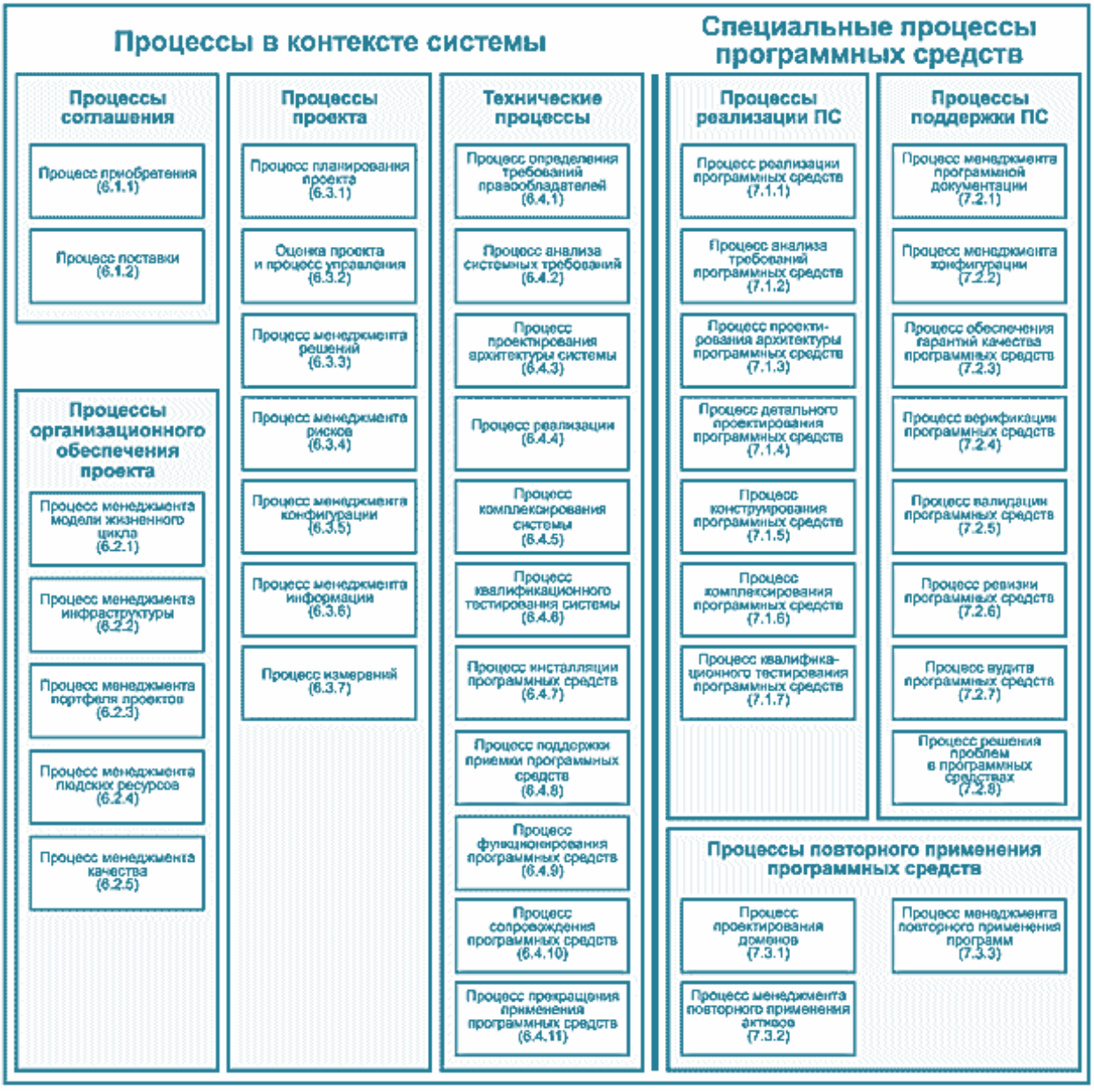


Рисунок 3. Группы процессов жизненного цикла

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.

2.2.3. Краткое содержание процессов жизненного цикла

●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.

●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.

●Процессы проекта

В 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. Обе категории процессов проекта описаны ниже.

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:

a) процесс планирования проекта;

b) процесс управления и оценки проекта.

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.

●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.

Технические процессы 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.

Технические процессы состоят из следующих процессов:

a) определение требований правообладателей;
b) анализ системных требований;
c) проектирование архитектуры системы;
d) процесс реализации;
e) процесс комплексирования системы;
f) процесс квалификационного тестирования системы;
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.

●Процессы реализации программных средств

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:

a) процесс анализа требований к программным средствам;
b) процесс проектирования архитектуры программных средств;
c) процесс детального проектирования программных средств;
d) процесс конструирования программных средств;
e) процесс комплексирования программных средств;
f) процесс квалификационного тестирования программных средств.

●Процессы поддержки программных средств

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.

●Процессы повторного применения программных средств

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.

●Процесс проектирования доменов

Цель процесса проектирования доменов заключается в разработке и сопровождении моделей доменов, архитектуры доменов и активов для доменов.

В результате успешного осуществления процесса проектирования доменов:
a) выбираются формы представления модели и архитектуры домена;
b) определяются границы домена и его взаимосвязи с другими доменами;
c) разрабатывается модель домена, которая объединяет в себе существенные общие и различные свойства, возможности, концепции и функции в этом домене;
d) разрабатывается архитектура домена, описывающая семейство систем в пределах домена, включая их общность и изменчивость;
e) специфицируются активы, относящиеся к домену;
f) соответствующие активы приобретаются или разрабатываются и поддерживаются в течение всего жизненного цикла;
g) модели и архитектуры домена поддерживаются в течении всего их жизненного цикла.

Примечание 1 - Проектирование доменов основано на повторном применении подхода к определению области применения (то есть определению домена), спецификации структуры (то есть архитектуры домена) и созданию активов (например, требований, конструкции, программного кода, документации) для класса систем, подсистем или приложений.

Примечание 2 - Процесс, относящийся к проектированию доменов, может перекрываться с процессами разработки и сопровождения, использующими активы, созданные процессом проектирования доменов.

●Анализ доменов

a) Разработчик доменов должен определять границы каждого домена и взаимосвязи между конкретным доменом и другими доменами.

b) Разработчик доменов должен идентифицировать текущие и предполагаемые потребности правообладателей программных продуктов в пределах этого домена.

c) Разработчик доменов должен создавать модели домена, используя формы представления, выбранные в действиях процесса реализации данного процесса.

d) Разработчик доменов должен составлять словарь, охватывающий терминологию для описания важных понятий доменов и взаимоотношений между сходными или общими активами домена.

e) Разработчик доменов должен классифицировать и документировать модели домена.

f) Разработчик доменов должен оценивать модели и словарь домена в соответствии с условиями выбранной техники моделирования и процедурами приемки и сертификации активов организации.

g) Разработчик доменов должен проводить анализ ревизий домена. Разработчики программных средств, менеджеры активов, эксперты домена и пользователи должны принимать участие в ревизиях.

h) Разработчик доменов должен представлять модели домена менеджеру активов.

●Проектирование доменов

a) Разработчик доменов должен создавать и документально оформлять архитектуру домена, согласовывать ее с моделью домена и следовать стандартам организации.

b) Архитектура домена должна оцениваться в соответствии с условиями выбранной техники проектирования архитектуры и процедурами приемки и сертификации активов организации.

c) Для каждого выбранного объекта, предназначенного для повторного применения, разработчик доменов должен разрабатывать и документально оформлять спецификацию активов.

d) Для каждого определенного актива спецификация должна оцениваться в соответствии с процедурами приемки и сертификации активов организации.

e) Разработчик доменов должен проводить ревизии проекта домена. Разработчики программных средств, эксперты домена и менеджеры активов должны участвовать в проведении этих ревизий.

f) Разработчик доменов должен предоставлять архитектуру домена менеджеру активов.

●Сопровождение доменов

a) соответствие с моделями и архитектурой домена;

b) воздействия на системы и программные продукты, которые используют активы;

c) воздействия на будущих пользователей активов;
d) воздействия на возможность повторного использования активов.

2.3. Стандарт ЖЦ согласно ГОСТ 34 – Разработка автоматизированной системы управления (АСУ)

Рассмотрим в отдельности каждую стадию и перечень документов согласно ГОСТ 34, который должен фиксировать результаты проведенных работ.

Жизненный цикл процесса создания АСУ согласно ГОСТ 34 (ГОСТ 34.601-90) включает следующие стадии:

a) Формирование требований к АС
b) Разработка концепции АС
c) Техническое задание
d) Эскизный проект
e) Технический проект
f) Рабочая документация
g) Ввод в действие
h) Сопровождение АС

2.3.1. Формирование требований к АС

На начальном этапе создания АС согласно требованиям ГОСТ 34 необходимо проведение обследования объекта автоматизации. В рамках обследования происходит сбор и анализ данных об организации, производственной структуре и функционировании объекта автоматизации. Источником для получения данных сведений могут послужить устав и регламенты организации, а также общегосударственные законы, постановления и другие нормативно-правовые акты.

Обследование также должно провести анализ автоматизированных систем, уже функционирующих в рамках объекта автоматизации. На данном этапе необходимо также определить степень интеграции создаваемой АС с существующими системами. Кроме того должен быть проведен сбор и анализ сведений о зарубежных и отечественных аналогах, создаваемой АС.

На базе полученных данных необходимо выявить основные функциональные и пользовательские требования к АС.

В результате проведенных исследований должен быть составлен аналитический отчет, который должен содержать следующую информацию:

a) Объект, цели исследования и методология проведения исследовательских работ
b) Основные конструктивные, технологические и технико-эксплуатационные характеристики
c) Основные требования пользователя к АС
d) Степень внедрения и рекомендации по внедрению АС
e) Область применения АС
f) Обоснование экономической эффективности создания АС
g) Прогнозы и предположения о развитии объекта исследования.

2.3.2. Разработка концепции АС

Исходя из результатов, проведенных исследований объекта автоматизации, согласно ГОСТ 34 разрабатывается несколько вариантов концепций АС, удовлетворяющих требованию пользователей. Концепции АС могут быть представлены заказчику в виде отчета о выполненных работах, или отдельного документа «Концепция АС», или стать частью аналитического отчета.

2.3.3. Техническое задание (ТЗ)

Ключевая роль при создании АС отводится именно разработке и согласованию технического задания, так как он должен определять требования и порядок разработки, развития и модернизации системы. В соответствии с данным документом должны будут проводиться работы по испытанию и приемке системы в эксплуатацию. Техническое задание может быть разработано как на систему в целом так и на ее части.

Стандартом для разработки данного документа является ГОСТ 34.602-89, регламентирующий содержание разделов и стиль изложения в ТЗ.

Согласно ГОСТ 34 техническое задание должно включать следующие разделы:

1) Общие сведения
2) Назначение и цели создания (развития) системы
3) Характеристика объектов автоматизации
4) Требования к системе
5) Состав и содержание работ по созданию системы
6) Порядок контроля и приемки системы
7) Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие
8) Требования к документированию
9) Источники разработки. \

2.3.4. Эскизный и технический проект

На данных этапах происходит разработка проектных решений АС и создание технической документации:

a) Пояснительная записка к техническому (эскизному) проекту
b) Схема организационной структуры
c) Схема комплекса технических средств (КТС)
d) Схема функциональной структуры
e) Схема автоматизации
f) Перечень входных и выходных сигналов и данных
g) Описание автоматизированных функций

Полный перечень документации, разрабатываемый на данных этапах создания АС приводится в ГОСТ 34.201-89.

Зачастую создание полного пакета документов эскизного и технического проекта, представленного в стандартах ГОСТ 34 является нецелесообразным.

Поэтому минимальный комплект документации согласовывается с заказчиком и фиксируется в техническом задании на создание АС.

2.3.5. Рабочая документация

Данный этап подразумевает разработку рабочей документации на АС или ее части. Данный пакет документов также согласовывается с заказчиком в индивидуальном порядке и фиксируется в ТЗ. Зачастую пакет рабочей документации ограничивается следующими документами:

a) Руководство пользователя (администратора)
b) Инструкция по эксплуатации КТС
c) Общее описание системы (в случае присутствия документа «Пояснительная записка к техническому (эскизному) проекту» данный документ нецелесообразен так большинство разделов дублируются)
d) Программа и методика испытаний

2.3.6. Ввод в действие

Стадия ввода в действие АС согласно ГОСТ 34 включает подготовку комплекса технических средств, проведение пусконаладочных работ и обучение персонала.

Перед вводом АС в эксплуатацию производятся предварительные испытания, по результатам которых формируется «Протокол испытаний». Протокол фиксирует все замечания к системе, порядок и сроки их устранения, и подтверждает ее готовность к вводу в опытную эксплуатацию.

Во время проведения опытной эксплуатации персоналу рекомендуется вести журнал, где должны фиксироваться все ошибки, сбои и отказы системы.

По завершению опытной эксплуатации проводятся приемочные испытания, результаты которых также должны быть зафиксированы протоколом. По результатам приемочных испытаний принимается решение о передаче АС в промышленную эксплуатацию.

После полной передачи системы обе стороны подписывают «Акт выполненных работ».

2.3.7. Сопровождение АС

Этап сопровождения АС подразумевает выполнение работ по гарантийному и послегарантийному обслуживанию системы.

3. Архитектурное представление

Цели архитектуры

Основной целью архитектуры является определение требований, которые влияют на структуру приложения. Хорошо продуманная архитектура снижает бизнес-риски, связанные с созданием технического решения, и устанавливает мост между бизнес- и техническими требованиями.

Некоторые из других целей следующие:

- Раскрыть структуру системы, но скрыть детали ее реализации.
- Реализовать все варианты использования и сценарии.
- Попытаться учесть требования различных заинтересованных сторон.
- Учесть как функциональные, так и качественные требования.
- Снизить цель владения и улучшить положение организации на рынке.
- Улучшить качество и функциональность системы.
- Повысить внешнее доверие к организации или системе.
- Ограничения

Архитектура программного обеспечения все еще является развивающейся дисциплиной в рамках программной инженерии.

Она имеет следующие ограничения:

- Отсутствие инструментов и стандартизированных способов представления архитектуры;
- Отсутствие методов анализа, позволяющих предсказать, приведет ли архитектура к реализации, отвечающей требованиям;
- Отсутствие осознания важности архитектурного проектирования для разработки программного обеспечения;
- Отсутствие понимания роли архитектора программного обеспечения и плохая коммуникация между заинтересованными сторонами;
- Отсутствие понимания процесса проектирования, опыта проектирования и оценки проектирования.

Описание архитектуры может состоять из набора представлений, каждое из которых показывает взгляд на программное обеспечения с точки зрения какой-либо заинтересованной стороны, и отдельного описания общих элементов, присутствующих во всех представлениях.

Представление или точка зрения на архитектуру является основной единицей архитектурной документации программного обеспечения. Представление (architectural view) является описанием архитектуры, которое пригодно для чтения одной из заинтересованных сторон.

Архитектура программного обеспечения служит планом системы. Она обеспечивает абстракцию для управления сложностью системы и создания механизма связи и координации между компонентами.

Она определяет структурированное решение для удовлетворения всех технических и эксплуатационных требований, оптимизируя при этом общие атрибуты качества, такие как производительность и безопасность.

Кроме того, она включает в себя набор важных решений организации, связанных с разработкой программного обеспечения, и каждое из этих решений может оказать значительное влияние на качество, сопровождаемость, производительность и общий успех конечного продукта.

Эти решения включают в себя:

1) Выбор структурных элементов и их интерфейсов, из которых состоит система.
2) Поведение, определенное взаимодействием между этими элементами.
3) Объединение этих структурных и поведенческих элементов в большую подсистему.
4) Архитектурные решения согласуются с бизнес-целями.
5) Архитектурные стили направляют организацию.
6) Проектирование программного обеспечения.

Проектирование программного обеспечения обеспечивает план проектирования, который описывает элементы системы, их соответствие и совместную работу для выполнения требований системы.

Наличие плана проектирования преследует следующие цели:

- Согласование требований к системе и определение ожиданий с клиентами, маркетинговым и управленческим персоналом;
- Действует в качестве чертежа в процессе разработки;
- Руководство задачами по реализации, включая детальное проектирование, кодирование, интеграцию и тестирование;
- Он разрабатывается до детального проектирования, кодирования, интеграции и тестирования и после анализа домена, анализа требований и анализа рисков;

3.1. Механизмы для описания архитектуры:

Существует несколько общих механизмов, используемых для описания архитектуры. Эти механизмы облегчают повторное использование успешных стилей описания, чтобы их можно было применять ко многим системам:

- точки зрения на архитектуру;
- языки описания архитектуры;
- архитектурные фреймворки;

Описания архитектуры программного обеспечения обычно организуются в виде представлений, которые аналогичны различным типам чертежей, составляемых при построении архитектуры.

Каждое представление рассматривает набор системных проблем, следуя соглашениям своей точки зрения, где точка зрения - это спецификация, описывающая нотации, методы моделирования, которые будут использоваться в

представлении для выражения рассматриваемой архитектуры с точки зрения заданного набора заинтересованных сторон и их проблем (ISO/IEC 42010). Точка зрения определяет не только сформулированные проблемы (т.е. то, что должно быть рассмотрено), но и представление, используемые виды моделей, используемые соглашения и любые правила согласованности (соответствия) для обеспечения согласованности представления с другими представлениями.

3.2. Примеры точек зрения

1) Функциональная точка зрения
2) Логическая точка зрения
3) Точка зрения информации/данных
4) Точка зрения модуля
5) Точка зрения компонентов и соединителей
6) Точка зрения требований
7) Точка зрения разработчика/внедрения
8) Точка зрения Concurrency/process/runtime/thread/execution
9) Точка зрения производительности
10) Точка зрения безопасности
11) Физическая точка зрения/ точка зрения развертывания/установки
12) Точка зрения действия пользователя/обратной связи

3.3. Языки описания архитектуры

Язык описания архитектуры (ADL) - это любое средство выражения, используемое для описания архитектуры программного обеспечения (ISO/IEC/IEEE 42010).

С 1990-х годов было разработано множество специализированных ADL, включая AADL (стандарт SAE), Wright (разработанный Карнеги-Меллон), Acme (разработанный Карнеги-Меллон), xADL (разработанный UCI), Darwin (разработанный Имперским колледжем Лондона), DAOP-ADL (разработанный Университетом Малаги) и ByADL (Университет Аквилы, Италия).

Ранние ADL делали акцент на моделировании систем в терминах их компонентов, разъемов и конфигураций. Более современные ADL (такие как ArchiMate и SysML) имеют тенденцию быть языками "широкого спектра", способными выражать не только компоненты и соединители, но и различные проблемы с помощью множества подъязыков.

В дополнение к специализированным языкам, существующие языки, такие как UML, могут использоваться в качестве ADL "для анализа, проектирования и реализации систем на основе программного обеспечения, а также для моделирования деловых и подобных процессов".

3.4. Архитектурные каркасы - фреймворки

Архитектурные рамки отражают "соглашения, принципы и практику описания архитектур, установленные в конкретной области применения и/или сообществом заинтересованных сторон" (ISO/IEC/IEEE 42010). Структура обычно реализуется в

виде одной или нескольких точек зрения или ADL. К фреймворкам, представляющим интерес для архитектуры программного обеспечения, относятся:

1) 4+1
2) Views and Beyond
3) TOGAF - ArchiMate
4) ARIS

3.5. Архитектурная модель 4+1 в нотации UML

Идея представление архитектуры программного обеспечения в виде 5 представлений, каждое из которых отражает архитектуру программной системы с различных точек зрения, принадлежит Филиппу Крачтену – канадскому специалисту по разработке программного обеспечения.

Архитектурная модель 4+1, представленная на рисунке 4, состоит из следующих представлений архитектуры программной системы:

1. Логическое представление;
2. Представление разработчика;
3. Процессное представление;
4. Физическое представление;
5. Варианты использования (сценарии).

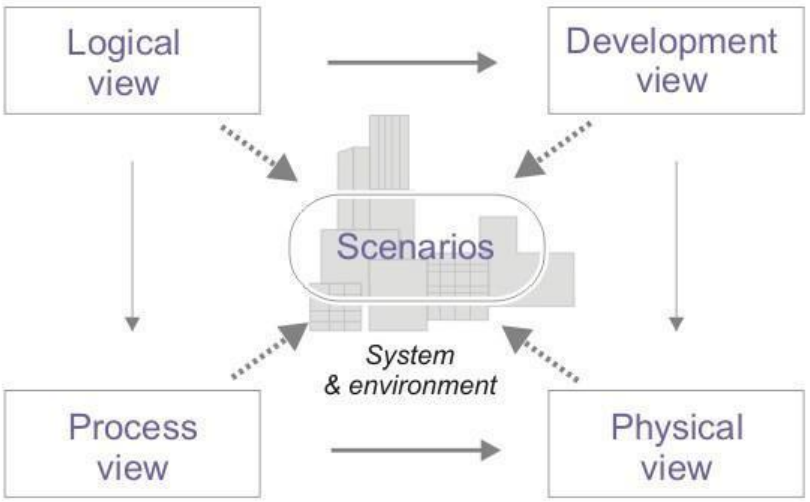


Рисунок 4 Архитектурная документация 4+1 Design Views

- **Logical view:** Логическое представление сфокусировано на функциональности, предоставляемой системой для конечных пользователей. В этом представлении используются UML-диаграммы классов, связей и последовательностей.

- **Development view:** Представление разработки показывает систему с точки зрения разработчика и касается управления программой. Это представление также известно как представление реализации. Здесь используются UML-диаграммы компонентов и пакетов для описания компонентов системы и их объединения в логические пакеты (например, слои).
- **Process view:** Представление процесса отражает динамические аспекты системы, показывает происходящие в системе процессы и связи между ними. Представление процесса фокусируется на поведении системы во время выполнения программы. Представление процесса отражает параллелизм выполнения, распределение, интеграцию, производительность, масштабирование и т.п. Представление процесса использует UML-диаграммы активности.
- **Physical view:** Представление физической структуры показывает систему с точки зрения системного инженера. Она показывает распределение программных компонентов по физическим уровням и физические каналы связи между уровнями. Это представление известно также как представление развёртывания системы. Представление физической структуры системы использует UML-диаграмму развёртывания.
- **Scenarios:** Описание архитектуры системы должно быть проиллюстрировано небольшим набором вариантов использования, или сценариев. Этот набор сценариев составляет пятое представление модели Крачтена. Сценарии должны описывать взаимодействие между объектами и между процессами. Сценарии используются для идентификации архитектурных элементов и для иллюстрирования и проверки качества архитектуры системы. Они также служат отправной точкой для тестирования архитектурного прототипа системы. Сценарии описываются UML-диаграммами вариантов использования.

Обычно сначала создается представление в виде вариантов использования (диаграммы Use Case Diagram), отражающее взгляд пользователя на программную систему.

Логическое представление описывает составные части программной системы, показывает внутреннюю структуру компонентов и взаимодействие составные частей программной системы. В логическом представлении используют диаграммы классов (Class Diagram), объектов (Object diagram), состояний (State Chart diagram), диаграмма потоков данных DFD (Data Flow Diagram). Диаграммы, используемые в методологиях группы IDEF.

Представление разработчика показывает как составные части программной системы организованы в модули компоненты. Используются UML Package Diagram и UML Component Diagram.

Физическое представление отображает с помощью UML Deployment Diagram как программная система развертывается на вычислительных устройствах.

Процессное представление служит для отображения процессов, происходящих в программной системе. Обычно для этих целей используют UML диаграммы деятельности, последовательностей (Sequence Diagram) и временная диаграмма (UML Timing Diagram).

В объектно-ориентированном подходе к проектированию и реализации программного обеспечения основной нотацией являются диаграммы UML.

Диаграммы UML подразделяют на два типа — это структурные диаграммы и диаграммы поведения.

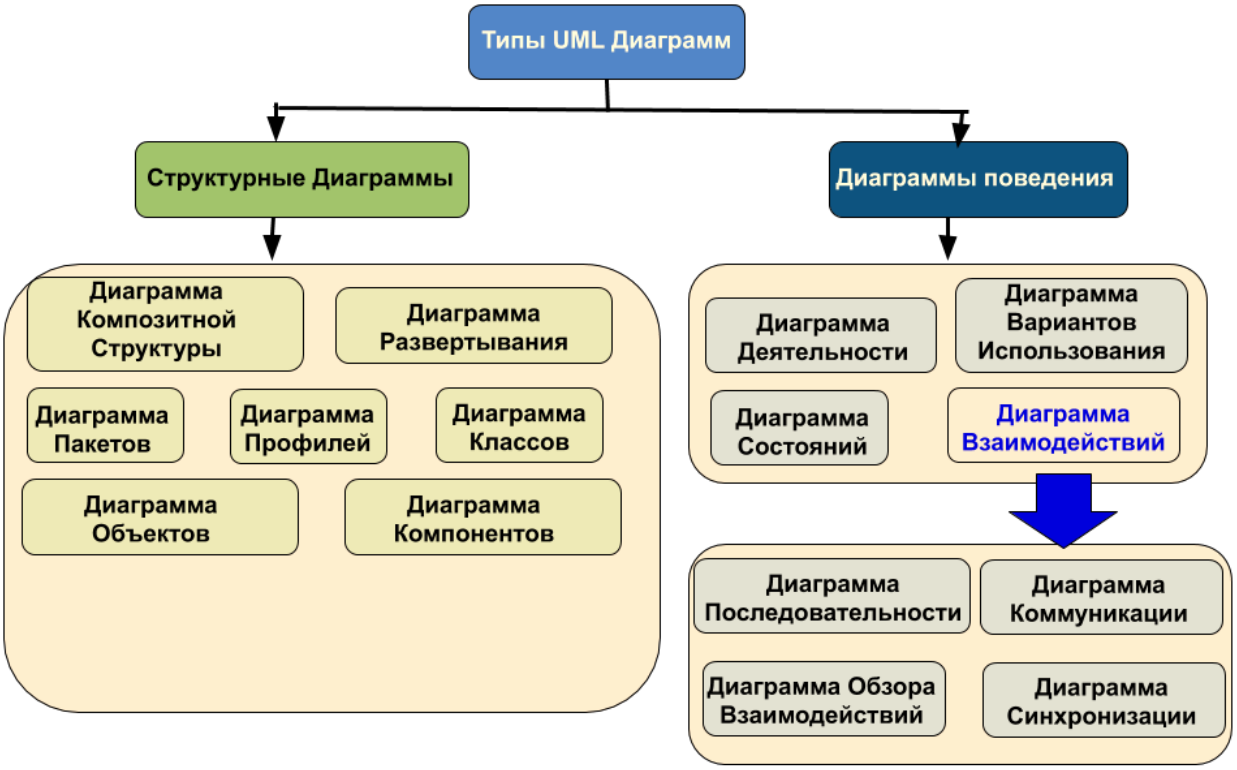


Рисунок 5. Декомпозиция UML диаграмм

Структурные диаграммы показывают статическую структуру системы и ее частей на разных уровнях абстракции и реализации, а также их взаимосвязь. Элементы в структурной диаграмме представляют значимые понятия системы и могут включать в себя абстрактные, реальные концепции и концепции реализации. Существует семь типов структурных диаграмм:

- Диаграмма составной структуры
- Диаграмма развертывания

- Диаграмма пакетов
- Диаграмма профилей
- Диаграмма классов
- Диаграмма объектов
- Диаграмма компонентов

Диаграммы поведения показывают динамическое поведение объектов в системе, которое можно описать, как серию изменений в системе с течением времени. А к диаграммам поведения относятся:

- Диаграмма деятельности
- Диаграмма прецедентов
- Диаграмма состояний
- Диаграмма последовательности
- Диаграмма коммуникаций
- Диаграмма обзора взаимодействия
- Временная диаграмма

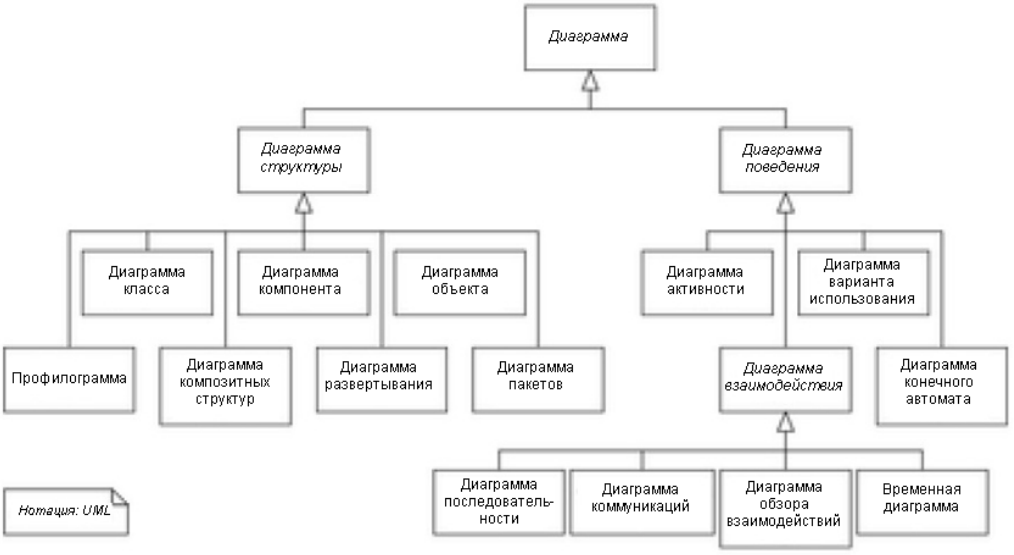


Рисунок 6. Пример декомпозиции видов представления архитектуры при объектно-ориентированном подходе

3.6. Подход Views and Beyond

Более усовершенствованным способом документирования программной архитектуры является подход Views and Beyond (V&B). Данный подход к документированию архитектуры был разработан Институтом программной инженерии Университета Карнеги - Меллон (США, Питсбург).

Подход Views and Beyond не только дает рекомендации по созданию документации, но и описывает распространенные архитектурные шаблоны.

В подходе Views and Beyond для архитектурного представления предлагается следующая структура:

1. Основная презентация (main presentation);

2. Каталог элементов и отношений между ними;

3. Контекстные диаграммы.

Основная презентация является отправной точкой документирования представления и показывает элементы представления и связи между ними. В основной презентации часто показывают наиболее существенные элементы и скрывают второстепенную информацию, такую как обработка ошибок, логирование и др.

Содержание главного представления зависит от используемой нотации. Часто в главной презентации используют графические нотации, но допустимым является использование таблиц, списков или иных способов представления текста. В случае необходимости главная презентация может содержать более одной диаграммы.

На контекстной диаграмме обычно показывается часть архитектуры, описанная в представлении, соотносящейся со всей программной системой и окружающей средой.

Подход V&B документирует наиболее важные архитектурные стили в руководстве по стилям. В руководстве по стилям V&B указаны задачи/проблемы проектирования, для решения которых подходит/не подходит данный стиль, обозначения, подходящие для отражения представлений в данном стиле, аналитические подходы, в которых используется данный стиль, и перечислены другие стили, с которыми он связан. Понимание архитектурных стилей, описанных в руководстве по стилю V&B, является предпосылкой для успешного применения подхода V&B.

Стоит подчеркнуть, что в одном архитектурном представлении могут сочетаться несколько архитектурных стилей. Например, в одном представлении модуля мы можем отразить зависимости использования между различными подмодулями (стиль Uses) вместе с функциональной декомпозицией системы на различные подмодули (стиль Decomposition).

Элементами стиля декомпозиции (Decomposition) являются модули. Определенные совокупности можно назвать подсистемами. Основное отношение, отношение декомпозиции, является специализированной формой отношения is-part-of и имеет в качестве основного

ограничения гарантию того, что элемент может быть частью не более чем одной совокупности.

Отношение декомпозиции может иметь свойство видимости, которое определяет, являются ли подмодули видимыми только для родительского модуля совокупности или также для других модулей. Считается, что модуль является видимым, если он может быть использован другими модулями. Благодаря этому свойству архитектор имеет некоторый контроль над видимостью модулей. Отношение декомпозиции, в котором ни один из содержащихся модулей не виден за пределами своего родителя, иногда называют отношением сдерживания. В отношении декомпозиции не допускаются циклы; то есть модуль не может содержать ни одного из своих предков. Ни один модуль в декомпозиционном представлении не может иметь более одного родителя.

Руководство по стилю V&B охватывает следующие стили архитектуры:

1) Модульные представления (Module views) - описывает как программная система представлена в виде значимых частей программного кода.
2) Стили модульной архитектуры:

- Стиль декомпозиции.
- Стиль использования.
- Стиль обобщения.
- Многослойный стиль.
- Стиль аспектов.

3) Стили компонентов и коннекторов:
   - Стили событий: стиль публикации и подписки, другие.
   - Стиль потока данных.
   - Стиль репозитория.

4) Стиль вызова и возврата:

   - Стиль SOA, клиент-сервер, одноранговый стиль.
   - Стили распределения:

5) Стиль развертывания.
6) Стиль распределения работы.
7) Компонентные представления (Component-and-connector views) - показывает программную систему с точки зрения взаимодействующих друг с другом элементов в процессе выполнения.
8) Представления развертывания и распределения (Allocation views) - показывает в программной системе используются аппаратные средства.

Архитектурный документ содержит набор представлений, распределённый по упомянутым выше группам, и документацию, общую для всех представлений.

Общая документация для всех представлений содержит:

- Введение в документ, позволяющее познакомить читателя с его целями и структурой и помочь быстрее найти необходимую информацию
- Общее описание программной системы и связей между представлениями
- Обоснование и ограничения архитектуры
- Рекомендации по сопровождению документации.

3.7. Представление TOGAF и ArchiMate

Разработан в Нидерландах в рамках исследовательского проекта, возглавляемого Telematica Instituut в сотрудничестве с рядом организаций и университетов.

Название языка составлено из двух частей: ArchiMate = Archi[tecture] + [Ani]mate. Первая часть слова – это часть слова Architecture (архитектура, конструкция, строение). Вторая часть слова – это часть слова Animate (оживить, вдохнуть жизнь, мультиплицировать, делать мультфильмы). В начале работы разработчики языка думали о том, что модели на языке будут «живыми», что-то наподобие мультипликации. В 2008 году право собственности и дальнейшего развития ArchiMate было передано одной из ведущих организаций по разработке открытых и независимых от поставщиков ИТ-стандартов -- консорциуму The Open Group, активно развивающего стандарт архитектуры предприятия TOGAF.

TOGAF и ArchiMate – объединенный подход к архитектуре

TOGAF и ArchiMate – это стандарты The Open Group, которые непосредственно относятся к разработке архитектуры предприятия. С одной стороны, у TOGAF и ArchiMate имеются свои спецификации, и они могут использоваться по отдельности, независимо друг от друга, или совместно с другими стандартами и фреймворками (Рисунок 6):

Рисунок 6. Независимое использование TOGAF и ArchiMate

С другой стороны, стандарты дополняют друг друга (Рисунок 7):

· TOGAF обеспечивает метод разработки архитектуры, в состав которого входят разделенный на фазы процесс разработки, руководства и техники;

· ArchiMate обеспечивает язык с необходимыми элементами, отношениями и графическими обозначениями для моделирования архитектур.

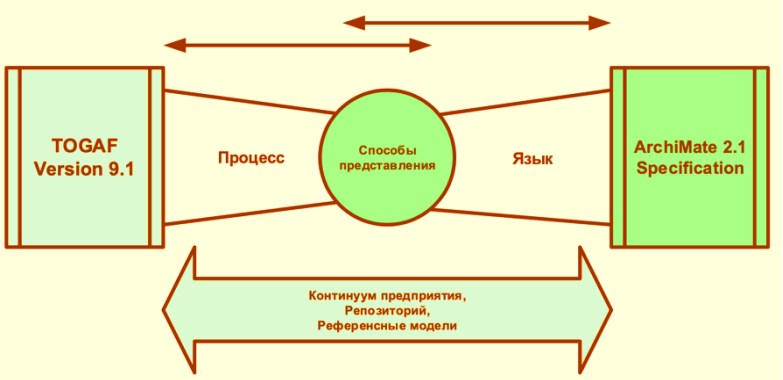


Рисунок 7 Компоненты объединенного подхода TOGAF и ArchiMate

Язык позволяет создавать как отдельные модели, соответствующие представлениям TOGAF, так и модели, объединяющие различные домены архитектуры. С включением двух расширений язык полностью покрывает все фазы метода разработки архитектуры TOGAF. Таким образом вместе эти два стандарта обеспечивают объединенный подход к архитектуре. В рамках консорциума The Open Group продолжаются работы по развитию ArchiMate, сближению его спецификации со спецификацией TOGAF. В частности, рассматривается разработка новых расширений языка, которые будут включать понятия для моделирования бизнес-политик и бизнес-правил, а также понятия для поддержки процессов принятия решений.

ArchiMate можно позиционировать как средство интегрированного высокоуровневого моделирования и анализа различных доменов предприятия и зависимостей между доменами (Рисунок 8):

Рисунок 8 Позиционирование языка ArchiMate

ArchiMate не фокусируется на деталях реализации. ArchiMate не заменяет UML, BPMN или ERD, а дополняет их. У него шире обзор, но и меньше возможностей по детализации, чем в этих языках.

ERD нотация - Entity-Relationship Diagram

ER-модель (англ. Entity-Relationship model, модель «сущность—связь») — модель данных, позволяющая описывать концептуальные схемы предметной области.

ER-модель используется при высокоуровневом (концептуальном) проектировании баз данных. С её помощью можно выделить ключевые сущности и обозначить связи, которые могут устанавливаться между этими сущностями.

Во время проектирования баз данных происходит преобразование схемы, созданной на основе ER-модели, в конкретную схему базы данных на основе выбранной модели данных (реляционной, объектной, сетевой или др.).

ER-модель представляет собой формальную конструкцию, которая сама по себе не предписывает никаких графических средств её визуализации. В качестве стандартной графической нотации, с помощью которой можно визуализировать ER-модель, была предложена диаграмма «сущность-связь» - Entity-Relationship diagram, ERD, ER-диаграмма.

Понятия «ER-модель» и «ER-диаграмма» часто не различают, хотя для визуализации ER-моделей могут быть использованы и другие графические нотации, либо визуализация может вообще не применяться (например, использоваться текстовое описание).

В ER-модели абстрактным объектам реальной действительности соответствует понятие «сущность».

Сущность – это абстрактный объект, который в конкретном контексте имеет независимое существование.

Различают понятия «тип сущности» и «экземпляр сущности».

Тип сущности (в дальнейшем просто сущность) представляет собой объект-тип, результат абстракции обобщения множества однородных объектов-экземпляров реальной действительности с одинаковыми свойствами.

Сущность имеет семантически значимое имя, как правило, имя существительное, например: «Студент», «Преподаватель», «Сотрудник», «Отдел», «Компьютер, «Книга» и т. д.

Экземпляр сущности соответствует объектам реальной действительности. Это конкретные персоны студентов, преподавателей, сотрудников, конкретные отделы на предприятии, конкретные компьютерные устройства, переплеты книг и т.д. Экземпляры сущностей уникальны, в природе не бывает двух одинаковых объектов. Следовательно, экземпляр сущности может быть идентифицирован уникальным образом.

Связь – это ассоциация сущностей или отношение между сущностями. Различают понятия «тип связи» и «экземпляр связи».

Тип связи (в дальнейшем просто связь) представляет собой отношение между типами сущностей.

Связь имеет семантически значимое имя, как правило, в форме глагола, например: «Преподаватель» «Ведет» «Дисциплину», «Отдел» «Включает» «Сотрудника» и т. д.

Экземпляр связи представляет собой отношение между экземплярами сущностей, например: Сидоров «Ведет» Историю , Отдел 123 «Включает» Кузнецова и т. д.

Связи обладают свойствами:

− вид (категориальная, идентифицирующая, неидентифицирующая);

− степень (унарная, бинарная, тернарная, N-арная); − кардинальность;

− внешний ключ сущности.

Атрибут – это свойство сущности или связи. Атрибутам как спецификаторам свойств сущностей или связей

присваиваются семантически значимые имена, как правило, в форме существительного.

Атрибутами сущности «Сотрудник» являются: табельный номер, фамилия, имя, отчество, дата рождения и другие персональные характеристики.

Атрибутами тернарной связи «Экзамен» между сущностями «Студент», «Преподаватель», «Дисциплина» являются идентификационные номера студента, преподавателя и дисциплины, а так же время и место проведения экзамена.

С понятием атрибута связаны понятия:

− домен атрибута;

− зависимости между атрибутами;

− потенциальный ключ сущности;

− детерминант и другие.

Сбалансированная система показателей – BSC

Сбалансированная система показателей (система сбалансированных показателей, ССП, англ. Balanced ScoreCard, BSC) — инструмент стратегического управления результативностью, частично стандартизированная форма отчётности, позволяющая менеджерам отслеживать исполнение заданий сотрудниками, а также последствия исполнения или неисполнения.

Другое определение: ССП — это система измерения эффективности деятельности всего предприятия (система стратегического планирования), основанная на видении и стратегии, которая отражает наиболее важные аспекты бизнеса. Система снабжена специальными методами проектирования и автоматизации.

Термин может соответствовать как системе в целом, так и индивидуальной реализации показателей. Важнейшими характеристиками подхода являются:

- акцент на стратегической повестке организации;
- отбор небольшого числа отслеживаемых данных;
- сочетание финансовых и нефинансовых данных.

Модель 5 Forces

Анализ пяти сил Портера (англ. Porter five forces analysis) — методика для анализа конкуренции в отрасли и выработки стратегии бизнеса, разработанная Майклом Портером в Гарвардской школе бизнеса в 1979 году.

Пять сил Портера включают в себя:

- анализ угрозы появления продуктов-заменителей;
- анализ угрозы появления новых игроков;
- анализ рыночной власти поставщиков;
- анализ рыночной власти потребителей;
- анализ уровня конкурентной борьбы.

Схематическое представление пяти сил Портера

Цепочка ценности

Цепочка ценности (англ.Value chain)— это инструмент стратегического анализа, направленный на подробное изучение деятельности организации с целью стратегического планирования. Идея цепочки ценности была предложена Майклом Портеромв книге «Конкурентное преимущество» для выявления источников конкурентного преимущества с помощью анализа отдельных видов деятельности компании.

Цепочка ценности «разделяет деятельность компании на стратегически важные виды деятельности с целью изучить издержки и существующие и возможные средства дифференциации». Конкурентное преимущество компании возникает как результат выполнения этих стратегических видов деятельности лучше конкурентов.

В большинстве компаний, вне зависимости от того, в какой отрасли она работает, присутствуют пять групп основных видов деятельности:

- входящая логистика;
- операции;

- исходящая логистика;
- маркетинг и продажи;
- сервис.

Базовые понятия языка

К базовым понятиям языка относятся понятия «элемент» и «отношение». Именно на основе элементов и отношений строятся описания (создаются модели) предприятия или его отдельных частей. Элементы – это «кирпичики» различного содержания, формы и предназначения,а отношения – различного рода соединения, связывающие элементы. Всего базовых элементов 32, а отношений – 12.

На рисунке 9 представлена обобщенная метамодель, содержащая основные понятия языка:

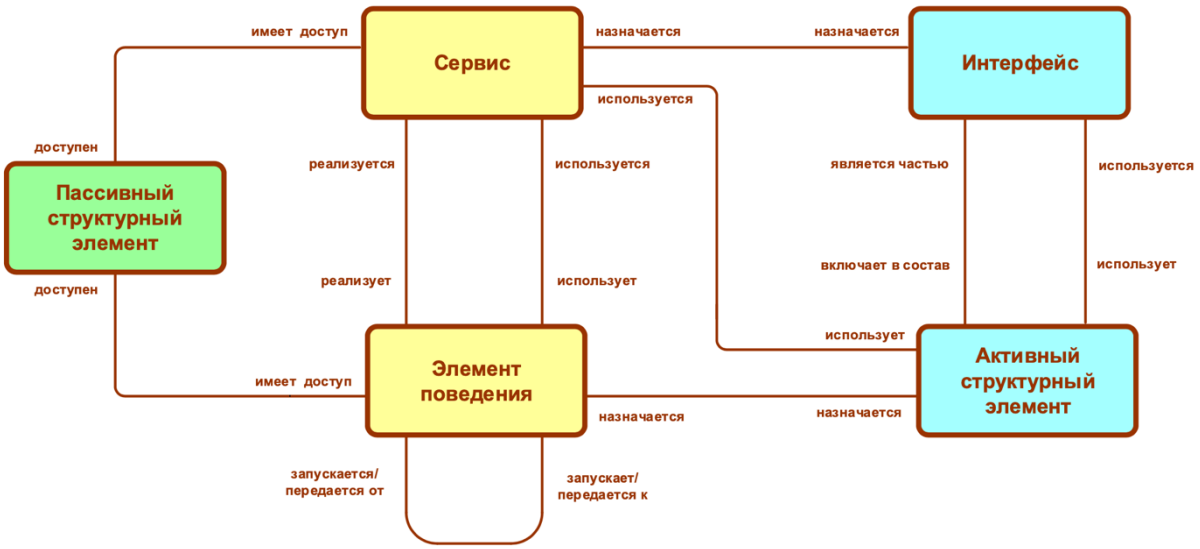


Рисунок 9 Обобщенная метамодель - основные понятия языка

Элементы языка

Элементы в языке различают по трем аспектам (измерениям):

- Структурный / поведенческий;
- Внешний / внутренний по отношению к окружению (взгляд на систему);
- Индивидуальный / коллективный.

В языке ArchiMate моделирование поведения (действий) отделяется от моделирования структуры.

В реальной жизни эта одна сущность.Это как две стороны одной монеты – одна не существует без другой.

Активный структурный элемент определяется как некая сущность, которая способна выполнять определенные действия.Это могут быть бизнес-исполнители, компоненты приложений или устройства, которые исполняют те или иные действия.

Пассивный структурный элемент определяется как некоторый объект, на котором или с которым выполняются действия.Обычно это информационные объекты или объекты данных, но также они могут быть использованы для представления физических объектов, над которыми выполняются те или иные действия.

Элемент поведения определяется как некоторая единица действия, выполняемая одним или несколькими активными структурными элементами. Элементами поведения являются процессы, функционалы, сервисы и события. Элементы поведения назначаются активным структурным элементам, чтобы показать, кто или что производит те или иные действия.

Следует отметить, что структура языка ArchiMate и разделение элементов по аспекту структурный/поведенческий напоминают структуру естественных (человеческих) языков, в состав предложений которых входят существительное-подлежащее (субъект), глагол-сказуемое (действие) и существительное- дополнение (объект).

В языке ArchiMate:

· существительное-подлежащее – это активный структурный элемент, то есть субъект поведения;

· глагол-сказуемое – это элемент поведения или действие, то есть выполнение поведения;

· существительное-дополнение - пассивный структурный элемент, то есть объект, на котором или с которым выполняется поведение.

3.8. **Представление архитектуры в методологии ARIS**

Общие принципы методологии и системы ARIS

ARIS – это одновременно и методология, и программный продукт, предназначенный для моделирования бизнес-процессов организаций. В дальнейшем под системой ARIS (либо инструментальной средой ARIS) будем понимать аппаратное и программное обеспечение, реализующие методологию ARIS, а под методологией ARIS – только подход к структурированному описанию деятельности организации. Методология ARIS представляет собой современный подход к структурированному описанию деятельности организации и представлению ее в виде взаимосвязанных и взаимодополняющих графических диаграмм, удобных для понимания и анализа.

Методология ARIS основывается на концепции интеграции, предлагающей целостный взгляд на процессы, и представляет собой множество различных методик, объединенных в рамках единого системного подхода.

ARIS – это сокращенное английское выражение (Architecture of Integrated Information Systems), что в переводе означает: архитектура интегрированных информационных систем. Под архитектурой подразумевается совокупность

технологий, обеспечивающих проектирование, управление, применение и реализацию бизнеса в виде «деловых» процедур бизнес-процессов предприятий и организаций, а также проектирование и создание интегрированных информационных систем поддержки бизнес-процессов.

Методология ARIS реализует принципы системного структурного анализа, основным понятием которого служит структурный элемент (объект).

Структурный анализ является методологической разновидностью системного анализа. В структурном анализе предполагается использование графического представления для описания структуры и деятельности организации. При этом реализуются основные принципы структурного анализа: разбиение на уровни абстракции с ограничением числа элементов на каждом уровне (обычно от 3 до 9); ограниченный контекст включающий только существенный на каждом уровне детали; использование строгих формальных правил записей; последовательное приближение к конечному результату (зависит от целей моделирования).

Методология ARIS также использует декомпозицию и позволяет детализировать предмет моделирования с помощью альтернативных или дополняющих друг друга моделей.

Основы методологии ARIS состоят в том, что любая организация рассматривается и визуально представляется во всех аспектах, т.е. как единая система, описание которой предусматривает четыре различных «взгляда»:

- Организационная структура
- Данные (потоки и структура)
- Функции («деревья» функций)
- Контроль и управление (деловые процессы)

Все данные подсистемы организации в реальности и в моделях должны быть связаны между собой. Методология ARIS дает возможность описывать достаточно разнородные подсистемы в виде взаимоувязанной и взаимосогласованной совокупности различных моделей, которые хранятся в едином репозитории (рисунок 10). Именно взаимосвязанность и взаимосогласованность моделей являются отличительными особенностями методологии ARIS.

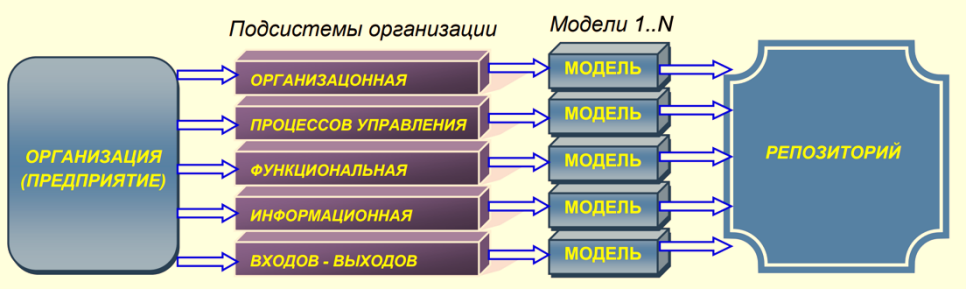


Рисунок 10. Структурная схема формирования репозитория.

В соответствии с правилами структурного анализа каждая из этих подсистем разбивается на элементарные блоки (модули), совокупность которых и составляет нотацию структурной модели той или иной подсистемы организации.

Естественно, что эти подсистемы не являются обособленными. Они взаимно проникают друг в друга, и поэтому одни и те же элементарные модули могут использоваться для описания различных структурных моделей. Для устранения избыточности методология ARIS ограничивает число типов моделей.

В связи с этим в методологии ARIS выделено пять типов представлений основных моделей, отражающих основные аспекты организации (рисунок 11):

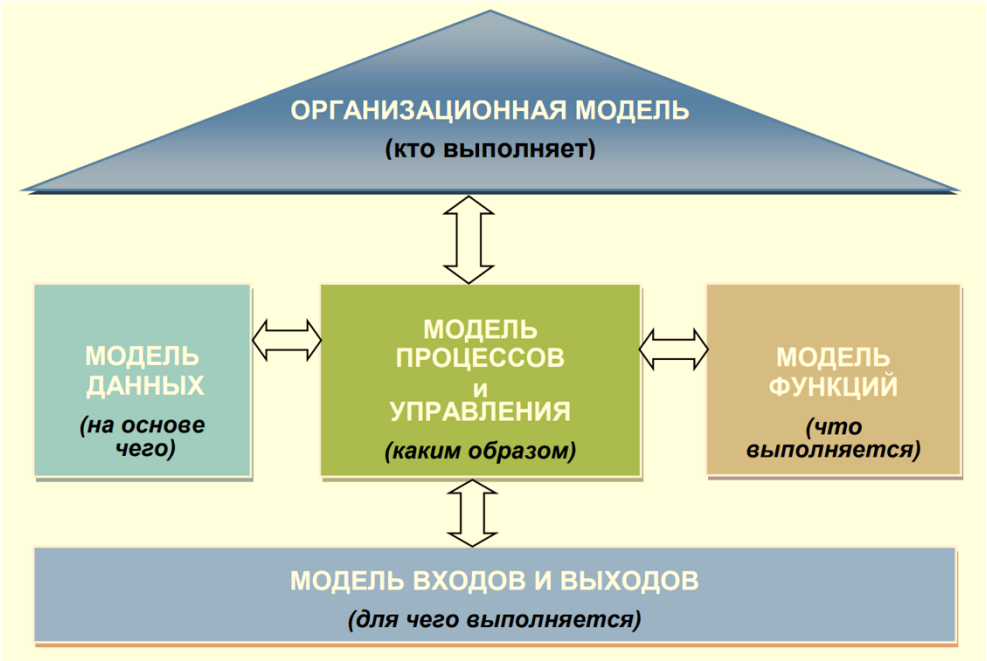


Рисунок 11. Взаимосвязи моделей деятельности организации

1. Организационные модели, описывающие иерархическую структуру системы, т.е. иерархию организационных подразделений, должностей, полномочий конкретных лиц, многообразие связей между ними, а также территориальную привязку структурных подразделений;
2. Функциональные модели, описывающие функции (процессы, операции), выполняемые в организации;
3. Информационные модели (т.е. модели данных), отражающие структуру информации, необходимой для реализации всей совокупности функций системы;
4. Модели процессов или управления, представляющие комплексный взгляд на реализацию деловых процессов в рамках системы и объединяющие вместе другие модели;
5. Модели входов и выходов, описывающие потоки материальных и нематериальных входов и выходов, включая потоки денежных средств.

Типы представления являются первой компонентой архитектуры. Они позволяют структурировать бизнес-процессы и выделять их составные части, что делает рассмотрение более простым. Применение этого принципа позволяет с различных точек зрения описывать содержание отдельных частей бизнес-процесса, используя специальные методы, наиболее полно соответствующие каждой точке

зрения. Это избавляет пользователя от необходимости учитывать множество связей и соединений.

Для построения моделей и проведения структурного анализа в ARIS используют следующие методы и средства визуального описания:

• DFD (Data Flow Diagrams) – диаграммы потоков данных для анализа и функционального проектирования моделей систем. Описывают источники и адресаты данных, логические функции, потоки данных и хранилища данных к которым осуществляется доступ;

• STD (State Transition Diagrams) – диаграммы перехода состояний для проектирования систем реального времени;

• ERD (Entity-Relationship Diagrams) – диаграммы сущность-связь, описывающие объекты (сущности), свойства этих объектов (атрибуты) и их отношения объектов (связи);

• SADT (Structured Analysis and Design Technique) - технология структурного анализа, проектирования и моделирования иерархических многоуровневых модульных систем;

• IDEF0 (Integration Definitionfor Function Modeling) – подмножество SADT – стандарт описания бизнес-процессов в виде иерархически взаимосвязанных функций;

• IDEF1 – стандарт описания движения информации; используется для определения структуры информационных потоков, правил движения, принципов управления информацией, связей потоков, выявления проблем некачественного информационного менеджмента;

• IDEF1X – стандарт разработки логических схем баз данных, основанный на концепции сущность-связь;

• IDEF3 – стандарт описания процессов, основанная на сценариях. Сценарий есть описание последовательности изменения свойств объекта в рамках некоторого процесса. Стандарт позволяет описать последовательность этапов изменения свойств объекта (Process Flow Description Diagrams - PFDD) и состояния объекта на этапах (Object State Transition Network - OSTN). Стандарт позволяет решать задачи документирования и оптимизации процессов;

• IDEF4 – стандарт описания структуры объектов и заложенных принципов их взаимодействия; позволяет анализировать и оптимизировать сложные объектноориентированные системы;

• IDEF5 – стандарт, позволяющий описать совокупность терминов, правил комбинирования терминов в утверждения для описания свойств и связей объектов, построить модель на основе этих утверждений. Такие модели позволяют изучать онтологию объектов. Онтология – это знания о совокупности фундаментальных свойств некоторого объекта или области, определяющих их поведение и изменение, собранные для детальной формализации;

• UML (Unified Modeling Language) – объектно-ориентированный унифицированный язык визуального моделирования. Позволяет описывать диаграммы действий, диаграммы взаимодействия, диаграммы состояний,

диаграммы классов и компонент. Используется как для анализа, так и для проектирования моделей информационных систем.

Другой особенностью методологии ARIS, обеспечивающей целостность разрабатываемой системы, является использование различных уровней описания, что поддерживает теорию жизненного цикла системы, существующего в сфере информационных технологий.

Для каждого «взгляда» придерживаются три уровня анализа (требования, спецификации, внедрения), что обеспечивает целостность разрабатываемой системы. Каждый уровень соответствует определенной фазе жизненного цикла информационной системы:

1. уровень определения требований (что система должна делать);
2. уровень проектной спецификации (основные пути реализации системы);
3. уровень описания реализации (физическое описание конкретных программных и технических средств.

Каждый из уровней анализа состоит из своего комплекта моделей различных типов, в том числе диаграмм UML, диаграмм SAP R/3 и др. Каждый объект моделей ARIS имеет множество атрибутов, позволяющих контролировать процесс разработки моделей, определить условия для выполнения функционально-стоимостного анализа, имитационного моделирования, взаимодействия с work flow-системами и т.д.

Архитектурные слои, области, типы архитектурных моделей

В структуре моделей выделены основные архитектурные слои и их компоненты (области), составляющие непересекающееся разбиение содержимого репозитория по функциональному назначению архитектурных моделей. Каждая область содержит модели, описывающие какой-либо аспект архитектуры ИС, например: модели бизнес-процессов, модели организационно-территориальной структуры. Модели архитектурной области строятся главным образом на основе классов объектов, соответствующих данной области (однако, могут включать и объекты других классов). Группировка архитектурных областей в слои соответствует важным для Компании аспектам управления архитектурой ИС (модели одного слоя имеют сходные механизмы управления).

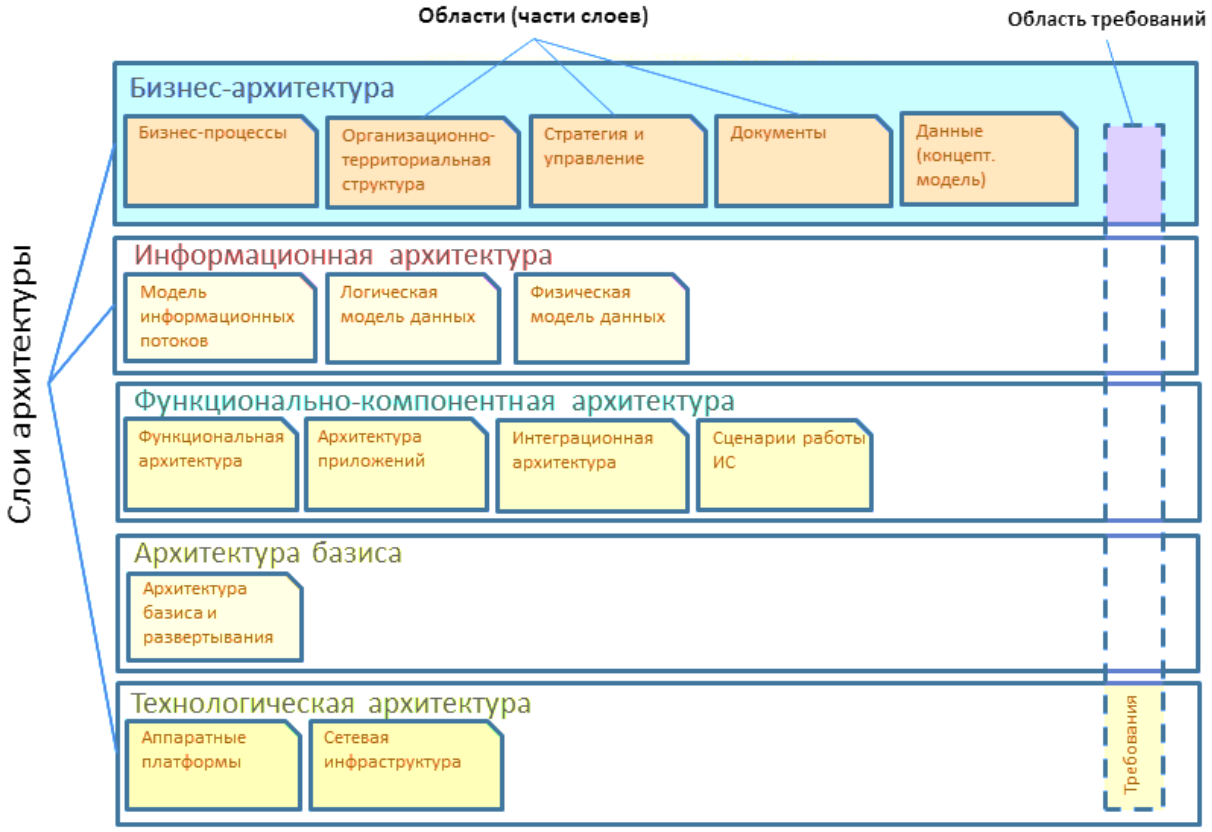


Рисунок 12. Структура слоев и областей архитектуры ИС

Базовыми архитектурными слоями являются:

- бизнес-архитектура;
- информационная архитектура;
- функционально-компонентная архитектура;
- архитектура базиса;
- технологическая архитектура.

В таблице 2 приводится описание архитектурных слоев:

Таблица 2. Описание архитектурных слоев.

Слой	Описание
Бизнес-архитектура	Модели существенных аспектов организационной, управленческой и процессной деятельности Компании
Информационная архитектура	Модели структуры информации в Компании, информационного взаимодействия и носителей данных
Функционально-компонентная архитектура	Модели структурного и функционального состава информационных систем Компании, а также их интеграции
Архитектура базиса	Модели прикладных и системных платформ и развертывания прикладных компонентов информационных систем
Технологическая архитектура	Модели технического обеспечения информационных систем Компании

Разбиение данных слоев на архитектурные области приведено в таблице 3.

Таблица 3. Разбиение слоев на архитектурные области.

Слой	Область	Описание
Бизнес-архитектура	Бизнес-процессы	Функциональные модели бизнес-процессов Компании
	Организационно-территориальная структура	Модели организационной и территориальной структуры Компании, структуры ролей участников бизнес-процессов
	Стратегия и управление	Модели целей и ключевых показателей деятельности Компании и бизнес-процессов
	Документы	Модели документационного обеспечения бизнес-процессов
	Данные (концептуальная модель)	Модели предметной области и используемых в бизнес-процессах бизнес-сущностей
Информационная архитектура	Модель информационных потоков	Модели информационных потоков между бизнес-решениями, бизнес-системами и организационными единицами
	Логическая модель данных	Логические информационные модели систем, взаимодействий (потоков, интеграционных сообщений) в терминах «сущность-связь»
	Физическая модель данных	Физические информационные модели баз данных
Функционально-компонентная архитектура	Функциональная архитектура	Модели функций информационных систем и их классификация
	Архитектура приложений	Модели структуры и взаимодействия бизнес-решений, бизнес-систем и их компонентов
	Сценарии работы ИС	Модели реализации бизнес-процессов в информационных системах и взаимодействия этих систем в ходе исполнения процессов
	Интеграционная архитектура	Модели, описывающие интеграцию ИС в терминах интерфейсов и передаваемых сообщений
Архитектура базиса	Архитектура базиса и развертывания	Модели, описывающие физические компоненты ИС (ландшафт), размещаемые на объектах инфраструктуры
Технологическая архитектура	Аппаратные платформы	Модели технических средств, обеспечивающих функционирование компонентов бизнес-систем и работу пользователей
	Сетевая инфраструктура	Модели технических средств, обеспечивающих взаимодействия между бизнес-системами, их компонентами и пользователями

Кроме того, в структуре архитектурного описания выделяются области, не принадлежащие к какому-то конкретному слою, либо охватывающие несколько базовых слоев (требования). Описание данных областей и правила их идентификации приведены в таблице 4.

Таблица 4. Описание слоя «требования».

Слой	Область	Описание
-	Требования	Модели, описывающие требования (бизнес-требования, функциональные и нефункциональные требования)

Спецификация метамодели

Метамодель (или «модель моделей») служит «словарем», определяющим разновидности объектов, составляющих архитектуру ИС и возможные

взаимоотношения (связи) между такими объектами, используемые при моделировании. Иными словами, метамодель описывает понятия, используемые при моделировании и имеющие графическое изображение в различных ракурсах представления.

Метамодель включает как классы объектов архитектуры ИС, так и классы объектов бизнес-архитектуры, которые могут вместе входить в одни ракурсы представления. На рисунке 13 представлен пример метамодели архитектуры ИС.

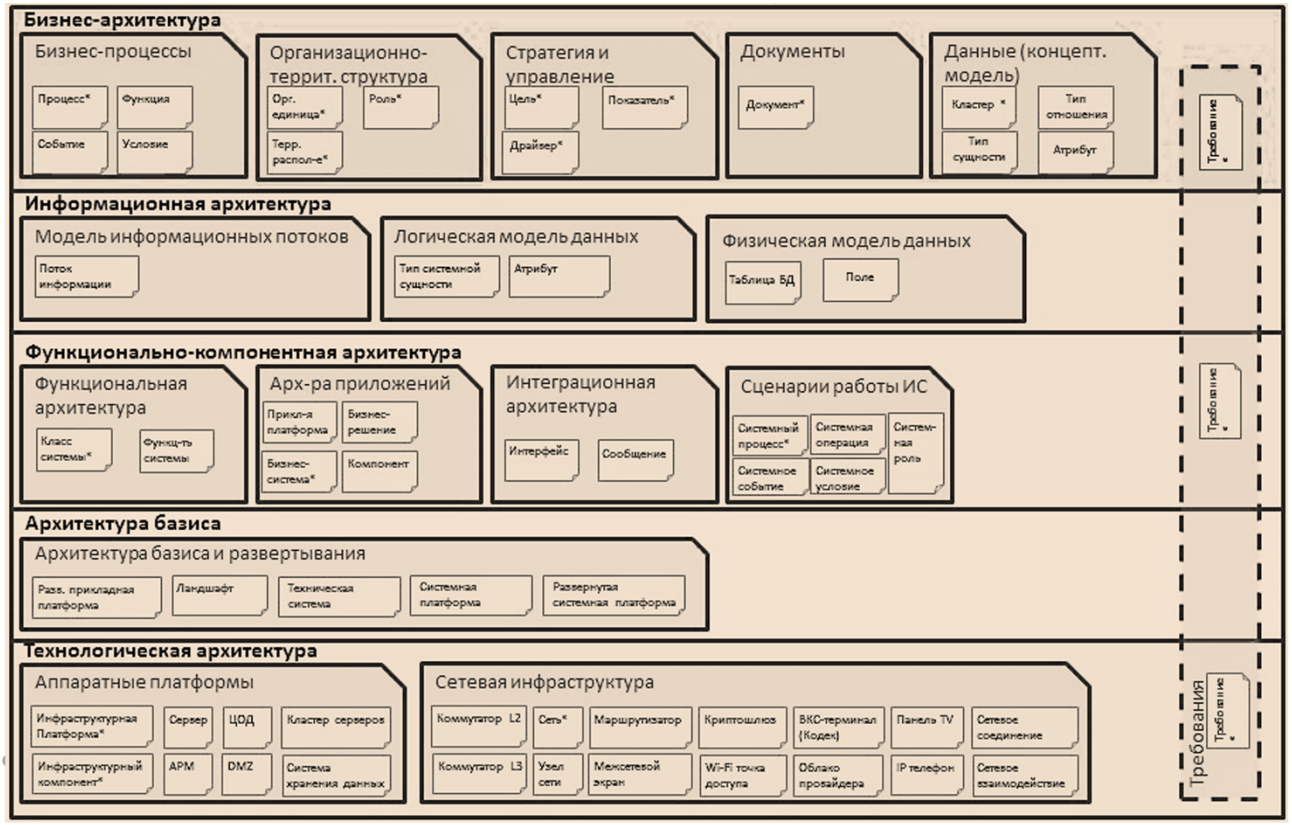
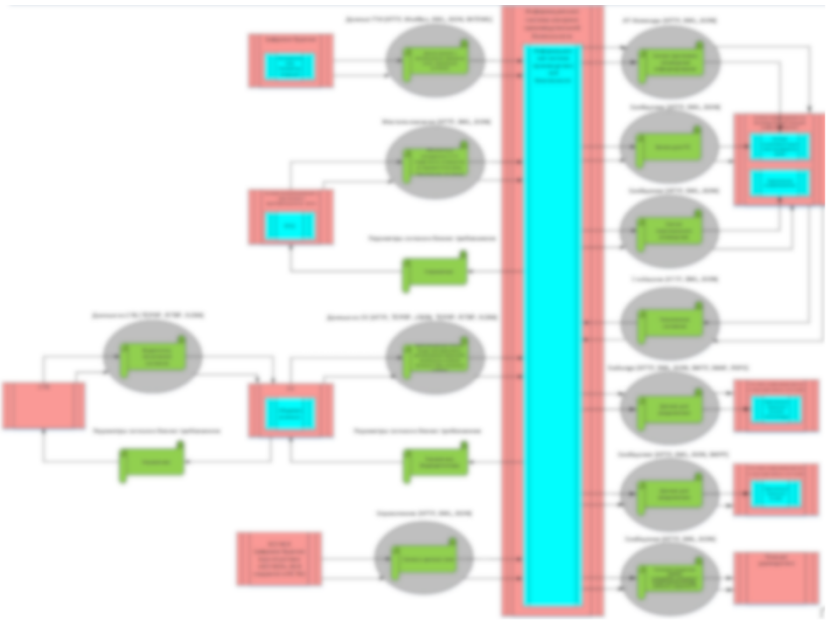


Рисунок 13. Метамодель архитектуры ИС.

Пример документа в нотации ARIS



4. Виды и содержание документов проектирования ИС

Для формирования необходимого и достаточного покрытия разработки ПО и ИС можно выделить следующий перечень документов:

1. SPMP (Software Project Management Plan)
2. SRS (Software Requirements Specification) – Требования к программному продукту, Техническое Задание, Функциональные-технические требования;
3. SDD (Software Design Document), AD (Architectural Design), SAD: Systematic Architecture Design – эскизный проект, описание программы, etc.
4. SVVP (Software Verification and Validation Plan)
5. SQAP (Software Quality Assurance Plan)
6. STD (Software Test Documentation) – Программа и методики тестирования
7. SCMP (Software Configuration Management Plan)
8. Source code

Рассмотрим документы используемые на этапе проектирования.

4.1. Software Requirement Specification — SRS

Требования к программному обеспечению могут быть задокументированы в техническом задании или документе Software Requirements Specification.

SRS — специальная документация для ПО которая содержит в себе информацию о том, как должна себя вести система, какие функции должна выполнять, какую нагрузку должна выдерживать и т.д.

Рекомендуемая стандартом IEEE 830 структура SRS:

- Введение
  - Цели
  - Соглашения о терминах
  - Предполагаемая аудитория и последовательность восприятия
  - Масштаб проекта
  - Ссылки на источники

- Общее описание
  - Видение продукта
  - Функциональность продукта
  - Классы и характеристики пользователей
  - Среда функционирования продукта (операционная среда)
  - Рамки, ограничения, правила и стандарты
  - Документация для пользователей
  - Допущения и зависимости

- Функциональность системы
  - Функциональный блок X (таких блоков может быть несколько)
    - Описание и приоритет
    - Причинно-следственные связи, алгоритмы (движение процессов, workflows)
    - Функциональные требования

- Требования к внешним интерфейсам
  - Интерфейсы пользователя (UX)
  - Программные интерфейсы
  - Интерфейсы оборудования
  - Интерфейсы связи и коммуникации

- Нефункциональные требования
  - Требования к производительности
  - Требования к сохранности (данных)
  - Критерии качества программного обеспечения
  - Требования к безопасности системы

- Прочие требования
  - Приложение А: Глоссарий
  - Приложение Б: Модели процессов и предметной области и другие диаграммы
  - Приложение В: Список ключевых задач

Структура SRS:

- Introduction
  - Purpose
  - Document conventions
  - Intended Audience and Reading Suggestions
  - Project scope
  - References
- Overall Description
  - Product perspective
  - Product features
  - User classes and characteristics
  - Operating environment
  - Design and implementation constraints
  - User documentation
  - Assumptions and dependencies
- System features
    - System feature X (таких блоков может быть несколько)

        - Description and priority
        - Stimulus/Response sequences
        - Functional requirements
- External interface requirements
    - User interfaces
    - Software interfaces
    - Hardware interfaces
    - Communication interfaces
- Non functional requirements
    - Performance requirements
    - Safety requirements
    - Software quality attributes
    - Security requirements
- Other requirements
- Appendix A: Glossary
- Appendix B: Analysis Models
- Appendix C: Issues list

Пояснение каждого пункта структуры SRS:

Introduction\Purpose

В данной секции описываем приложение или продукт, функционал которого будет описываться в SRS.

Introduction\Document conventions

Здесь используются все непонятные технические слова или термины которые встречаются в SRS. Заметьте, что описание непонятного слова не может содержать другое непонятное слово. Старайтесь расписать как можно подробнее термин который Вы используете простым и понятным всем языком. Не экономьте на этой секции потому, что чем больше вы распишете непонятных вещей, тем проще будет потом проектировать.

Introduction\References

В данной секции мы пишем ссылки на литературу в которой можно найти основания использованных технологий и фактов. Допустим можно вставлять RFC если вы пишете приложение работающее с TCP/IP, вставлять ссылки на документы на которые вы ссылаетесь и т.д.

Overall Description\Product features

В данном разделе мы описываем части функционала на высоком уровне. Более детально каждая часть функционала будет описана в своем разделе ниже. Тут желательно разместить DFD-диаграмму которая покажет общее взаимодействие системы.

Overall Description\Operating environment

Как понятно из названия раздела тут мы описываем окружение в котором будет работать продукт. ОС, версии компиляторов, базы данных, сервера, софт, железо и другие вещи которые необходимы для работы вашего продукта.

Overall Description\Design and implementation constraints

Данный раздел определяет стандарты и ограничения разработки. Такими ограничениями могут быть, например, такие вещи:

- Язык программирования, база данных
- Стандарты кодирования
- Стандарты обмена данными
- Ограничения накладываемые Operational Enviroment, допустим совместимость
- Ограничения которые могут быть наложены бизнес-логикой проекта

Overall Description\User documentation

Описываем какая документация нужна для пользователей данного продукта. Возможно это книга по HTML если это HTML редактор.

System features\System feature X

Называем функцию (фичу) проекту и даем ей уникальный идентификатор. Например, server.document.editor. Уникальный идентификатор дается для того, чтобы единообразно ссылаться на неё в документах или описаниях.

System features\System feature X\Description and priority

Описываем детально фичу продукта. Для чего она? Что должна делать? Какой у нее приоритет выполнения?.. Из этого раздела читающему SRS человеку должно сразу стать понятно зачем этот функционал присутствует в системе.

System features\System feature X\Stimulus/Response sequences

Триггер запуска фичи. Когда она запускается и как себя ведет при запуске? Например, HTML редактор показывается при нажатии пользователя на ссылку меню Открыть HTML редактор

System features\System feature X\Functional requirements

Подробное и детальное описание фичи. Описываем все: как работает, как реагирует на ошибки, что должно проверять, как отображать данные, как и куда что сохраняет и т.д.

External interface requirements

Описание того как система будет взаимодействовать с внешним миров. Если будет конечно. Какие API, как получить те или иные данные и т.п. Подразделы служат для детализации требований. Какие софт интерфейсы, «аппаратные» интерфейсы, комуникационные интерфейсы и прочее.

Non functional requirements

Не функциональные требования. Есть требования которые невозможно описать как какую то фичу, например, требования к безопасности, доступности, восстанавливаемости.

Non functional requirements\Performance requirements

Требования к производительности. Это не фича, однако налагает определенные ограничения. Допустим база данных проекта должна выдерживать

1000 запросов в секунду и т.п. Эти требования приводят к колоссальной работе по оптимизации проекта.

Non functional requirements\Software quality attributes

Тут мы описываем требования к качеству кода. Какие тесты использовать? Какие метрики использовать для определения качества кода? Сколько кода должно быть покрыто тестами?

Non functional requirements\Security requirements

Требования по безопасности. Если это HTML редактор, через которые можно изменять что-то на сайте, то это может быть нечто вроде: через HTML редактор не должно быть возможности поставить shell на сервер и т.п.

Appendix A: Glossary

Приложение. Иногда в SRS пытаются вставлять описание протоколов и т.д и т.п. Этого делать не нужно. Однако иногда нужно таки прояснить какую то концепцию. Для этого существует этот раздел. Вставляем ссылочки на такие пояснения. Такой словарик.

Appendix B: Analysis Models

Раздел определяет какие диаграммы нужно использовать при написании SRS. Например, DFD, какие то общие диаграммы взаимодействия и работы

Appendix C: Issues list

Данный раздел используется для очень формальных SRS. Описывает пункты TBD(To Be Done) — что в будущем надо еще сделать, но тут не описано.

4.2. Software design description – SDD

Следующий важный документ имеет несколько наименований SAD/SDD, используемых в разном контексте проектирования ПО и ИС, но имеет схожее назначение.

Вот некоторые из них:

SAD:

- Systematic Architecture Design
- Solution Architecture Definition
- Software Architecture Description
- Software Architecture Design

SDD:

- Software design description

Описание разработки программного обеспечения (так называемый дизайн программного документа или SDD , просто дизайн документ , также дизайн Спецификация программного обеспечения) является представлением разработки программного обеспечения, которое будет использоваться для записи информации о конструкции, решении различных проблем проектирования, и передавать эту информацию в дизайнах заинтересованным сторонам.

SDD обычно сопровождает диаграмму архитектуры с указателями на подробные характеристики более мелких частей проекта.

На практике описание требуется для координации большой команды в рамках единого видения, оно должно быть стабильным справочником и описывать все части программного обеспечения и то, как они будут работать.

Описание конструкции программного обеспечения.

SDD обычно содержит следующую информацию:

1) Дизайн данных описывает структуры, которые находятся в программном обеспечении.
2) Атрибуты и отношения между объектами данных диктуют выбор структур данных . В дизайне архитектуры использует информацию , протекающие характеристики, и отображают их в структуру программы.
3) Метод преобразования преобразования применяется для демонстрации четких границ между входящими и исходящими данными.
4) Диаграммы потоков данных распределяют управляющий ввод, обработку и вывод по трем отдельным модулям.
5) Дизайн интерфейса описывает внутренние и внешние программные интерфейсы, а также дизайн человеческого интерфейса .
6) Дизайн внутреннего и внешнего интерфейса основан на информации, полученной из модели анализа.

Процедурно дизайн описывает структурированные концепции программирования с использованием графического, табличные и текстовые обозначения. Эти средства проектирования позволяют архитектору представлять детали процедуры, что облегчает преобразование в код. Этот план реализации является основой для всех последующих работ по разработке программного обеспечения.

Цель документа архитектуры программного обеспечения (SDD) - предоставить информацию, которая дополняет код.

На высоком уровне это может включать в себя:

1) Общее описание архитектуры программного обеспечения, включая основные компоненты программного обеспечения и их взаимодействие.
2) Общее понимание движущих сил (требований, ограничений и принципов), которые влияют на архитектуру.
3) Описание аппаратных и программных платформ, на которых построена и развернута система.

Стоит отметить, что все эти представления не обязательно требуются при учете таких факторов, как размер проекта, сложность, требования, команда и т.д. Если информация доступна в другом месте, документ архитектуры программного обеспечения должен содержать ссылку на этот источник, а не повторять его

(например, нефункциональные требования могут быть изложены в документации требований, а дизайн - в отдельных проектных документах или модели UML).

SDD - это живой документ, и в любой момент времени он может не содержать всей перечисленной здесь информации.

1. Контекст
2. Функциональный вид
3. Представление процесса
4. Нефункциональный вид
5. Ограничения
6. Принципы
7. Логический взгляд
8. Интерфейсный взгляд
9. Вид дизайна
10. Вид инфраструктуры
11. Представление развертывания
12. Операционный вид
13. Представление безопасности
14. Представление данных
15. Выбор технологии
16. Обоснование архитектуры

5. Стандарты относящие к разработке ПО и ИС

6.

6.1. Международные стандарты.

- ISO/IEC/IEEE 29148:2018 Systems and software engineering — Life cycle processes — Requirements engineering
- CMMI® for Development, Version 1.3 - 2010/ CMMI v2.0 - 2018
- ISO/IEC 20246:2017(en) Software and systems engineering — Work product reviews.
- ISO/IEC TR 29110-5-6-1:2015(en) Systems and software engineering — Lifecycle profiles for Very Small Entities (VSEs) — Part 5-6-1: Systems engineering — Management and engineering guide: Generic profile group: Entry profile.
- ISO/IEC/IEEE 29148:2011(E) - ISO/IEC/IEEE International Standard - Systems and software engineering -- Life cycle processes --Requirements engineering.
- ISO/IEC TR 29110-1:2016 2nd Software engineering -- Lifecycle profiles for Very Small Entities (VSEs) -- Part 1: Overview.
- ISO/IEC TR 29110-3-1:2015 1st Systems and software engineering -- Lifecycle profiles for Very Small Entities (VSEs) -- Part 3-1: Assessment guide.
- ISO/IEC TR 29110-5-6-1:2015 1st Systems and software engineering -- Lifecycle profiles for Very Small Entities (VSEs) -- Part 5-6-1: Systems engineering -- Management and engineering guide: Generic profile group: Entry profile. ISO/IEC/IEEE 15288, Systems and software engineering — System life cycle processes.
- ISO/IEC/IEE 24748-4, Systems and software engineering — Lifecycle management — Part 4: Systems engineering planning.
- ISO/IEC/IEEE 24765, Systems and software engineering — Vocabulary.
- IEEE Std 1016TM-2009 IEEE Standard for Information Technology—Systems Design— Software Design Descriptions

6.2. Международные стандарты адаптированные под ГОСТ

- ГОСТ Р ИСО/МЭК 25010-2015 Информационные технологии (ИТ). Системная и программная инженерия. Требования и оценка качества систем и программного обеспечения (SQuaRE). Модели качества систем и программных продуктов.
- ГОСТ Р ИСО/МЭК 90003-2014 Разработка программных продуктов. Руководящие указания по применению ИСО 9001:2008 при разработке программных продуктов.
- ГОСТ Р 57193-2016 Системная и программная инженерия. Процессы жизненного цикла систем. ISO\IEC\EEE 15288:2015 NEQ. ГОСТ Р ИСО-МЭК 15288-2005 Процессы жизненного цикла систем
- ГОСТ Р ИСО/МЭК 12207-2010 Информационная технология (ИТ). Системная и программная инженерия. Процессы жизненного цикла программных средств.
- ГОСТ Р 57100-2016/ISO/IEC/IEEE 42010:2011 Системная и программная инженерия. Описание архитектуры.

6.3. ГОСТ

6.3.1. Общего назначения

- ГОСТ Р ИСО/МЭК 9126-93. Информационная технология. Оценка программной продукции. Характеристики качества и руководства по их применению.
- ГОСТ 15971-90 Системы обработки информации. Термины и определения.
- ГОСТ 19781-90 Обеспечение систем обработки информации программное. Термины и определения.
- ГОСТ 28806-90 Качество программных средств. Термины и определения.
- ГОСТ 28195-89 Оценка качества программных средств. Общие положения.
- Стандарты ЕСКД – Единая система конструкторской документации
- ГОСТ 2.001-2013 ЕСКД. Общие положения

6.3.2. Стандарты АСУ ГОСТ 34 – Автоматизированные системы управления

Автоматизированная система (АС) — система, состоящая из персонала и комплекса средств автоматизации его деятельности, реализующая информационную технологию выполнения установленных функций. В зависимости от вида деятельности выделяют, например, следующие виды АС: автоматизированные системы управления (АСУ), системы автоматизированного проектирования (САПР), автоматизированные системы научных исследований (АСНИ) и другие.

ГОСТ 34 разделяет виды обеспечения АС:

- Организационное;
- Методическое;
- Техническое;
- Математическое;
- Программное обеспечение;
- Информационное;
- Лингвистическое;
- Правовое;
- Эргономическое.

Автоматизированная система — это не программа, а комплекс видов обеспечения, среди которых есть и программное обеспечение.

- ГОСТ 34.601-90 Автоматизированные системы. Стадии создания
- ГОСТ 34.602-89 Техническое задание на создание автоматизированной системы (Взамен ГОСТ 24.201-85)
- ГОСТ 34.201-89 Виды, комплектность и обозначения документов при создании автоматизированных систем
- РД 50-34.698-90 Автоматизированные системы. Требования к содержанию документов
- ГОСТ 34.320-96 Концепции и терминология для концептуальной схемы и информационной базы
- ГОСТ 34.321-96 Информационные технологии. Система стандартов по базам данных. Эталонная модель управления

6.3.3. Стандарты ЕСПД ГОСТ 19 – Единая система программной документации

ЕСПД (единая система программной документации) - комплекс государственных стандартов (ГОСТ), устанавливающих взаимосвязанные правила разработки, оформления и обращения программ и программной документации. Стандарты ЕСПД устанавливают требования, регламентирующие разработку, сопровождение, изготовление и эксплуатацию программ.

Определения из ЕСПД:

- Программа — данные, предназначенные для управления конкретными компонентами системы обработки информации в целях реализации определённого алгоритма.
- Программное обеспечение — совокупность программ системы обработки информации и программных документов, необходимых для эксплуатации этих программ. ГОСТ 19.

- ГОСТ 19.001-77 Общие положения
- ГОСТ 19.002-80 Схемы алгоритмов и программ. Правила выполнения
- ГОСТ 19.003-80 Схемы алгоритмов и программ. Обозначения условные графические
- ГОСТ 19.004-80 Термины и определения
- ГОСТ 19.005-85 Р-схемы алгоритмов и программ. Обозначения условные графические и правила выполнения
- ГОСТ 19.101-77 Виды программ и программных документов
- ГОСТ 19.102-77 Стадии разработки
- ГОСТ 19 105-78 Общие требования к программным документам
- ГОСТ 19.301-79 Программа и методика испытаний. Требования к содержанию и оформлению
- ГОСТ 19.402-78 Описание программы. Требования к содержанию и оформлению
- ГОСТ 19.404-79 Пояснительная записка. Требования к содержанию и оформлению
- ГОСТ 19.602-78 Правила дублирования, учета и хранения программных документов, выполненных печатным способом
- ГОСТ 19.603-78 Общие правила внесения изменений
- ГОСТ 19.604-78 Правила внесения изменений в программные документы, выполненные печатным способом
- ГОСТ 19.701-90 Схемы алгоритмов, программ, данных и систем. Условные Обозначения и правила выполнения

6.3.4. Отличия ГОСТ 34 и ГОСТ 19

- Автоматизированная система, как правило, содержит организационное решение под конкретного пользователя и заказчика.
- Программа может быть создана и растиражирована под большое количество пользователей без привязки к какому-либо предприятию.
- Если разрабатывается документация на программу, которую создают под конкретное предприятие, то используется ГОСТ 34.

- Если разрабатывается документация на массовую программу, то используется ГОСТ 19.
- Пункты технического задания ГОСТа 34 и ГОСТа 19 отличаются.

6.4. Список основных стандартов для написания документации:

- IEEE Std 1063-2001 «IEEE Standard for Software User Documentation» — стандарт для написания руководства пользователя;
- IEEE Std 1016-1998 «IEEE Recommended Practice for Software Design Descriptions» — стандарт для написания технического описания программы;
- ISO/IEC FDIS 18019:2004 «Guidelines for the design and preparation of user documentation for application software» — ещё один стандарт для написания руководства пользователя. Содержит большое количество примеров, похоже на руководство по написанию руководства пользователя.
- ISO/IEC 26514:2008 «Requirements for designers and developers of user documentation» — стандарт для дизайнеров и разработчиков пользователей документации.
- ISO/IEC/IEEE 15289, Systems and software engineering — Content of life-cycle information items (documentation).
- 1016-2009 - IEEE Standard for Information Technology-Systems Design- Software Design Descriptions.
- ГОСТ Р ИСО/МЭК ТО 9294-93 Информационная технология (ИТ). Руководство по управлению документированием программного обеспечения.
- ГОСТ Р ИСО-МЭК 15910-2002 Процесс создания документации пользователя программного средства
- ГОСТ Р ИСО 9127-94 Системы обработки информации. Документация пользователя и информация на упаковке для потребительских программных пакетов

7. Домашнее задание

Собрать комплект архитектурной документации проекта (рабочий, либо личный проект):

- Указать фазу жизненного цикла ИС или ПО.
- Вид и назначение документа, нотация.
- Кто потребитель данных документов.

Используемые источники

V&B
https://itabok.iasaglobal.org/itabok3_0/views-and-viewpoints/sei/
Архитектурные представления
https://portal.tpu.ru/SHARED/m/MAXIMKA/uchebnaya_rabota_Pavel_Banokin/Tab1/Lection_software_design.pdf
ARIS
https://portal.tpu.ru/SHARED/h/haperskaya/Materials/IT/Уч-мет.ARIS%20(1).pdf
SRS
http://dit.isuct.ru/Publish_RUP/index.htm#extend.formal_resources/guidances/templates/srs_traditional_C9394A15.html
http://seal.ifi.uzh.ch/fileadmin/User_Filemount/Vorlesungs_Folien/SOPRA/IEEE-SRS-Standard.pdf
SDD
https://ru.abcdef.wiki/wiki/Software_design_description
SAD
https://www.se.rit.edu/~co-operators/SoftwareArchitectureDocumentation.pdf
https://ru.wikipedia.org/wiki/Архитектура_программного_обеспечения
ГОСТ
https://docs.cntd.ru/document/1200139542
PM-ODP
https://en.wikipedia.org/wiki/RM-ODP
TOGAF-ArchiMate
https://en.wikipedia.org/wiki/The_Open_Group_Architecture_Framework
https://www.cfin.ru/itm/EA_ArchiMate.shtml
BSC
https://ru.wikipedia.org/wiki/Сбалансированная_система_показателей
5 forces
https://ru.wikipedia.org/wiki/Анализ_пяти_сил_Портера
4+1
https://en.wikipedia.org/wiki/4%2B1_architectural_view_model
SAD
https://www.sap.com/cxworks/article/2589632501/solution_architecture_definition_sad_template_download
SRS
http://www.garshin.ru/it/_pdf/standards/ieee-std-830-1993-rus.pdf
SLCM
https://webstore.iec.ch/preview/info_isoiecieee29148%7Bed2.0%7Den.pdf