Software Engineering Conference Russia
14-15 ноября, 2019. Санкт-Петербург

Микросервисные архитектуры с позиции инженерии систем

Роман Цирульников


romkavt@yandex.ru


@romanvt

Яндекс Деньги

Сложность систем нарастает

http://www.mutlulukbedava.com/UserFiles/ArticleFiles/ana-slider-ve-default-pop-up-icerigi6681747.jpg

Признаки недоброго

- Реализация заказанной функциональности требует доработки большого количества сервисов
- Зависимости релизов компонент
- План проекта содержит множество зависимостей, синхронизаций работы людей/команд
- Непрозрачная ответственность за сервисы

Distributed Monolith?

- Декомпозиция на сервисы проведена по неверным критериям, не соответствует прикладным доменам
- Сервисы зависимы друг от друга
- Сервисы обладают несвойственными их домену данными
- Многократное усложнение системы без обещанных выгод MSA

https://www.lpaneque.com/2018/09/the-distributed-monolith-antipattern.html
https://www.bankinghub.eu/banking/research-markets/complexity-kills-european-banking-models-change-complex-world

CRUD service / Entity storage?

- Сервис предоставляет операции Create/Read/Update/Delete над набором сущностей
- Прикладная логика оказывается на стороне клиента и множится по различным клиентам
- REST API — это не CRUD
- Любая информация может быть представлена в виде ресурса, в том числе поведение

Roy Thomas Fielding's dissertation: Architectural Styles and the Design of Network-based Software Architectures, ch.5
https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Frontend, управляющий логикой?

- Прикладной процесс собирается из набора сервисов на уровне интерфейса пользователя
- Система становится вещью в себе, с единственно возможным интерфейсом пользователя для одного класса пользователей
- Всегда должен присутствовать оркестратор прикладного процесса, в явном виде, вне интерфейса пользователя

Особенности сервисных архитектур

(Микро)сервисная архитектура — это **система взаимодействующих** сервисов

- Обладая знанием лишь об устройстве отдельных сервисов, невозможно определить поведение системы в целом
- Прикладной процесс становится надкомпонентной сущностью, невидимой «в коде»

Системы и их свойства

https://www.pinterest.ru/pin/484981453616311977/

Система

Комбинация взаимодействующих элементов, организованная для достижения определенных целей

https://www.sebokwiki.org/wiki/Introduction_to_Systems_Engineering

ISO/IEC/IEEE 15288:2015, ГОСТ Р 57193-2016 Systems and software engineering — System life cycle processes

Система

Комбинация взаимодействующих элементов, организованная для достижения определенных целей

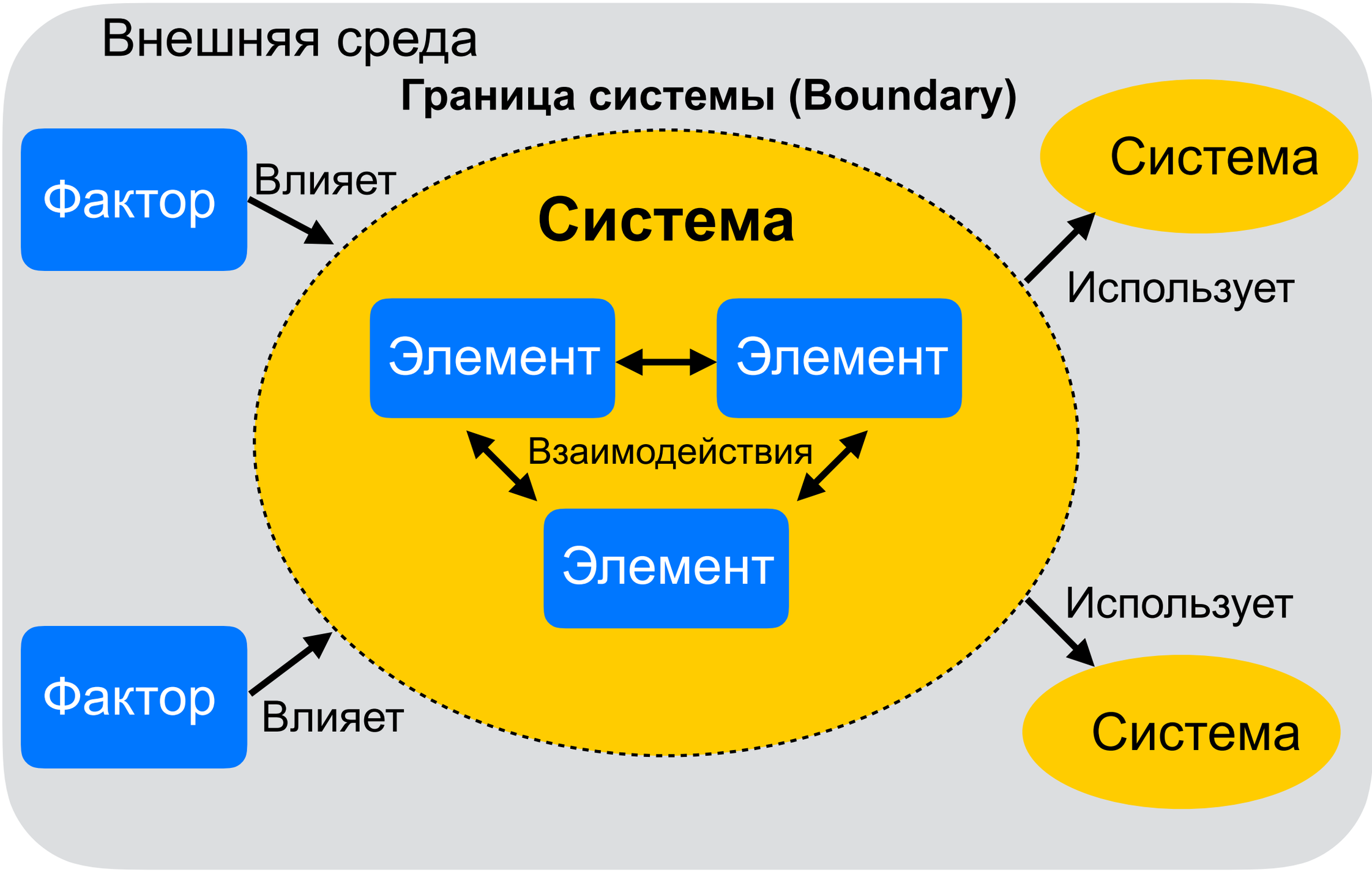
- Элементами системы могут быть: ПО, оборудование, люди, оргструктуры, информация, инфраструктура

- Система имеет границы и взаимодействует с внешним окружением (средой), в том числе с другими системами

https://www.sebokwiki.org/wiki/Introduction_to_Systems_Engineering

ISO/IEC/IEEE 15288:2015, ГОСТ Р 57193-2016 Systems and software engineering — System life cycle processes

System of Interest (SoI)



Свойства систем: сложность

> **Любая система достигает со временем уровня абстракции, недоступного пониманию отдельного индивидуума**

https://en.wikipedia.org/wiki/Software_Peter_principle
https://en.wikipedia.org/wiki/Complex_system

Свойства систем: эмержентность

Система обладает свойствами и поведением, которые отсутствуют у ее элементов в отдельности

Свойства систем: эмержентность

Система обладает свойствами и поведением, которые отсутствуют у ее элементов в отдельности

- Зная лишь свойства и поведение элементов системы, невозможно предсказать поведение системы в целом, также необходимо обладать знанием о совокупности связей элементов системы

Свойства систем: нелинейность

Система может реагировать по-разному на одни и те же внешние воздействия в том же окружении

Свойства систем: нелинейность

Система может реагировать по-разному на одни и те же внешние воздействия в том же окружении

› Не обладая знаниями о всей совокупности связей элементов внутри системы, нельзя предсказать реакцию системы на внешнее воздействие

Свойства систем: деградация

Растущая сложность систем затрудняет дальнейшее их развитие

Свойства систем: деградация

Растущая сложность систем затрудняет дальнейшее их развитие

- Техническая деградация больших систем
- Снижение ожиданий, поиск временных, лёгких решений

Patterns of Systems Thinking: Drift to Low Performance
https://www.sebokwiki.org/wiki/Patterns_of_Systems_Thinking

Архитектура как инженерия систем

- Определение границ систем и подсистем
- Выявление внешних факторов и зависимостей
- Управление взаимодействиями элементов
- Управление взаимодействиями с внешней средой

Структурируем сложность

https://www.pinterest.ru/pin/538391330425398783/?lp=true

Универсальных решений не бывает

Желание использовать одно универсальное решение для различных классов задач в корне неверно

Знакомы ли вам эти слова?

- Stakeholder
- Concern

Определение: Stakeholder

› Заинтересованное лицо

> **Персона или организация, имеющая право, участие, требования или выгоду в системе или обладании ее характеристиками, удовлетворяющими их потребностям и ожиданиям**

ISO/IEC/IEEE 15288:2015, ГОСТ Р 57193-2016 Systems and software engineering — System life cycle processes

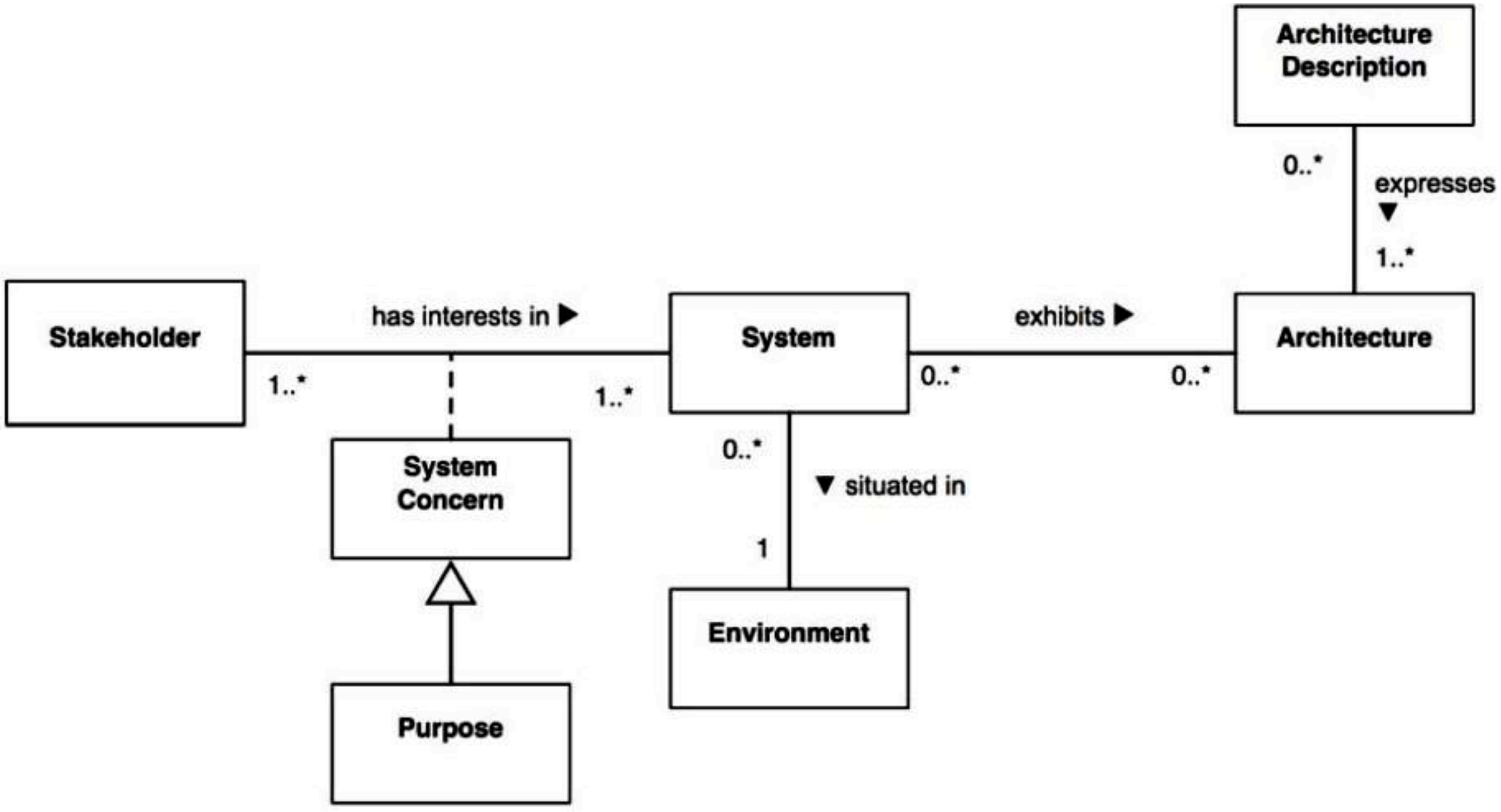
Определение: Concern

- Интерес, проблема, отношение, значимость

> **Интерес может быть выражен как потребности заинтересованных лиц, их цели, ожидания, требования, ограничения, предположения, зависимости, риски по отношению к системе**

ISO/IEC/IEEE 15288:2015, ГОСТ Р 57193-2016 Systems and software engineering — System life cycle processes

Архитектура по ISO/IEC 42010



ISO/IEC 42010:2011, ГОСТ Р 57100-2016 Systems and software engineering — Architecture description

ONE SIZE DOESN'T FIT ALL

https://annejanzer.com/one-size-fits-writer/

Практические наблюдения

- Все усложняется, когда появляется больше одного продукта или класса клиентов
- Разные стейкхолдеры выдвигают противоречивые требования
- Разные продукты требуют различного поведения
- Конфликты интересов

Практические наблюдения

- Все усложняется, когда появляется больше одного продукта или класса клиентов
- Разные стейкхолдеры выдвигают противоречивые требования
- Разные продукты требуют различного поведения
- Конфликты интересов

Решение: принцип Separation of Concerns

Separation of Concerns (SoC)

> **Элементы системы должны реализовывать единственную функцию и иметь единственную область ответственности**

Edsger W. Dijkstra, “On the role of scientific thought”, 1974
http://aspiringcraftsman.com/2008/01/03/art-of-separation-of-concerns/
https://en.wikipedia.org/wiki/Separation_of_concerns
https://www.sebokwiki.org/wiki/Principles_of_Systems_Thinking

Separation of Concerns (SoC)

Элементы системы должны реализовывать единственную функцию и иметь единственную область ответственности

- Никакой элемент не должен разделять свою функцию с другими, равно как и включать в себя постороннюю ответственность

Edsger W. Dijkstra, “On the role of scientific thought”, 1974
http://aspiringcraftsman.com/2008/01/03/art-of-separation-of-concerns/
https://en.wikipedia.org/wiki/Separation_of_concerns
https://www.sebokwiki.org/wiki/Principles_of_Systems_Thinking

Separation of Concerns (SoC)

Элементы системы должны реализовывать единственную функцию и иметь единственную область ответственности

- Никакой элемент не должен разделять свою функцию с другими, равно как и включать в себя постороннюю ответственность

- Крупные задачи решаются эффективнее, при разбиении их на мелкие

Edsger W. Dijkstra, “On the role of scientific thought”, 1974
http://aspiringcraftsman.com/2008/01/03/art-of-separation-of-concerns/
https://en.wikipedia.org/wiki/Separation_of_concerns
https://www.sebokwiki.org/wiki/Principles_of_Systems_Thinking

Separation of Concerns (SoC)

Элементы системы должны реализовывать единственную функцию и иметь единственную область ответственности

- Никакой элемент не должен разделять свою функцию с другими, равно как и включать в себя постороннюю ответственность
- Крупные задачи эффективней решаются путем разбиения их на мелкие

- Критерии декомпозиции: выявление заинтересованных лиц и их интересов

Edsger W. Dijkstra, “On the role of scientific thought”, 1974
http://aspiringcraftsman.com/2008/01/03/art-of-separation-of-concerns/
https://en.wikipedia.org/wiki/Separation_of_concerns
https://www.sebokwiki.org/wiki/Principles_of_Systems_Thinking

Архитектура — это не только технология

https://www.institutefordevelopmentimpact.com/services/organizationaleffectiveness/

Conway’s law

Технологическая декомпозиция всегда воспроизводит структуру коммуникаций организации (орг.структуру)

Melvin Conway, 1967

http://www.melconway.com/Home/Conways_Law.html

Conway’s law

- Организационная структура предприятия должна выстраиваться в соответствии с запланированной архитектурой
- Технологическая архитектура и организационная структура неразделимы
- Не допускайте пересечений интересов подразделений в одном техническом компоненте, он будет «ничей»

https://www.thoughtworks.com/radar/techniques/inverse-conway-maneuver

Холистический подход

Акцент на понимании бизнес-процесса в целом, а не отдельных технологических аспектов

- Система должна рассматриваться как единое целое, а не набор деталей
- Изучайте предметную область, оперируйте терминами предметной области

https://www.sebokwiki.org/wiki/Principles_of_Systems_Thinking

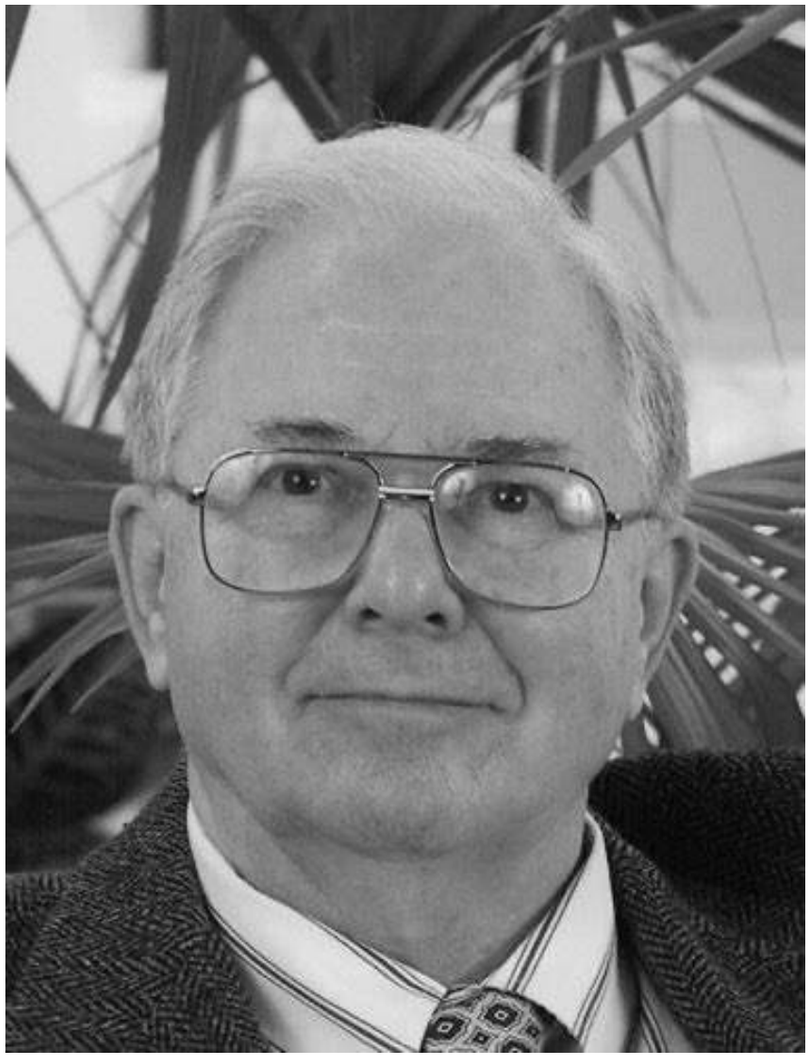
Legacy-системы

https://www.outsideonline.com/2347674/dinosaur-artist-jorge-gutierrez-sculpture-poached-rachel-love-nuwer

Legacy-системы

> **Legacy неизбежно, система поступательно развивается из чего-то ранее существовавшего**

Gall’s law



Сложная система, созданная с нуля, никогда не заработает. Начинать надо с работающей простой системы.

John Gall, Systemantics: How Systems Really Work and How They Fail, 1977

Gall’s law

- Сложные системы, созданные с нуля, не проходили через естественный отбор под воздействием факторов окружения

- Невозможно заранее обладать знанием всех факторов окружения и связей в системе, вы будете постоянно сталкиваться с различными непредвиденными проблемами

- Предыдущая версия системы уже прошла проверку на состоятельность

Legacy-системы

- Legacy неизбежно, система поступательно развивается из чего-то ранее существовавшего
- Окружение определяет применимость решений
- Окружение меняется, реализация остается

Netscape Navigator 6.0

Разработчики совершили худшую из возможных ошибок в сфере программного обеспечения:

> **Они решили переписать все с нуля**

- Выбрасывая работающий код и начиная с нуля, вы теряете накопленные знания и годы работы. Вы подарите вашим конкурентам несколько лет.

- Нет никаких оснований полагать, что [в том же окружении] новая реализация будет лучше, чем старая

https://www.joelonsoftware.com/2000/04/06/things-you-should-never-do-part-i/

Evolutionary Architecture

Эволюционная архитектура руководствуется принципом последовательного проведения серии небольших управляемых изменений в разных аспектах системы

Neal Ford

http://nealford.com/downloads/Evolutionary_Architecture_Keynote_by_Neal_Ford.pdf

Энтропия и построение систем

https://www.tokkoro.com/3099118-bloom_blossom_by-looking_close_common-dandelion_dandelion_filigree_flora_flower_macro_meadow_nature_.html

Закон Е.А. Седова

Только при условии ограничения разнообразия нижележащего уровня можно формировать разнообразные функции и структуры находящихся на более высоких уровнях социальных систем

Е.А Седов: Информационно-энтропийные свойства социальных систем
http://ecsocman.hse.ru/data/149/386/1217/009_SEDOV.pdf
https://traditio.wiki/%D0%97%D0%B0%D0%BA%D0%BE%D0%BD_%D0%A1%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0

Построение сложных систем

Стандартизация интерфейсов взаимодействий неизбежна

- Синергия благодаря стандартизации
- API-driven design сервисов
- Спецификации API для независимого развития сервисов
- Сокрытие реализации , loose-coupling, повторное использование

Мы построим

башню до небес

https://www.wikiwand.com/en/Tower_of_Babel

Second System Effect: IBM OS/360

> **Первый проект системы стремится к скромности ввиду имеющихся ограничений, вторичное откладывается**

Frederick P. Brooks, The Mythical Man-Month: Essays on Software Engineering, 1975

http://wiki.c2.com/?SecondSystemEffect

https://en.wikipedia.org/wiki/Second-system_effect

Second System Effect: IBM OS/360

- Первый проект системы стремится к скромности ввиду имеющихся ограничений, вторичное откладывается

> **Вторая система таит наибольшие опасности: на волне успеха проект перегружен идеями, ожиданиями и украшательствами**

Frederick P. Brooks, The Mythical Man-Month: Essays on Software Engineering, 1975

http://wiki.c2.com/?SecondSystemEffect

https://en.wikipedia.org/wiki/Second-system_effect

Second System Effect: IBM OS/360

- Первый проект системы стремится к скромности ввиду имеющихся ограничений, вторичное откладывается
- Вторая система таит наибольшие опасности: на волне успеха проект перегружен идеями, ожиданиями и украшательствами

При работе над третьей и последующими системами закрепляется опыт в отношении общих характеристик подобных систем

Frederick P. Brooks, The Mythical Man-Month: Essays on Software Engineering, 1975

http://wiki.c2.com/?SecondSystemEffect

https://en.wikipedia.org/wiki/Second-system_effect

Поговорим про «микро»

Micro-Service Architecture

Определяет принципы декомпозиции на сервисы, а не размер отдельного сервиса

- Декомпозиция на сервисы по границам прикладных доменов
- В сервисы выделяются независимые процессы, сущности

https://techbeacon.com/app-dev-testing/forget-monoliths-vs-microservices-cognitive-load-what-matters

Micro-Service Architecture

Сервисы выделяются по мере роста системы, появления новых domains и subdomains

- Начальная декомпозиция на сервисы не нужна, если у вас всего один прикладной домен
- Предвидеть совершенно правильную декомпозицию в будущем невозможно

В организационном аспекте

Границы сервисов проводятся по прикладным доменам в соответствии с компетенциями команд

- Сложность системы структурируется по продуктам и бизнес-процессам
- Критически важно сохранение знаний
- Однозначная матрица ответственности за сервисы
- Независимость и относительная простота сервисов снижает организационные зависимости

https://techbeacon.com/app-dev-testing/forget-monoliths-vs-microservices-cognitive-load-what-matters

В заключение

- Обладая знанием лишь об устройстве отдельных сервисов, невозможно определить поведение системы в целом
- Акцент на понимании бизнес-процесса в целом, а не отдельных технологических аспектов, изучайте предметную область
- Стандартизация интерфейсов взаимодействий неизбежна для построения высокоорганизованных систем
- Окружение непрерывно меняется, воспринимайте мир в динамике

Яндекс Деньги

Благодарю за внимание

https://yadi.sk/i/6Ur0-Oe1QiKZMw

Роман Цирульников

Архитектор ИТ-решений

romkavt@yandex.ru

@romanvt