Документ «Детальная архитектура» (architecture)

Разработка документации

Exported on 02/02/2023

Table of Contents

1 Неприменимость документа

Невозможна! Так как если ПО существует, то у него есть некая структура. А если у него есть некая структура, то существует и его архитектура как совокупность структурных элементов и их интерфейсов, с помощью которых составлена система, а также их поведения в рамках сотрудничества структурных элементов.

2 Неприменимость по разделам

Раздел	Неприменимость	Может ли поменяться в будущих релизах
Цель создания (objective)	**Невозможна.** Цель создания — это ответ на вопрос: «Зачем существует программный компонент? Какую потребность закрывает (какие задачи потребителя решает)?».	-
Концептуальная модель предметной области (conceptual-model), диаграмма классов	**Невозможна.** Диаграмма классов имеет прямую связь с ООП и может описывать любую классификацию объектов, будь то: • объекты реального мира (класс автомобиль наследует некоторые атрибуты класса наземный транспорт, который наследует атрибуты класса транспортные средства). Это концептуальный уровень диаграмм классов. Может не содержать методов и атрибутов. • бизнес-логика (класс Заказ связан с зависимым классом Стоимость заказа). Это спецификационный уровень диаграмм классов. Может не содержать методов и атрибутов. • непосредственно уровень кода. Это имплементационный уровень диаграмм классов, иногда уже уровень диаграммы объектов (то есть не просто параметров и типов параметров в атрибутах, а атрибутов со значениями параметров). Код одних языков проще переложить на такую диаграмму (так называемых объектно-ориентированных языков программирования), чем других (сложно переложить код функциональных языков на ООП, например, Golang).	Раздел в любом случае применим, но непосредственным разработчикам документации может быть сложно сразу разобраться, как нарисовать диаграмму классов, например, для нефункциональных языков, а сроки релизов поджимают. + не знают, не понимают нотацию UML
Основные функции (functions)	**Невозможна.** Если компонент существует, то он осуществляет некую работу и некие задачи, функционирует и имеет функции.	-

Раздел	Неприменимость	Может ли поменяться в будущих релизах
Варианты использования (use-cases)	**Невозможна.** Если компонент существует, то он так или иначе используется: • акторами «живыми людьми» — это непосредственно акторы по ролевой модели компонента, причем не только пользователь/оператор в GUI, но и системный администратор, и прикладной разработчик, то есть любой человек, который так или иначе взаимодействует с системой. • акторами «сервисами» — это сервисы, которые также взаимодействуют с системой компонента, например, отправляют в нее данные или получают их от нее. Это применимо и для библиотек	Раздел в любом случае применим, но непосредственным разработчикам документации может быть сложно сразу разобраться, как нарисовать диаграмму вариантов использования, например, для компонента без GUI, а сроки релизов поджимают. + не знают, не понимают нотацию UML
Сценарии использования (scenarios)	**Невозможна.** Сценарии использования — это по сути текстовая легенда для диаграммы вариантов использования, где сценарии строятся на основе вариантов использования.	-

Раздел	Неприменимость	Может ли поменяться в будущих релизах
Нефункциональные требования к программному компоненту (non-functional-requirements)	**Невозможна/возможна** ? У любого ПО есть функциональные (непосредственно требования к функциям ПО) и нефункциональные требования, где, например, BABOK в технике «Анализ нефункциональных требований» выделяет следующие их категории: • ***доступность*** – степень работоспособности и доступности решения для использования, часто выражается в процентах времени; • ***совместимость*** — степень успешности взаимодействия решения с другими окружающими компонентами (бизнес-процессами, информационными системами, аппаратным обеспечением и пр.); • ***функциональность*** — степень соответствия функций решения потребностям пользователей, включая пригодность, точность, совместимость, т.е. корреляция с требованиями стейкхолдеров; • ***ремонтопригодность*** — легкость изменения решения или его компонента, чтобы улучшить их, исправить ошибки или адаптировать к изменениям окружающей среды; • ***эффективность работы*** — способность решения или его компонента выполнять свои целевые функции с минимальным потреблением ресурсов в контексте или временном периоде, например, в условиях пиковой нагрузки; • ***надежность*** — способность решения или его компонента выполнять требуемые функции в определенных условиях в течение конкретного периода времени, например, наработка на отказ, измеряемая в часах и равная среднему времени работы устройства до сбоя; • ***масштабируемость*** — способность решения расти или развиваться по мере роста объемов данных и количества пользователей; • ***расширяемость*** – возможность добавления в решение новых функциональных возможностей. Это свойство тесно связано с архитектурой ПО, например, микросервисная архитектура проще поддается модификациям, чем «монолитная». • ***переносимость*** — легкость переноса решения или его компонента из одной среды в другую, например, миграция на новую версию операционной системы или аппаратной платформы;	Стандарт в плане того, что относится к нефункциональным требованиям, почему-то выделяет только: • время отклика (latency), • пропускная способность (throughput), • объем одной транзакции, • безопасность • и т.д. Время отклика, пропускная способность и объем транзакции по большому счету в большей степени относятся к компонентам с сетевыми взаимодействиями. Однако есть компоненты, которые устанавливаются локально, а безопасность обеспечиваются средствами опенсорсной платформы, тогда именно указанные категории нефункциональных требований могут быть к ним практически неприменимы. Однако остается категория т. д., которая может как пониматься очень широко и попадать под всю категоризацию из BABOK, так и не пониматься никак, тогда разработчик документации будет примерять на свой компонент только 5 выделенных нефункциональных требований.

Раздел	Неприменимость	Может ли поменяться в будущих релизах
	• ***безопасность*** – защита данных и компонентов решения от случайного или злонамеренного доступа, неправомерного использования, разрушения или раскрытия; • ***удобство использования*** – легкость взаимодействия пользователя с решением, включая простоту ежедневной работы и обучения; • ***сертификация*** — соответствие отдельным стандартам или отраслевым соглашениям; • ***соответствие*** — нормативные, финансовые или правовые ограничения в зависимости от контекста и юрисдикции; • ***локализация*** – адаптация текстовых и графических компонентов интерфейса, а также представления данных к языкам, законам, валютам и особенностям культуры пользователей в определенной местности или отрасли; • ***соглашения об уровне обслуживания*** между поставщиком и пользователем решения. На практике это свойство тесно связано с доступностью, о чем мы поговорим далее.	

Раздел	Неприменимость	Может ли поменяться в будущих релизах
Компонентно-логическая диаграмма (logic-diagram)	**Невозможна/возможна** ? Компонентно-логическая диаграмма — это диаграмма, которая показывает, из каких логических частей (или компонент) состоит система и по каким интерфейсам они общаются между собой. По сути это более абстрактная и концептуальная форма диаграммы развертывани или переходная форма от той же диаграммы классов к диаграмме развертывания (т.е. определили классы, теперь определим в компонентно-логической диаграмме, как они будут лежать, а в диаграмме развертывания — где именно будут лежать). Например, компонентно-логическая диаграмма покажет, что этот микросервис ходит в этот и вот этот микросервис по вот такому протоколу. А вот диаграмма развертывания показывает физическое оборудование, на котором будет работать программная система и связь между этим оборудованием. Например: у тебя есть X серверов в кубере и конфигурации для них лежат где-то на Y серваке. И еще где-то сервер приложений на физической машине. Можно это изобразить на такой схеме.	По идее любое ПО раскладываемо на компонентно-логические составляющие, однако хотелось бы понять до конца взаимосвязанность компонентной-логической диаграммы с диаграммы с диаграммой развертывания. Так как диаграмма развертывания зачастую по умолчанию включает в себя компонентно-логическую диаграмму, где диаграмма развертывания — конкретная реализация компонетно-логической диаграммы. А потому если дл компонента предполагается только один вариант развертывания, то отказаться от отрисовки компонентно-логической диаграммы может быть вполне правомерно. Однако в дальнейшем вариантов развертывания может стать несколько, где при реальном развертывании могут задействоваться не все возможные элементы компонентной-логической диаграммы. То есть, например, мы захотим показать полное и неполное развертывание. И тогда компонетно-логическая диаграмма должна будет возникнуть отдельно от диаграмм развертывания как их абстрактный, концептуальный и более полный исходник.

Раздел	Неприменимость	Может ли поменяться в будущих релизах
Компоненты программного компонента (components)	**Невозможна/возможна** ? Компоненты программного компонента — это по сути текстовая легенда для компонентной-логической диаграммы.	Употребимость привязана к употребимости компонентной-логической диаграммы.
Интерфейсы (interfaces)	**Невозможна/возможна** ? Интерфейс в UML — это структура программы, определяющая отношение с объектами, объединенными некоторым поведением. **Интерфейс** – это набор методов класса, доступных для использования. Интерфейсом класса будет являться набор всех его публичных методов в совокупности с набором публичных атрибутов. По сути, интерфейс специфицирует класс, чётко определяя все возможные действия над ним. Подробнее здесь[1] и вот здесь[2]. Стандарт в плане того, что относится к интерфейсам, выделяет, что интерфейсы — это то, что имеет: • идентификаторы; • названия операций; • сигнатуры операций; • реализующие компоненты. Это термины, по которым понятие интерфейсов относится к пониманию интерфейсов в нотации UML и относительно диаграммы классов.	А если диаграмма классов компонента не описывает операционный уровень между классами (пока или вообще), то интерфейсы же тогда неприменимы (пока или вообще)? + Сейчас слово интерфейсы разработчики документации иногда понимают как: • Нет у нас GUI – нет и интерфейсов (пользовательского); • Нет у нас API - нет и интерфейса (интерфейса API).

1 https://github.com/DenisPolagaev/Java-Theory/blob/master/OOP.md#Расскажите-про-основные-понятия-ООП-класс-объект-интерфейс
2 https://openu.ru/Books/UML/Interface.asp

Раздел	Неприменимость	Может ли поменяться в будущих релизах
Логическая модель базы данных (db-logic-model)	**Возможна**	Пока неприменимость покрывается фразами-заглушками, которые необходимо перенести в документ о неприменимости. Самое простое обоснование: «Этот раздел не применим к данному компоненту, поскольку компонент Сервис трассировки функционирует без обращения к базам данных». Это некое зафиксированное текущее архитектурное состояние системы и раздел может быть неприменим на текущий момент, однако в теории компонент может нарастить функциональность, начать выполнять часть задач самостоятельно (например, решит выполнять аутентификацию самостоятельно или хранить сертификаты у себя) и тогда ему может потребоваться хранилище в будущем. То есть применимость тут оценивается относительно текущей архитектуры.

Раздел	Неприменимость	Может ли поменяться в будущих релизах
Физическая модель базы данных (db-physical-model)	**Возможна**	Пока неприменимость покрывается фразами-заглушками, которые необходимо перенести в документ о неприменимости. Самое простое обоснование: «Этот раздел не применим к данному компоненту, поскольку компонент Сервис трассировки функционирует без обращения к базам данных». Это некое зафиксированное текущее архитектурное состояние системы и раздел может быть неприменим на текущий момент, однако в теории компонент может нарастить функциональность, начать выполнять часть задач самостоятельно (например, решит выполнять аутентификацию самостоятельно или хранить сертификаты у себя) и тогда ему может потребоваться хранилище в будущем. То есть применимость тут оценивается относительно текущей архитектуры.

Раздел	Неприменимость	Может ли поменяться в будущих релизах
Элементы развертывания (deployment-elements)	**Возможна.** Неприменимость возможна для библиотек.	Применимость/ неприменимость этого раздела можно определить на уровне первого релиза и вряд архитектурно возможно изменения для компонента, ибо компонент либо библиотека и не разворачивается сейчас и никогда, либо он — не библиотека и может разворачиваться и сейчас, и потом. Пример обоснования, которое приняли: Поскольку компонент реализован в виде набора Java-библиотек, развертывание не требуется.
Диаграммы развертывания (deployment-diagrams)	**Возможна.** Неприменимость возможна для библиотек.	Поскольку компонент реализован в виде набора Java-библиотек, развертывание не требуется. Развертывание конечного продукта, использующего компонент, определяется разработчиком этого продукта и будет зависеть от его реализации.

Раздел	Неприменимость	Может ли поменяться в будущих релизах
Взаимодействия (interactions)	**Невозможна.** По сути раздел представляет собой текстовую легенду для диаграммы последовательностей.	В стандарте для раздела с взаимодействиями зафиксировано, что взаимодействия должны быть представлены в виде таблицы, со следующими столбцами: • Потребитель — актор или компонент программного компонента — инициатор взаимодействия. • Поставщик — ссылка на API актора или название интерфейса компонента. • Момент взаимодействия — старт приложения, выполнение приложения, оба. • Протокол передачи данных — HTTP, UDP, FTP и др. • Технология взаимодействия — REST, SOAP и др. • Протокол шифрования — TLS, SSL и др., если это может быть указано. • Электронная подпись — используется (с указанием типа) или не используется для передаваемых сообщений.

Раздел	Неприменимость	Может ли поменяться в будущих релизах
		Однако в таких формулировках команды могут рассматривать и таблицу взамодействий, и диаграммы последовательностей как разделы, которые относится только к сетевым взаимодействиям.
Диаграммы последовательностей (sequence-diagram)	**Невозможна.** Диаграмма последовательности отражает поток событий, происходящих в рамках варианта использования. Покуда существует компонент, у него есть варианты использования и акторы, а покуда они существуют, то между ними происходят взаимодействия в определенной последовательности и передача сообщений, под которыми, например, подразумевается: • запрос у объекта-получателя на выполнение одной из его операций; • передача некой информации для обновления его состояния; • запрос на выдачу некоторой информации об объекте-получателе; • сообщение, запрашивающее у объекта-получателя выполнение некоторых действий. Подробнее здесь[3].	-

3 https://studfile.net/preview/2806643/page:3/#7

Раздел	Неприменимость	Может ли поменяться в будущих релизах
Механизмы безопасности (security-mechanisms)	**Возможна.** Если нет собственных механизмов безопасности и/или вся необходимая информация более подробно описана в документе «Руководство по безопасности».	Может быть неприменимым разделом в случае, если компонент не реализует собственных механизмов безопасности, а только использует механизмы безопасности, например, платформы Kubernetes. Однако на каком-то этапе развития компонента он может реализовать собственные механизмы безопасности и/или. например, жестко зафиксировать уровень привилегий при доступе в namespaces и это будет специфичным для этого компонента. Тогда раздел станет применимым.
Прочие поведенческие механизмы (other-behavior-mechanisms)	**Возможна.** В разделе должны быть описаны: кеширование и конфигурирование компонентов, обеспечение согласованности данных и прочее.	Та же история в стандарте, что и с нефункциональными требованиями. Под «прочее» можно понимать что угодно, а в компоненте может не быть перечисленных ниже механизмов: • кеширования; • конфигурирования; • согласованности данных. Однако они могут появиться в будущем.
Варианты работы в различном окружении	**Возможна.** Привязано к диаграмме и элементам развертывания. Если вариант развертывания только один, то вариант окружения только один, а значит раздел неприменим.	Станет применимым, когда вариантов развертывания станет несколько.

Раздел	Неприменимость	Может ли поменяться в будущих релизах
Сценарии отказа	***Возможна.*** Стандарт предусматривает, что сценарии отказа существуют для компонента программного компонента или внешней системы, или внешнего компонента платформы, от которой зависит программный компонент.	В такой формулировке стандарта не предполагается описывать у компонента сценарий отказа его самого, а только его составных частей и внешних зависимостей. Тогда, если на каком-то этапе своего развития компонент односоставен (не состоит из множества частей) и/или не имеет внешних зависимостей, то раздел для него неприменим. Однако в будущем он может стать многосоставным или получить внешние зависимости.
Системное программное обеспечение	**Возможна.** Системное программное обеспечение, необходимое для функционирования компонента, существует в любом случае, однако сейчас этот раздел целиком состоит из ссылки на «Руководство по установке» с разделом «Системное программное обеспечение».	-