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

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)

IEEE 1016-2009, 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

IEEE 42010:2011, требования к документации

- Общая информация о документе и о системе
- Стейкхолдеры и их интересы
  - пользователи, операторы, приобретатели, владельцы, поставщики, разработчики, строители, сопровождающие
  - назначение, соответствие архитектуры решаемым задачам, выполнимость разработки и развёртывания, риски и влияние системы на стейкхолдеров, способность к эволюции
- Определение Viewpoint-ов
- Архитектурные виды
- Отношения между элементами архитектуры
- Обоснование архитектуры

Как писать

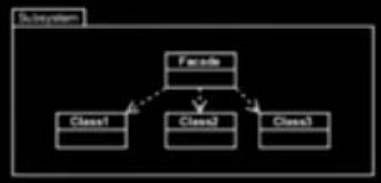
- Должен документировать не только принятые решения, но и:
  - Альтернативы
    - Чётко формулировать, что в итоге решили
  - Преимущества принятого решения
  - Риски
  - Связь с требованиями
- Должны быть *полнота* и *консистентность*
- Стандарты IEEE 1016-2009 и ISO/IEC/IEEE 42010:2011 (он же ГОСТ Р 57100-2016)

Когда применять

- Когда в приложении используется много мелких объектов
- Они допускают разделение состояния на внутреннее и внешнее
  - Внешнее состояние было вычислимо
- Идентичность объектов не важна
  - Используется семантика Value Type
- Главное, когда от такого разделения можно получить ощутимый выигрыш

“Фасад” (Facade), детали реализации

- Абстрактный Facade
  - Существенно снижает связность клиента с подсистемой

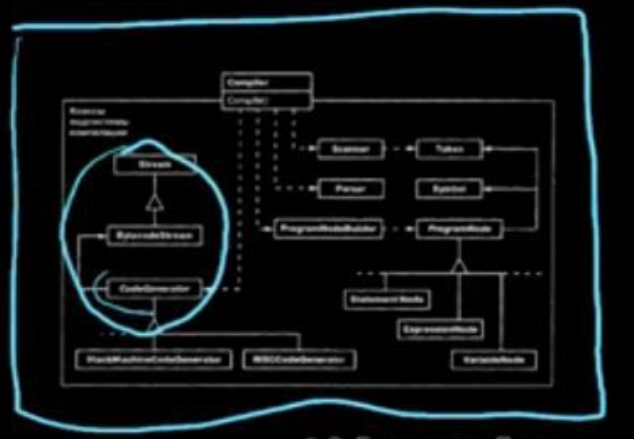


- Открытые и закрытые классы подсистемы
  - Пространства имён и пакеты помогают, но требуют дополнительных соглашений
    - Пространство имён details
  - Инкапсуляция целой подсистемы — это хорошо

Паттерн “Фасад”

Facade

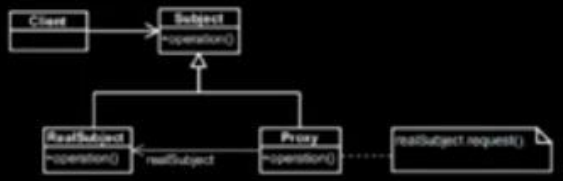
- Простой интерфейс к сложной системе
- Отделение подсистем от клиента и друг от друга
- Многоуровневая архитектура



© Э. Гамма и др., Приемы объектно-ориентированного проектирования

Паттерн “Заместитель”

Proxy



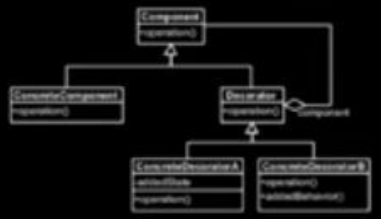
- Замещение удалённых объектов
- Создание “тяжёлых” объектов по требованию
- Контроль доступа
- Умные указатели
  - Подсчёт ссылок
  - Ленивая загрузка/инициализация
  - Работа с блокировками
  - Копирование при записи

“Стратегия” (Strategy), детали реализации (2)

- Стратегия может быть параметром шаблона
  - Если не надо её менять на лету
  - Не надо абстрактного класса и нет оверхеда на вызов виртуальных методов
- Стратегия по умолчанию
  - Или просто поведение по умолчанию, если стратегия не установлена
- Объект-стратегия может быть приспособленцем

“Декоратор” (Decorator), детали реализации

- Интерфейс декоратора должен соответствовать интерфейсу декорируемого объекта
  - Иначе получится “Адаптер”
- Если конкретный декоратор один, абстрактный класс можно не делать
  - Component должен быть по возможности небольшим (в идеале, интерфейсом)
    - Иначе лучше паттерн “Стратегия”
    - Или самодельный аналог, например, список “расширений”, которые вызываются декорируемым объектом вручную перед операцией или после неё



“Компоновщик”, детали реализации (2)

- Ссылка на родителя
  - Может быть полезна для простоты обхода
  - “Цепочка обязанностей”
  - Но дополнительный инвариант
  - Обычно реализуется в Component
- Разделяемые поддеревья и листья
  - Позволяют сильно экономить память
  - Проблемы с навигацией к родителям и разделяемым состоянием
  - Паттерн “Приспособленец”
- Порядок потомков может быть важен, может нет
- Кеширование информации для обхода или поиска
  - Например, кеширование ограничивающих прямоугольников для фрагментов картинки
  - Инвалидация кеша
- Удаление потомков
  - Если нет сборки мусора, то лучше в Composite
  - Следует опасаться разделяемых листьев/поддеревьев

Начнём с примера

Текстовый редактор

WYSIWYG-редактор, основные вопросы:

- Структура документа
- Форматирование
- Создание привлекательного интерфейса пользователя
- Поддержка стандартов внешнего облика программы
- Операции пользователя, undo/redo
- Проверка правописания и расстановка переносов

Паттерны проектирования

Шаблон проектирования — это повторимая архитектурная конструкция, являющаяся решением некоторой типичной технической проблемы

- Подходит для класса проблем
- Обеспечивает переиспользуемость знаний
- Позволяет унифицировать терминологию
- В удобной для изучения форме
- НЕ конкретный рецепт или указания к действию

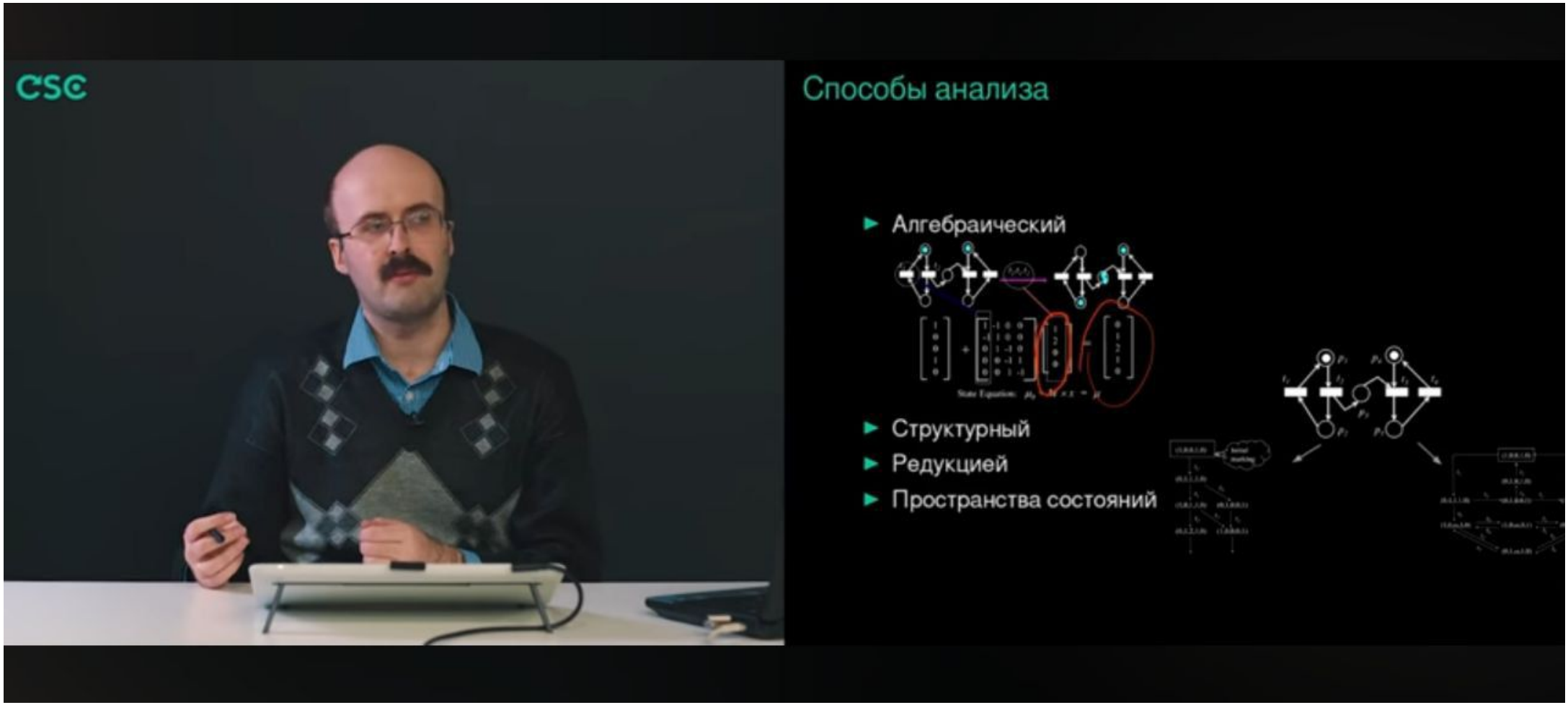

CSC
Способы анализа
Алгебраический
Структурный
Редукцией
Пространства состояний

Диаграмма развёртывания UML

- Показывает отображение компонентов и физических артефактов на реальные (или виртуальные) устройства
- Бывает полезна на начальных этапах проектирования, даже до диаграмм компонентов

© М. Фаулер, UML. Основы

Диаграммы конечных автоматов

Диаграммы состояний

- Состояния объекта как часть жизненного цикла
- Моделирование реактивных объектов
  - Например, сетевое соединение
  - Или знакомый пример с торговым автоматом
- Имеют исполнимую семантику
- Д. Харел, 1987

Свойства, которые можно проверить

- Поведенческие свойства:
  - Достижимость
  - Ограниченность (безопасность)
  - Живость (L0 - L4)
  - “Реверсабельность” и “домашнее состояние”
  - ...
- Структурные свойства
  - Структурная живость
  - Полная контролируемость
  - Структурная ограниченность
  - ...

Моделирование требований

Первый этап разработки любой системы — сбор и анализ требований

- Понимание разработчиками решаемой задачи
- Соглашение между разработчиками, заказчиками и пользователями
  - Заказчики и пользователи часто разные люди с разными потребностями
- Чёткое обозначение границ системы
- Основа для планирования проекта
- Чаще всего словесное описание требований, реже формальные модели

Синтаксис свойств

Yurii Litvinov

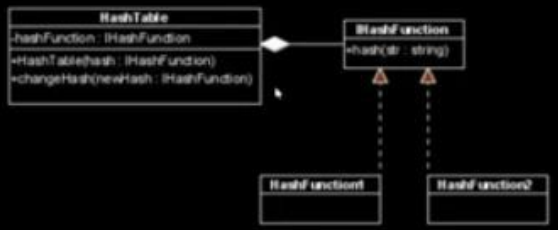
- Объявление поля:
  - видимость имя: тип кратность = значение по умолчанию {строка свойств}
- Видимость:
  - + (public), – (private), # (protected), ~(package)
- Кратность:
  - 1 (ровно 1 объект), 0..1 (ни одного или один), * (сколько угодно), 1..*, 2..*

CSC

Вернёмся к визуальным моделям

Yuri Litvinov

- **Метафора визуализации** — договорённость о том, как будут представляться сущности языка
- **Точка зрения моделирования** — какой аспект системы и для кого моделируется
- Бывают одноразовые модели, документация и графические исходники
  - **Семантический разрыв** — неспособность модели полностью специфицировать систему

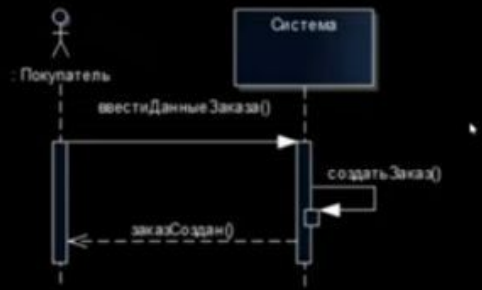


CSC

Возможные преимущества моделей

Yurii Litvinov

- Инструмент, направляющий и облегчающий проектирование
- Средство коммуникации между разработчиками
- Наглядный инструмент для общения с заказчиком
- Средство документирования и фиксации принятых решений
- Исходник для генерации кода?



CSC

Interface segregation principle

- Клиенты не должны зависеть от методов, которые они не используют
  - Слишком “толстые” интерфейсы необходимо разделять на более мелкие и специфические

Liskov substitution principle

- Функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа, не зная об этом

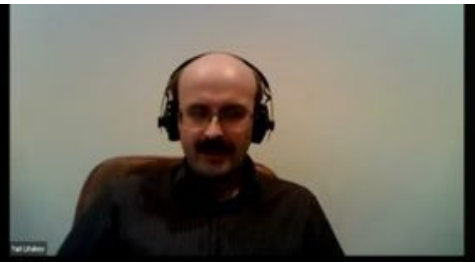
CSC

Single responsibility principle

- Каждый объект должен иметь одну обязанность
- Эта обязанность должна быть полностью инкапсулирована в объект

CSC

Изоляция служебной функциональности



- Служебная функциональность может быть инкапсулирована
  - Репозитории
  - Фабрики
  - Диспетчеры, медиаторы
  - Статические классы (*Сервисы*)
  - ...

CSC

Изоляция возможных изменений

- Потенциальные изменения могут быть инкапсулированы
- Источники изменений
  - Бизнес-правила
  - Зависимости от оборудования и операционной системы
  - Ввод-вывод
  - Нестандартные возможности языка
  - Сложные аспекты проектирования и конструирования
  - Третьесторонние компоненты
  - ...

CSC

Изоляция сложности

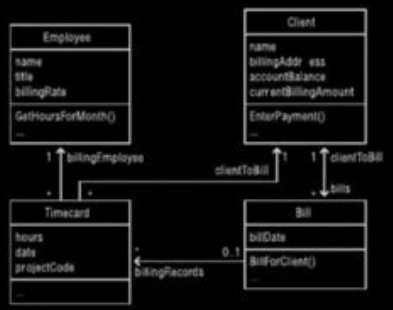
- Сложные алгоритмы могут быть инкапсулированы
- Сложные структуры данных — тоже
- И даже сложные подсистемы
- Надо внимательно следить за интерфейсами

CSC

Определение объектов реального мира

Объектная модель предметной области

- Определение объектов и их атрибутов
- Определение действий, которые могут быть выполнены над каждым объектом (назначение ответственности)
- Определение связей между объектами
- Определение интерфейса каждого объекта



CSC

Наследование и композиция

- **Наследование**
  - Отношение “Является” (is-a)
  - Способ абстрагирования и классификации
  - Средство обеспечения полиморфизма
- **Композиция**
  - Отношение “Имеет” (has-a)
  - Способ создания динамических связей
  - Средство обеспечения делегирования
- Более-менее взаимозаменяемы
  - Объект-потомок на самом деле включает в себя объект-предок
  - Композиция обычно предпочтительнее

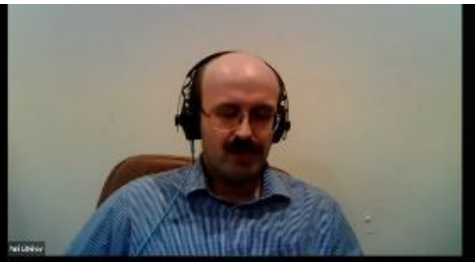
CSC

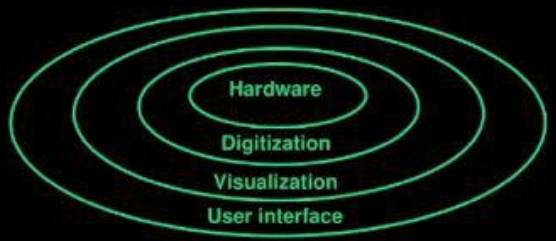
Выводы

- Можем делать утверждения о свойствах системы, базируясь на её структурных свойствах
  - Не написав ни строчки кода и даже не выбрав язык реализации
- Рассуждения очень субъективны
  - Многое зависит от интуиции и вкуса архитектора, однако ошибки очень дороги
- Можно выделить *архитектурные стили* — «архитектуры архитектур»
- Можно выделить *архитектурные точки зрения* и *архитектурные виды*
- Разный уровень детализации

CSC

Вариант 2: слоистая архитектура

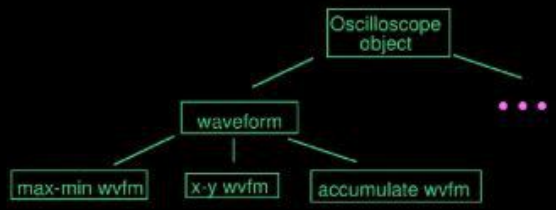




© Garlan D., Shaw M. An introduction to software architecture

CSC

Вариант 1: объектная модель



© Garlan D., Shaw M. An introduction to software architecture

CSC

Пример: ПО для осциллографа

- Считывать параметры сигнала
- Оцифровывать и сохранять их
- Выполнять разные фильтрации и преобразования
- Отображать результаты на экране
  - С тач-скрином и встроенным хелпом
- Возможность настройки под конкретные задачи

© Hantek

По статье Garlan D., Shaw M. An introduction to software architecture

CSC

Архитектор vs разработчик

- Широта знаний
- Коммуникационные навыки
- Часто архитектор играет роль разработчика и наоборот
  - Архитектор в «башне из слоновой кости» — это плохо

CSC

Профессия «Архитектор»

- Архитектор — специально выделенный человек (или группа людей), отвечающий за:
  - разработку и описание архитектуры системы
  - доведение её до всех заинтересованных лиц
  - контроль реализации архитектуры
  - поддержание её актуального состояния по ходу разработки и сопровождения

CSC

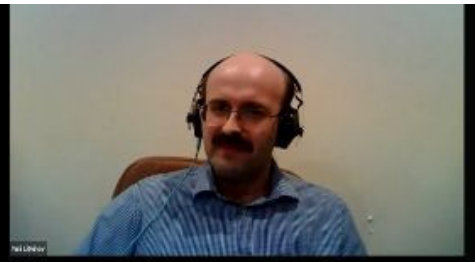
Архитектура

- Совокупность важнейших решений об организации программной системы
  - Эволюционирующий свод знаний
  - Разные точки зрения
  - Разный уровень детализации
- Для чего
  - База для реализации, «фундамент» системы
  - Инструмент для оценки трудоёмкости и отслеживания прогресса
  - Средство обеспечения переиспользования
  - Средство анализа системы ещё до того, как она реализована

© Интернеты

CSC

Размер типичного ПО



Простая игра для iOS	10000 LOC
Unix v1.0 (1971)	10000 LOC
Quake 3 engine	310000 LOC
Windows 3.1 (1992)	2.5M LOC
Linux kernel 2.6.0 (2003)	5.2M LOC
MySQL	12.5M LOC
Microsoft Office (2001)	25M LOC
Microsoft Office (2013)	45M LOC
Microsoft Visual Studio 2012	50M LOC
Windows Vista (2007)	50M LOC
Mac OS X 10.4	86M LOC

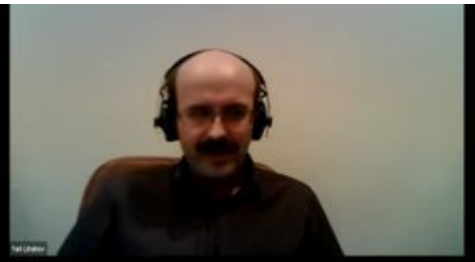
http://www.informationisbeautiful.net/visualizations/million-lines-of-code/

CSC

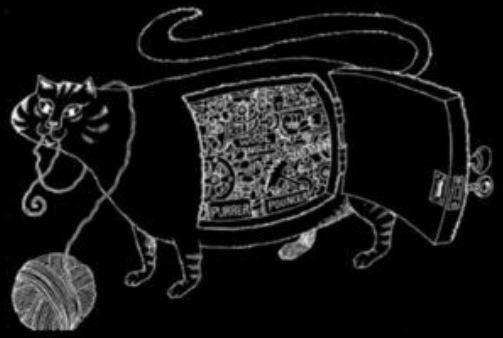
Программа и программный продукт

© Ф. Брукс, «Мифический человеко-месяц»

Инкапсуляция



Инкапсуляция разделяет интерфейс (**контракты**) абстракции и её реализацию
Инкапсуляция защищает **инварианты** абстракции



© G. Booch, “Object-oriented analysis and design”

CSC

Абстракция

Абстракция выделяет существенные характеристики объекта, отличающие его от остальных объектов, с точки зрения наблюдателя

© G. Booch, “Object-oriented analysis and design”

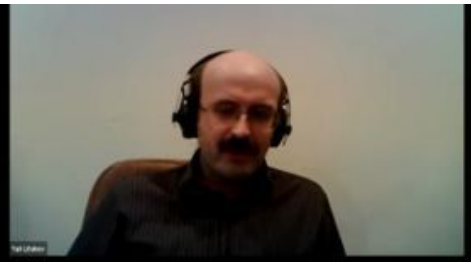
CSC

Объекты

- Имеют
  - Состояние
    - Инвариант
  - Поведение
  - Идентичность
- Взаимодействуют через посылку и приём сообщений
  - Объект вправе сам решить, как обработать вызов метода (**полиморфизм**)
  - Могут существовать в разных потоках
- Как правило, являются экземплярами **классов**

CSC

Сопряжение и связность



- **Сопряжение (Coupling)** — мера того, насколько взаимозависимы разные модули в программе
- **Связность (Cohesion)** — степень, в которой задачи, выполняемые одним модулем, связаны друг с другом
- Цель: слабое сопряжение и сильная связность

CSC

Модульность

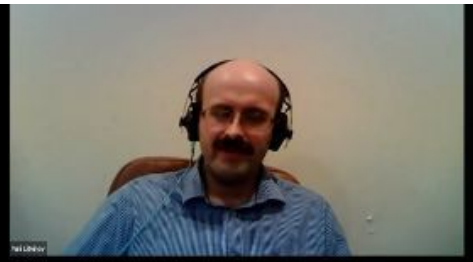
- Разделение системы на компоненты
- Потенциально позволяет создавать сколь угодно сложные системы
- Строгое определение контрактов позволяет разрабатывать независимо
- Необходим баланс между количеством и размером модулей

Архитектура и разработка

- *prescriptive architecture* — архитектура, как её определил архитектор
- *descriptive architecture* — архитектура, как она есть в системе
  - Архитектура у ПО есть всегда, как вес у камня
- *architectural drift* — «сползание» фактической архитектуры
  - появление в ней важных решений, которых нет в описательной архитектуре
- *architectural erosion* — «размывание» архитектуры
  - отклонения от описательной архитектуры, нарушения ограничений
- Антипаттерн «*Big ball of mud*» — результат эрозии

CSC

Архитектура и проектирование — задачи



- Декомпозиция задачи
- Определение границ компонентов
- Определение интерфейсов между компонентами
- Общие для всей системы вопросы
  - Стратегия обработки ошибок
  - Стратегия логирования
  - Стратегия обновлений
  - Стратегия разделения доступа
  - Вопросы локализации
  - ...
- Анализ и верификация архитектуры

CSC

Требования

- Функциональные — то, *что* система должна делать
- Нефункциональные — то, *как* система должна это делать
  - Эффективность
  - Масштабируемость
  - Удобство использования
  - Надёжность
  - Безопасность
  - Сопровождаемость и расширяемость
  - ...
- Ограничения
  - Технические
  - Бизнес-ограничения

CSC

Требования

- Функциональные — то, *что* система должна делать
- Нефункциональные — то, *как* система должна это делать
  - Эффективность
  - Масштабируемость
  - Удобство использования
  - Надёжность
  - Безопасность
  - Сопровождаемость и расширяемость
  - ...
- Ограничения
  - Технические
  - Бизнес-ограничения

CSC