

Практика 7: архитектурная документация

Design Documents

Юрий Литвинов
yurii.litvinov@gmail.com

28.02.2022

Design Document, что это и зачем

- ▶ Основной продукт работы архитектора
- ▶ Представляет и объясняет основные принятые архитектурные решения
 - ▶ НЕ набор UML-диаграмм
- ▶ Решение проблемы “Architecture By Implication”
- ▶ Наличие хорошего диздока может сократить затраты на кодирование
 - ▶ в разы

Design Document, что это и зачем

- ▶ В индустриальной практике он часто неформально, но обязательно присутствует — как правило, это набор вики-страниц, иногда встречаются формальные документы
- ▶ Чеклист, позволяющий проверить, что обо всём подумали и всё служит какой-то осмысленной цели
- ▶ Не путать с Game Design Document (хотя он служит тем же целям и очень похож по структуре)

Как писать

- ▶ Достаточно подробно, чтобы при программировании не требовалось принимать важных архитектурных решений
- ▶ Разные *точки зрения*, предназначенные для разных аудиторий
 - ▶ Даже для одной целевой аудитории используется несколько точек зрения, например, статическая структура, поведение, схема БД и требования
- ▶ Рекомендуется использовать диаграммы для иллюстрации архитектуры

Как писать

- ▶ Должен документировать не только принятые решения, но и:
 - ▶ Альтернативы
 - ▶ Чётко формулировать, что в итоге решили
 - ▶ Преимущества принятого решения
 - ▶ Риски
 - ▶ Связь с требованиями
- ▶ Должны быть *полнота* и *консистентность*
- ▶ Стандарты IEEE 1016-2009 и ISO/IEC/IEEE 42010:2011

Типичное содержание документа

- ▶ Различная служебная информация
- ▶ Общие сведения о системе (несколько абзацев)
 - ▶ Назначение
 - ▶ Границы системы (Scope)
 - ▶ Контекст, в котором существует система
- ▶ Architectural drivers

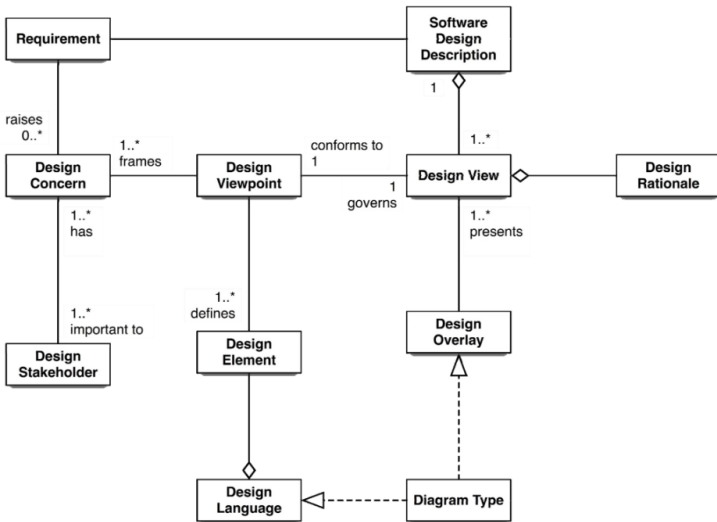
Architectural drivers

- ▶ Architectural drivers — ключевые требования, определяющие архитектуру
- ▶ Бывают:
 - ▶ Технические ограничения
 - ▶ Бизнес-ограничения
 - ▶ Качественные характеристики системы
 - ▶ Сопровождаемость, расширяемость и т.д.
 - ▶ Масштабируемость, производительность
 - ▶ Безопасность
 - ▶ Ключевые функциональные требования

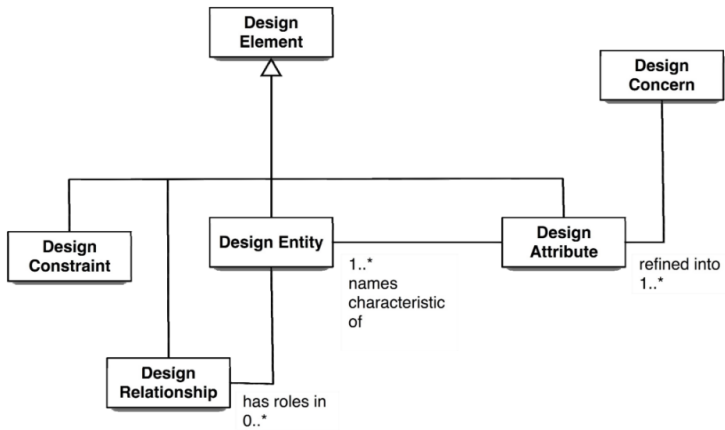
Типичное содержание документа

- ▶ Views (каждый из которых — экземпляр Viewpoint-a)
 - ▶ Требования, роли и случаи использования
 - ▶ Структура системы
 - ▶ Поведение системы
 - ▶ Структура данных
 - ▶ ...
- ▶ Причины принятых решений, за/против
 - ▶ Эта информация обычно приводится и во viewpoint-ax, тут summary

IEEE 1016, концепции документа



IEEE 1016, элементы архитектуры



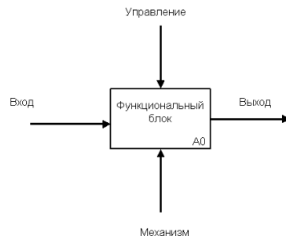
IEEE 1016, точки зрения

- ▶ Всего выделено 12 точек зрения
 - ▶ Контекст
 - ▶ Композиция
 - ▶ Логическая структура
 - ▶ Зависимости
 - ▶ Информационная структура
 - ▶ Использование шаблонов
 - ▶ Интерфейсы
 - ▶ Структура системы
 - ▶ Взаимодействия
 - ▶ Динамика состояний
 - ▶ Алгоритмы
 - ▶ Ресурсы
- ▶ Все точки зрения в документе не обязательны
 - ▶ Тем не менее, есть требование полноты
- ▶ Есть ещё overlays — виды с дополнительной информацией

Контекст системы

- ▶ Назначение — описывает, что система должна делать, фиксирует окружение системы. Состоит из сервисов и акторов, которые могут быть связаны информационными потоками. Система представляет собой “чёрный ящик”
 - ▶ Может быть определён Deployment overlay
 - ▶ Может быть отдельным видом, если аппаратное обеспечение — часть разработки
- ▶ Соображения — функциональные требования, роли, границы системы
 - ▶ Корень иерархии уточняющих дизайн системы видов, стартовая точка при проектировании системы
- ▶ Типичные языки — диаграмма активностей UML, IDEF0 (SADT)

IDEF0, пример диаграммы верхнего уровня



Композиция

- ▶ Назначение — на самом деле, “декомпозиция”, описывает крупные части системы и их предназначение
- ▶ Соображения — локализация и распределение функциональности системы по её структурным элементам, impact analysis, переиспользование (в том числе, покупка компонентов), оценка, планирование, управление проектом, инструментальная поддержка (репозитории, трекер и т.д.)
- ▶ Типичные языки — диаграммы компонентов UML, IDEF0

Логическая структура

- ▶ Назначение — структура системы в терминах классов, интерфейсов и отношений между ними
 - ▶ Используются также примеры экземпляров классов для пояснения решений
- ▶ Соображения — разработка и переиспользование
 - ▶ Разделение на то, что можно взять и приспособить, и то, что придётся написать
- ▶ Типичные языки — диаграммы классов UML, диаграммы объектов UML

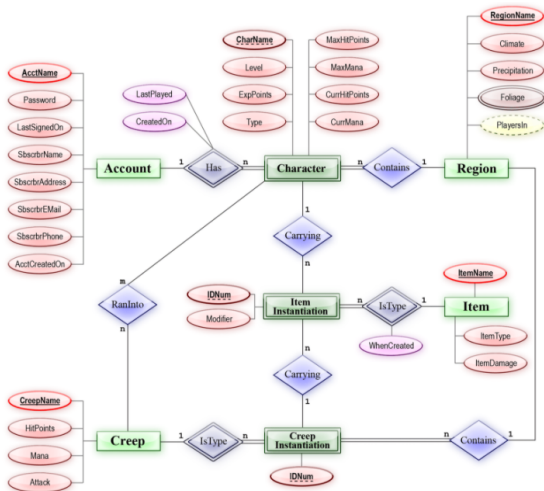
Зависимости

- ▶ Назначение — определяет связи по данным между элементами
 - ▶ Разделяемая между элементами информация, порядок выполнения и т.д.
- ▶ Соображения — анализ изменений, идентификация узких мест производительности, планирование, интеграционное тестирование
- ▶ Типичные языки — диаграммы компонентов UML, диаграммы пакетов UML

Информационная структура

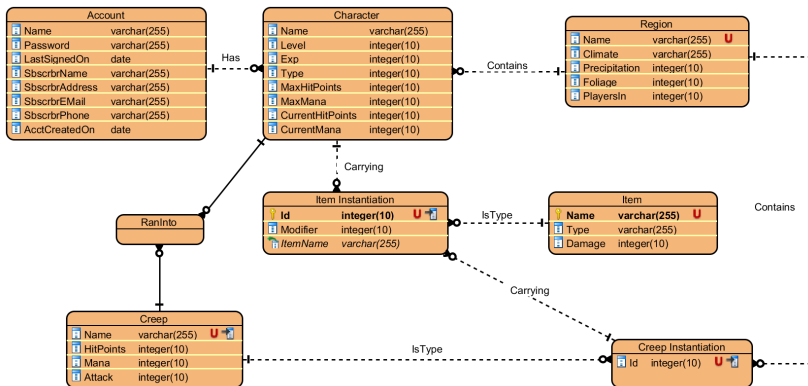
- ▶ Назначение — определяет персистентные данные в системе
- ▶ Соображения — информация, которую требуется хранить, схема БД, доступ к данным
- ▶ Типичные языки — диаграммы классов UML, IDEF1x, ER, ORM

ER, нотация Чена

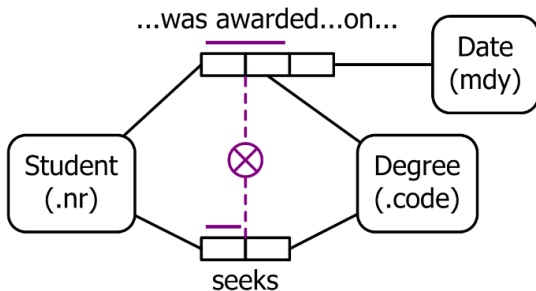


© <https://ru.wikipedia.org>

ER, нотация “вороньей лапки”



ORM, пример диаграммы



Использование шаблонов

- ▶ Назначение — документирование использования локальных паттернов проектирования
- ▶ Соображения — переиспользование на уровне идей и архитектурных стилей
- ▶ Типичные языки — диаграммы классов UML, диаграммы пакетов UML, диаграммы коллабораций UML

Интерфейсы

- ▶ Назначение — специфицирует информацию о внешних и внутренних интерфейсах, не прописанную явно в требованиях
 - ▶ Пользовательский интерфейс рассматривается отдельным видом в рамках этой точки зрения
- ▶ Соображения — договорённости о конкретных схемах взаимодействия компонентов, позволяющие разрабатывать и тестировать их независимо
- ▶ Типичные языки — IDL, диаграммы компонентов UML, макеты пользовательского интерфейса, неформальные описания сценариев использования

Структура системы

- ▶ Назначение — рекурсивное описание внутренней структуры компонентов системы
- ▶ Соображения — структура системы, переиспользование
- ▶ Типичные языки — диаграммы композитных структур UML, диаграммы классов UML, диаграммы пакетов UML

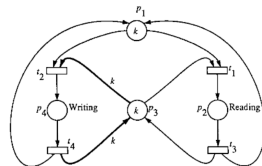
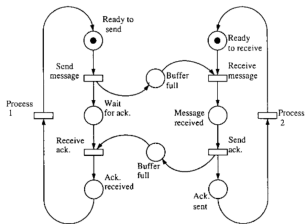
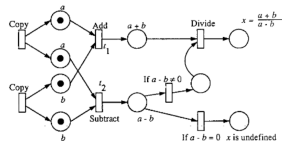
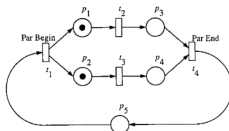
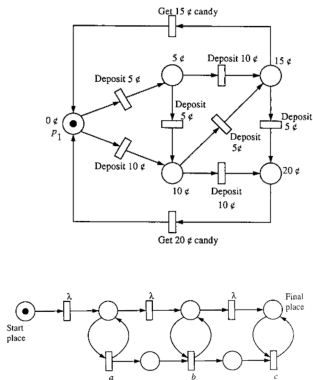
Взаимодействия

- ▶ Назначение — описывает взаимодействие между сущностями: почему когда, как и на каком уровне выполняется взаимодействие
- ▶ Соображения — распределение ответственностей между участниками взаимодействия, определение протоколов взаимодействия
- ▶ Типичные языки — диаграммы композитных структур UML, диаграммы взаимодействия UML, диаграммы последовательностей UML

Динамика состояний

- ▶ Назначение — описание состояний и правил переходов между состояниями в реактивных системах
- ▶ Соображения — поведение системы, включая внутренние состояния, события и логику переходов
- ▶ Типичные языки — диаграммы конечных автоматов UML, диаграммы Харела, сети Петри

Сети Петри, пример



© Murata Tadao. Petri nets: Properties, analysis and applications

Алгоритмы

- ▶ Назначение — описывает в деталях поведение каждой сущности, логику работы методов
- ▶ Соображения — анализ эффективности работы программы, реализация, юнит-тестирование
- ▶ Типичные языки — диаграммы активностей UML, псевдокод, настоящие языки программирования

Ресурсы

- ▶ Назначение — описывает использование внешних ресурсов (как правило, аппаратных или третьесторонних сервисов)
- ▶ Соображения — эффективность работы программы, доступность и эффективность использования ресурсов
- ▶ Типичные языки — диаграммы развёртывания UML, диаграммы классов UML, OCL

Примеры

- ▶ Формальные документы:

- ▶ http://robotics.ee.uwa.edu.au/courses/design/examples/example_design.pdf
- ▶ <https://arxiv.org/ftp/arxiv/papers/1005/1005.0595.pdf>

- ▶ Неформальные документы:

- ▶ <https://github.com/dotnet/efcore/blob/master/docs/security.md>
- ▶ <https://github.com/fsharp/fslang-suggestions/>

Домашнее задание: Roguelike

- ▶ Жанр компьютерных игр, назван в честь игры Rogue, 1980 года выхода
- ▶ Характеризуется:
 - ▶ Простой тайловой или консольной графикой
 - ▶ Активным использованием случайной генерации
 - ▶ Перманентной смертью персонажа и невозможностью загрузить предыдущее сохранение
 - ▶ Чрезвычайно развитым набором игровых правил
 - ▶ Высокой свободой действий персонажа (“игры-песочницы”)
- ▶ Примеры:
 - ▶ <https://en.wikipedia.org/wiki/NetHack>
 - ▶ [https://en.wikipedia.org/wiki/Angband_\(video_game\)](https://en.wikipedia.org/wiki/Angband_(video_game))
 - ▶ https://en.wikipedia.org/wiki/Ancient_Domains_of_Mystery

Функциональные требования

- ▶ Персонаж игрока, способный перемещаться по карте, управляемый с клавиатуры
 - ▶ Карта обычно генерируется, но для некоторых уровней грузится из файла
 - ▶ Характеристики — здоровье, сила атаки и т.д.
 - ▶ Экспа и уровни персонажа, с ростом уровня повышаются характеристики
- ▶ Инвентарь персонажа, включающий элементы, влияющие на его характеристики, которые можно надеть и снять
- ▶ Несколько разных видов мобов, способных перемещаться по карте
- ▶ Боевая система — движущиеся объекты, пытающиеся занять одну клетку карты, атакуют друг друга

Домашнее задание

Разделиться на команды по три человека и написать архитектурное описание Roguelike

- ▶ Общие сведения о системе
- ▶ Architectural drivers
- ▶ Роли и случаи использования
 - ▶ Описание типичного пользователя, как было в курсе по SE
- ▶ Композиция (диаграмма компонентов)
- ▶ Логическая структура (диаграмма классов)
- ▶ Взаимодействия и состояния (диаграммы последовательностей и конечных автоматов)