

Разработка мобильных приложений

Лекция 8

Механизмы Android,

а также немного размышлений и мотивации

Ключев А.О. к.т.н., доцент ФПИиКТ Университета ИТМО

Санкт-Петербург

2025

Эпиграф

Солнце - это такая большая светящаяся лампочка на небе, которая греет нас по утрам и помогает расти овощам на грядке. Без него было бы темно и холодно, как ночью зимой, и ничего бы не росло на огороде, а мы бы все замёрзли и не смогли бы сушить бельё на верёвке.

[YandexGPT 5]

Литература

- Роберт Мартин. Чистая архитектура
- Непейвода Н. Н. ,Скопин И. Н. Основания программирования.
- Гради Буч. Объектно-ориентированный анализ и проектирование с примерами приложений на C++
- Непейвода Н.Н. Инфософия. Введение в системный и логический анализ. Курс лекций

Литература

- **Как работает Android (Хабр, блог компании Ростелеком-Солар Разработка под Android)**
 - <https://habr.com/ru/companies/solarsecurity/articles/334796/>
 - <https://habr.com/ru/companies/solarsecurity/articles/338292/>
 - <https://habr.com/ru/companies/solarsecurity/articles/338494/>
 - <https://habr.com/ru/companies/solarsecurity/articles/427431/>
- **Binder: как устроена работа с несколькими процессами в Android**
 - <https://apptractor.ru/info/media/binder-kak-ustroena-rabota-s-neskolkiimi-protsessami-v-android.html>
 - <https://youtu.be/yyaw0C6oA5k>

Текущие наблюдения

- Для многих курс не интересен. Как правило виден крайне прагматичный подход состоящий в изучения минимума материала для получения максимально возможной зарплаты здесь и сейчас. О перспективах никто не думает (особенности поколения?).
 - Практически отсутствуют вопросы.
 - Посещаемость лекций - низкая (не более 25%).
 - Посещаемость практических занятий – низкая (несколько раз я приходил в пустую аудиторию), вопросов почти никто не задает.
- Практически никто не читает предлагаемую литературу. Материалами лекций никто не интересуется (в основном все ссылаются на усталость и высокую нагрузку по работе и учебе). Правильно, «что тут думать, трясти надо» (С) Бородатый анекдот.
- Уровень подготовки подавляющего большинства студентов- посредственный или низкий, при этом многие работают по специальности, но занимаются там довольно примитивной работой.
- Многие студенты работающие по специальности принципиально не хотят отходить от используемого на работе стека технологий и узнавать что-то новое.

Заменяет ли работа по специальности учебу в университете?

- **В большинстве случаев, при работе в обычной коммерческой фирме – нет.** Возможно заменит, если вы будете работать в исследовательской лаборатории, как это делал я в свое время.
- Несомненно, работа по специальности является прекрасным **дополнением** к учебе, она определенно вас развивает. Я регулярно заседаю на предзащитах дипломных работ и сам беру много дипломников, мне прекрасно видно, что вы собой представляете как специалисты.
- Минусы перекоса в сторону одной работы:
 - Работа дает вам лишь **узкий сектор** специфических, сиюминутных знаний и умений с малым сроком жизни в несколько лет.
 - Общая система знаний, необходимая вам для дальнейшего саморазвития и понимания сложных систем которых еще нет, выпадает из вашего учебного процесса и делает вас ущербными специалистами, с крайне **узким кругозором**.
 - Упрощенный и узкий взгляд на мир делает ваши технические решения не всегда адекватными, эффективными и уместными.
 - Не факт, что вам в принципе удастся решить какую либо задачу, за пределом вашего кругозора.
 - Ваш любимый стек технологий, который вы сейчас с таким трудом изучили, через несколько лет исчезнет. Изменятся также и подходы к проектированию, а также многие другие вещи.

У современного высшего образования проблемы

- Система знаний даваемая высшим образованием всегда запаздывает по объективным причинам, особенно в наукоемких областях. Система образования консервативна и инерционна.
- Запаздывание постоянно растет, особенно сильно проявляется запаздывание во время технологических прорывов: цифровая электроника 80-х, персональные компьютеры 90-х, ООП 2000-х, мобильные технологии и облачные вычисления 2010-х, ИИ вот прямо здесь и сейчас.
- Технологии плодятся в геометрической прогрессии и быстро заменяются новыми технологиями. Шанс, что вы к окончанию университета освоите именно то, что вам понадобится на работе ничтожно мал (если быть точнее, ничтожно мал шанс получить требуемые на вашей работе практические навыки).

Можете ли студент траекторию своего обучения?

- В некоторых пределах – определенно да
- Полностью ставить под сомнение необходимость изучения тех или иных предметов – определенно нет.
- Почему? Ответы просты:
 1. Вы еще сами не знаете, что вам будет нужно, а что нет.
 2. Если вы думаете, что знаете что вам нужно, то это широко распространенное заблуждение, см, п1
 3. Вам нужна общая техническая культура, так как вы будущая техническая элита, которая будет вынуждена думать за других, предупреждать и исправлять чужие ошибки, а также настаивать на своих решениях, если вы видите, что ваши сотрудники не правы и их технические решения угробят ваш проект.
 4. Вам нужно тренировать свой мозг, подсовывая ему самые неудобные, сложные, непонятные, а также никому не нужные задачи, например такие как физика, математика или философия. Может быть квантовая физика вам и не пригодится, но думать вам в вашей жизни придется постоянно.
 5. Вам нужно создать в голове более-менее правильную систему технических знаний.
 6. Вам нужно как-то адаптироваться к постоянно меняющемуся миру.
- Если вы хотите изучать только какой-то конкретный и простой стек технологий (например, для создание фронтэнда) и больше ничего, то вы вошли не в ту дверь. Вам нужно было идти на курсы вместо университета. К сожалению, уровень образования многих это уровень ПТУ.

Проблемы младших разработчиков или черный лебедь

Основные последствия развития IT-технологий

(см. книгу Нассима Талеба «Черный лебедь»)



- Бешеный рост количества фреймворков, библиотек и языков программирования, основанных на старых идеях, **которых вы пока не знаете**. Новые названия со старым содержанием. Пример: **оркестрация** и **хореография** в распределенных системах.
- Упрощение фреймворков и языков программирования, а также бурное развитие различных конструкторов и платформ создания ПО без кода **ведет к увеличению конкуренции на рынке труда** со стороны необразованных (и дешевых) новоиспеченных айтишников, прошедших краткосрочные курсы после школы, непрофильных университетов и т.п.
- Все больше увеличивается **риск замены простых разработчиков на ИИ** (посмотрите на возможности ChatGPT и его аналогов, а также средств поддержки программирования, встраиваемые прямо в IDE уже сейчас). Еще пару лет назад ничего подобного не было.
- Возможна **радикальная смена стека технологий**. В ближайшем будущем весьма вероятны новые открытия в вычислительной технике, массовое внедрение которых могут сделать ненужным различные профессии и целые отрасли промышленности, как это уже бывало ранее. Например:
 - Появление АТС сделало ненужными телефонисток
 - Автомобили вытеснили гужевой транспорт
 - Мобильные технологии и Интернет сильно ударили по проводной телефонной связи, ТВ, кинотеатрам, бумажным газетам и книгам, практически уничтожили рынок фотоаппаратов и магнитофонов и вообще очень сильно изменили жизнь людей.
 - Автопилоты могут сделать безработными таксистов и водителей грузовиков, а роботизированные производства и 3D-печать всего, что угодно сделают безработными низкоквалифицированных рабочих.
 - Теоретически возможно изобретение имплантов или биотехнологий существенно улучшающих когнитивные способности головного мозга человека, что приведет в последствии к [технологической сингулярности](#).

Система знаний о вычислительной технике

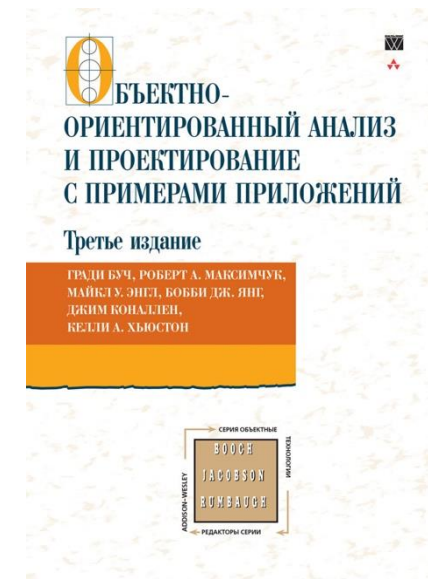
- То, что вы проходите (к сожалению, не все из вас) в университете позволяет вам сформировать систему понятий, входящих в мир вычислительной техники. Это происходит даже если вам предмет совсем не нравится, кажется бесполезным и неприменимым на практике, а лектор не умеет ничего рассказывать.
- Отсутствие некоторых понятий сильно ограничивает ваши когнитивные способности как программистов, умеющих решать поставленные задачи в заданные сроки и с нужным качеством.
- Быстрое решение задачи невозможно, если вам нечем заполнить семантическую пропасть.

Теперь поговорим немного о том, как думают начинающие программисты



Семантическая пропасть

- **Семантическая пропасть** или **семантический разрыв** - разрыв между понятиями языка программирования на котором вы программируете (в нашем случае это Kotlin) и понятиями решаемой задачи. Про это можно почитать у Гради Буча и Роберта Мартина.
- Чем выше уровень языка и чем ближе понятия языка к понятиям задачи, тем пропасть меньше.



Гипотеза лингвистической относительности

- Гипотеза лингвистической относительности (релятивизма) предполагает, что структура языка влияет на мировосприятие и воззрения его носителей, а также на их когнитивные процессы.
- С точки зрения лингвистов эта гипотеза может быть спорной, но они говорят о влиянии естественных языков на когнитивный процесс обычных людей.
- В нашем случае речь идет о влиянии языка программирования или набора инструментальных средств (например, операционной системы и IDE) на когнитивный процесс в голове у программиста.
- Язык программирования это инструмент, придуманный для решения вполне определенного спектра задач. Видимо, если кроме понятий даваемых выбранным языком, в голове у программиста ничего нет, то задача будет решаться в рамках понятий языка программирования и не факт, что семантическую пропасть между языком и задачей удастся чем-то заполнить.
- Теперь представьте, что программист закончил краткосрочные курсы по программированию и в его голове нет понятий, стоящих даже за обычным языком программирования (я молчу про понятия из вычислительной техники и из прикладной области, в которой решается задача).
- К сожалению, с этой проблемой я постоянно сталкиваюсь на практике: **если ты молоток, то все вокруг – гвозди.**

Понимание

- Понимание — универсальная операция мышления, связанная с усвоением нового содержания, **включением его в систему устоявшихся идей и представлений** [Википедия]
- Если понимания предыдущего содержания в голове нет, то понимание чего-то нового в принципе невозможно. С такими пробелами вы должны принимать новую информацию на веру, без возможности как-то ее верифицировать. В принципе, вы так воспринимаете большинство достижений современной науки, но в вашем роде деятельности этот номер не пройдет.
 - Человек из каменного века не поймет вашу речь, даже если выучит ваш язык. В голове человека из прошлого нет тех понятий, которые есть у вас.
 - У вас возникнут те же проблемы понимания при общении с негуманоидными космическими пришельцами (если вы с ними когда либо встретитесь), что хорошо описано в романах Питера Уоттса «Ложная слепота» и «Эхопраксия», в повести «Малыш» братьев Стругацких, а также в фильме «Прибытие» и других художественных произведениях.
- Между мышлением с использованием языка программирования и решаемой задачей есть семантическая (смысловая) пропасть, которую понятиями, имеющимися в языке программирования (например Java), заполнить в принципе невозможно. Чувствуете параллель с вышесказанным? А вы, между прочим, пытаетесь думать над решаемой задачей в рамках языка Java и любимых фреймворков.
- Именно для поэтапного решения этой проблемы придумана уровневая архитектура.
- На каждом уровне, с помощью его инструментов и понятий строится основа для построения нового уровня абстракций, основанного на новых принципах, возможно на принципиально другой модели вычислений.
 - Например, на уровне императивного программирования (машина Фон-Неймана) с помощью механизма прерываний строится основа для создания процессов и у вас появляется возможность программировать с использованием процессов, потоков, корутин и очередей.
 - ООП позволяет вам приближать уровень языка к уровню решаемой задачи путем поэтапного приближения конструкций языка к конструкциям реального мира, путем введения новых типов данных. Класс — это просто новый тип данных. Иерархия классов — классификация (см. онтология, концептуализация, таксономия). Но обычные языки ООП императивные, со всеми вытекающими последствиями (см. «Проблемы с потоками» Эдварда Ли).

Уровни архитектуры, здравый смысл и правота

- Вы можете долго спорить со своими сотрудниками о том или ином техническом решении, но каждый может оказаться прав, может быть крайне логичен и в конце спора остаться при своем мнении.
- Проблема в том, что правота каждого из спорщиков может проистекать из корректных и вполне здравых рассуждений, проведенных на разных уровнях и с учетом разных условий и критериев оценки (подумайте о том, что такое здравый смысл для муравья, лисицы и человека).

Китайская комната

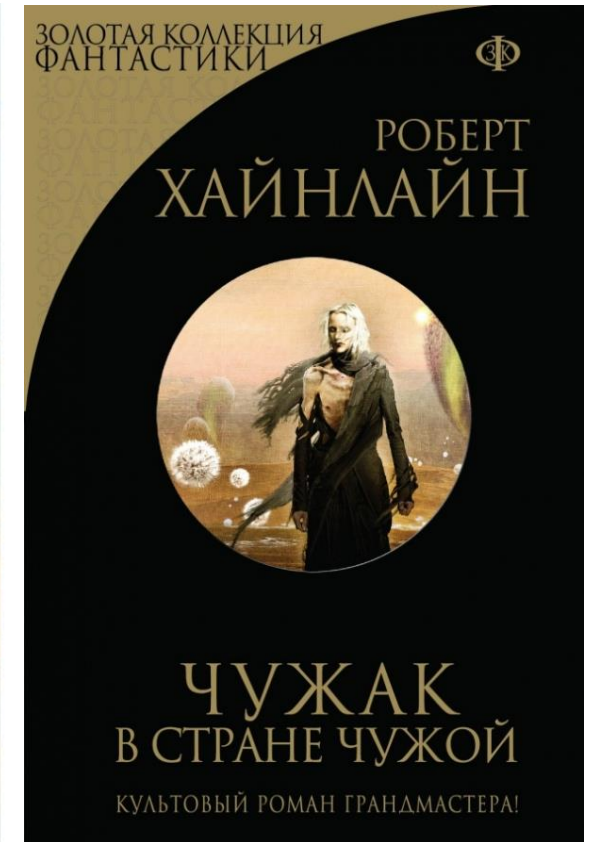
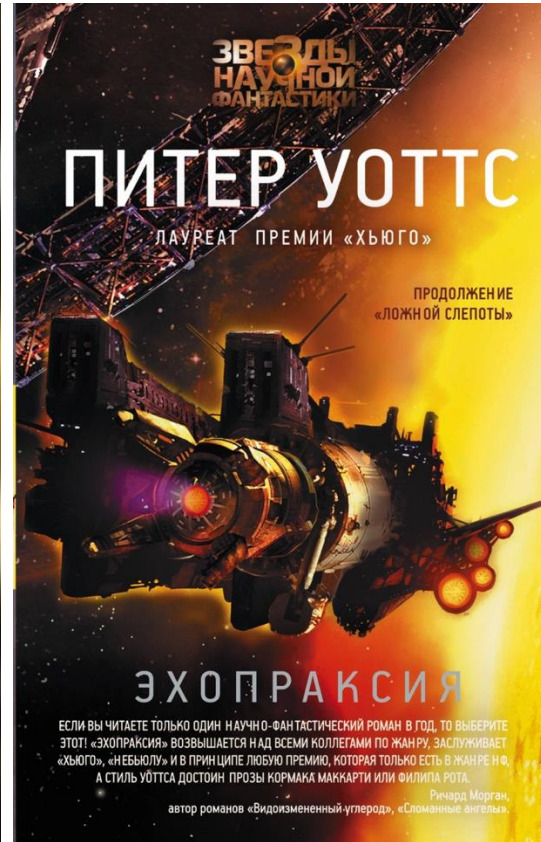
- Эксперимент, в котором человек сидит в изолированной комнате и получает листки с китайскими иероглифами, которых он не понимает.
- Эксперт, знающий китайский язык передает листки с вопросами в комнату.
- Человек внутри комнаты дает ответы, строго по заранее подготовленным инструкциям, которые позволяют давать адекватные ответы на вопросы.
- В результате, человек вообще не понимает что он отвечает, а у стороннего наблюдателя создается впечатление, что он получает ответы от этого человека.
- *Это напоминает процесс программирования, который мы наблюдаем у тех, кто закончил краткосрочные курсы по Web дизайну или тех, кто любит играть с Arduino.*
- Преподавание дает много возможностей увидеть студентов, которые успешно имитируют умственную деятельность. Тоже самое эти студенты скорее всего будут делать и на своем рабочем месте, поэтому доверять кому попало нельзя.

О языке и понимании (для будущих программистов)

М. Хайдеггер. «Из диалога о языке. Между японцем и спрашивающим».

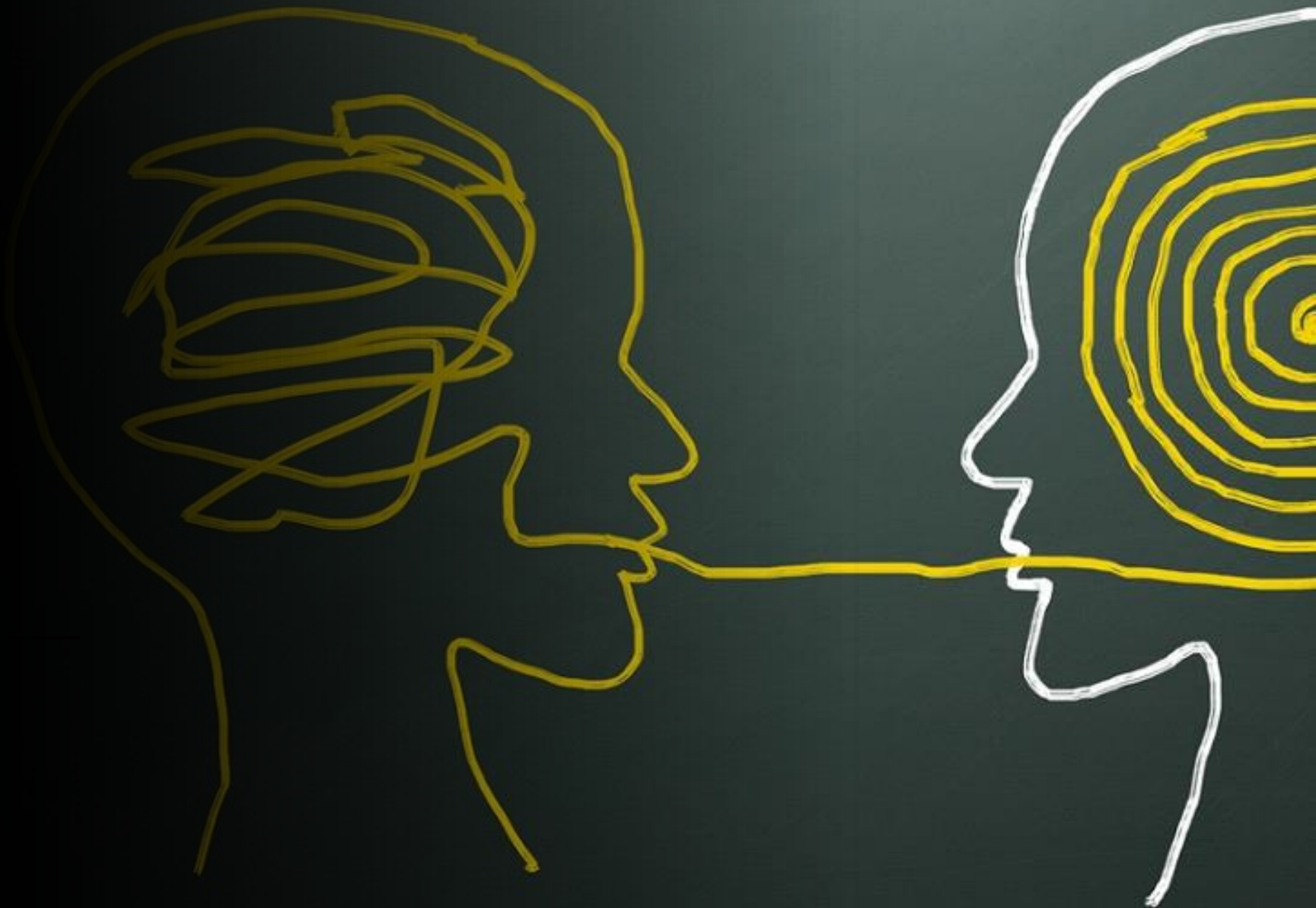
Братья Стругацкие, повесть «Малыш»

Р. Хайнлайн. Чужак в стране чужой и т.д.





Как понять что-либо?



Если жизненный цикл IT-технологий такой короткий, то как жить дальше?

- Нужно пытаться изучать более общие основы, которые не меняются годами.
 - Например, если вы не понимаете как работает какой-либо фреймворк вам нужно осознать на основе каких идей он построен и прежде чем пытаться решать практические задачи, попытаться изучить эти идеи. Иначе вы будете использовать этот фреймворк тем же способом, что и футболист на минном поле.
- В основе современных технологий лежат изобретения, которым может исполниться несколько десятков лет. В основном, мы можем наблюдать сочетание старых идей и их практическое использование.
- Ваша задача состоит не в том, чтобы тренироваться быстро учить новые технологии, а в том, чтобы понять на основе каких идей эти технологии сделаны. Зная основополагающие идеи разобраться с новшествами гораздо проще.
- Нужно смотреть на мир не с позиции молодого (и бездумного) программиста, а пытаться думать как инженер и философ.

Зачем нам онтология?

- Основной вопрос онтологии - «Что существует?»
- *Кстати, UML позволяет изобразить онтологию, но такого рода диаграммы весьма сложны для понимания, хотя они в достаточной степени формальны.*
- Назначение онтологии – концептуализация, то есть описание объектов и связей между ними:
 - Выделение важных объектов
 - Поиск отношений между объектами (см. Г. Буч. Объектно-ориентированный анализ и проектирование)
 - Распознавание функций и ролей объектов
 - Построение мета-моделей
- Очень полезно с помощью приемов онтологии изучать предметную область, архитектуру, строить модели (например, иерархию классов)
- К сожалению, проблемы с концептуальным видением проекта есть не только у будущих бакалавров, но и у магистров и аспирантов. Это объективно сложно, но это важно.

Обобщение и формализация

- Обобщение - поиск закономерностей и каких-то общих свойств, с целью выделения классов сущностей
- Формализация - представление какой-либо содержательной области (рассуждений, доказательств, процедур классификации, поиска информации, научных теорий) в виде формальной системы или исчисления, то есть математически интерпретируемое таким образом, чтобы по мере возможности сохранялись существенные для нынешней цели свойства.

Эмпирические и теоретические знания


- Эмпирические знания получают опытным путем на основе экспериментов и наблюдения за исследуемым объектом.
- Теоретические знания получают на основе систематизации и обобщения знаний полученных эмпирическим путем, содержат в себе понимание внутренних взаимосвязей и концепций.

ИНДУКТИВНЫЙ ПОДХОД

- Индуктивный подход – продвижение от частного к общему. Знания получаются эмпирическим путем, только во время наблюдений за чем либо.
 - Вы смотрите гигабайтные логи или шагаете по строкам своей бескрайней программы и пытаетесь понять, где у вас ошибка .
 - Вы изучаете новый фреймворк, но у вас есть только примеры его использования и нет никакого понимания как он устроен внутри.
- Если у вас нет теории, имеющей такие важнейшие свойства как возможность что-то объяснить и предсказывать, то вы будете очень долго искать ошибки в вашем хтоническом и хаотическом ужасе, который вы называете программой (к вопросу о детерминизме).

Если ли научная составляющая в современных IT-технологиях или насколько все круто?

- Определенно есть, но не там, где вы скорее всего планируете работать. Научная составляющая есть в следующих областях:
 - Создание инструментальных средств (нейросетей, синтезаторов, компиляторов, виртуальных машин, средств отладки как для программного обеспечения, так и для аппаратуры)
 - Разработка новых САПР
 - Создание новых компьютерных архитектур
 - Разработка новых встроенных и киберфизических систем
 - Разработка новых систем на кристалле, процессоров, контроллеров и т.д.
- Где используются технологии, которым 30,40 или 50 лет?
 - В тех местах, где используются давно известные и готовые технологии, фреймворки и инструменты, т.е. на вашей работе. В создании бекэнда, Web-приложения или мобильного приложения нет ничего нового, даже если вы используете самый навороченный, новый, супер-пупер-гипер-мега-модный фреймворк.
 - **Это банальная рутина**, вроде подметания улицы или уборки снега с использованием метлы, лопаты и дворника. Разберите по полочкам ваши инструменты и вы сами это увидите.
 - Основная задача, которую решает IT индустрия в поддержке бизнеса – минимизация затрат на разработку ПО с помощью готового стека технологий при расширении функционала продуктов, требующихся корпоративным пользователям. Минимизация затрат достигается за счет увеличения производительности труда, упрощения порога вхождения в технологический стек для привлечением более дешевой рабочей силы (с меньшим набором знаний, умений и менее развитыми когнитивными возможностями).



Чем отличается человек от обезьяны?



Чем отличается человек от обезьяны?

- Обезьяна умеет использовать инструменты, например, она может шандарахнуть своего оппонента палкой по голове или кинуть в него камень. Обезьяна может даже зубами заострить палку, чтобы было удобнее в чем-то ковыряться. Это уровень маленького человеческого ребенка.
- Человек умеет делать инструменты второго порядка, то есть он умеет делать инструменты для создания других инструментов. В каменном веке люди придумали каменные инструменты, с помощью которых было удобно делать рубила и всякие скребки, а сейчас в современных школах детей учат делать молотки на уроках труда...

Что сложнее?


1. Использовать готовый инструмент или его создавать?
2. Придумывать новый инструмент или придумывать новую технологию?
3. Придумывать технологию или новый метод?



Уровни знаний и умений

- Каждый из этих трех видов деятельности требует разных по сложности подходов к мышлению
- См. уровни знаний и умений, описанные в книге «Основания программирования» Непейводы и Скопина





Чем отличается человек от текущей версии ИИ?



Чем отличается человек от ИИ в наше время?

- Человек умеет придумывать новое и понимать, а языковая модель на базе искусственной нейросети просто предсказывает следующее слово с определенной вероятностью. См. удожественный роман Питера Уотса «Ложная слепота», там про концепцию, а статья на Хабре «[Как работает ChatGPT](#)» уже про конкретику для начинающих.
- Тем не менее ИИ может обобщать массивы информации, то есть работать как поисковик, выдающий на выходе обзоры имеющихся статей и страницы а ля Википедия на заданную тему или решать схожие задачи на базе заложенного в него каталога решений.
- Есть риск, что люди решающие простые задачи, которые не требуют специфических и сложных подходов к мышлению, могут быть заменены таким ИИ уже сейчас.
- Что сдерживает развитие ИИ:
 - ИИ требуются колоссальные вычислительные ресурсы
 - Использование ИИ пока довольно дорого. Классические компьютеры на базе машины Фон-Неймана и использующиеся повсеместно, не подходят для запуска сколько-нибудь сильного ИИ.
 - Недостатки вычислительной мощности приводят к тому, что контекст слишком короткий и ИИ довольно быстро начинает галлюцинировать
- Другими словами у разработчиков «фронтэнда и бекэнда» есть еще пара лет относительно спокойной и безбедной жизни.

Цель, задачи, метод, технология, инструментарий, механизм и всякое такое

Камень, ножницы, бумага, ящерица, Спок

- **Цель** – отвечает на вопрос «зачем»? Например, мы хотим облегчить себе жизнь и придумать какой-то удобный способ крепления досок друг к другу при постройке дома из дерева (ну надоело нам делать шалаши...). Для достижения цели вам нужно понять, какие **задачи** нужно решить.
- **Инструмент** – это технологическая оснастка, которая позволяет решить вам какую-либо задачу. Например, молоток позволяет вам забить гвоздь, чтобы прикрепить одну доску к другой, а клещи позволяют вам этот гвоздь вытащить, если вы его забили не туда.
- Наличие молотка, гвоздей и клещей позволяет вам решить **задачу** – в данном случае вы можете скрепить вместе два куска дерева, а при необходимости разобрать эту конструкцию.
- Практическое решение задачи описанным выше способом – это **технология** соединения деревянных конструкций.
- В основе технологии всегда лежит какая-то идея и научный или практический **метод**, то есть способ (алгоритм) решения задачи, а основе метода из примера лежат такие науки как физика, материаловедение, сопромат и т.д.
- Результатом применения технологии может стать **механизм**, нечто, что должно работать на практике. Механизм позволяет решить какую-то проблему, например скрепить два куска дерева друг с другом.
- **Инструментарий** – это целый комплекс инструментов для какой-либо сложной, комплексной деятельности. Например, для разработки ПО.

Вычислительный механизм *

- Вычислительный механизм – техническое решение, реализующее часть (фрагмент, элемент, «аспектный срез») вычислительного процесса.
- Вычислительный механизм в отличие от функционального элемента (например, регистра) сочетает в себе некоторую функциональность и внутреннее устройство (что делает и как делает)

[] А.Е. Платунов. Теоретические и методологические основы высокоуровневого проектирования встраиваемых вычислительных систем*

Примеры механизмов из обычной жизни

- Механизм ли это?
 - Механизм всегда реализует какую-либо цель в нашей конструкции. Всегда можно спросить: зачем нужна эта штука и что она делает?
- Рычаг – для подъема тяжестей
- Письменность – для запоминания текстов.
- Освещение – для обеспечения органов зрения необходимым количеством света.
- Медицина – для лечения болезней.
- Нож – для разрезания материалов
- Молоток – для заколачивания гвоздей
- Гвоздь – для крепления двух и более элементов друг к другу и т.д.

Примеры механизмов

- Система прерываний процессора – механизм для реализации параллелизма в системе с одним процессором Фон-Неймана
- ПДП – прямой доступ к памяти (DMA, Direct Memory Access) – механизм для реализации доступа каких-либо устройств (обычно системы ввода-вывода) к памяти вычислительной системы без участия процессора.

Механизм и архитектура

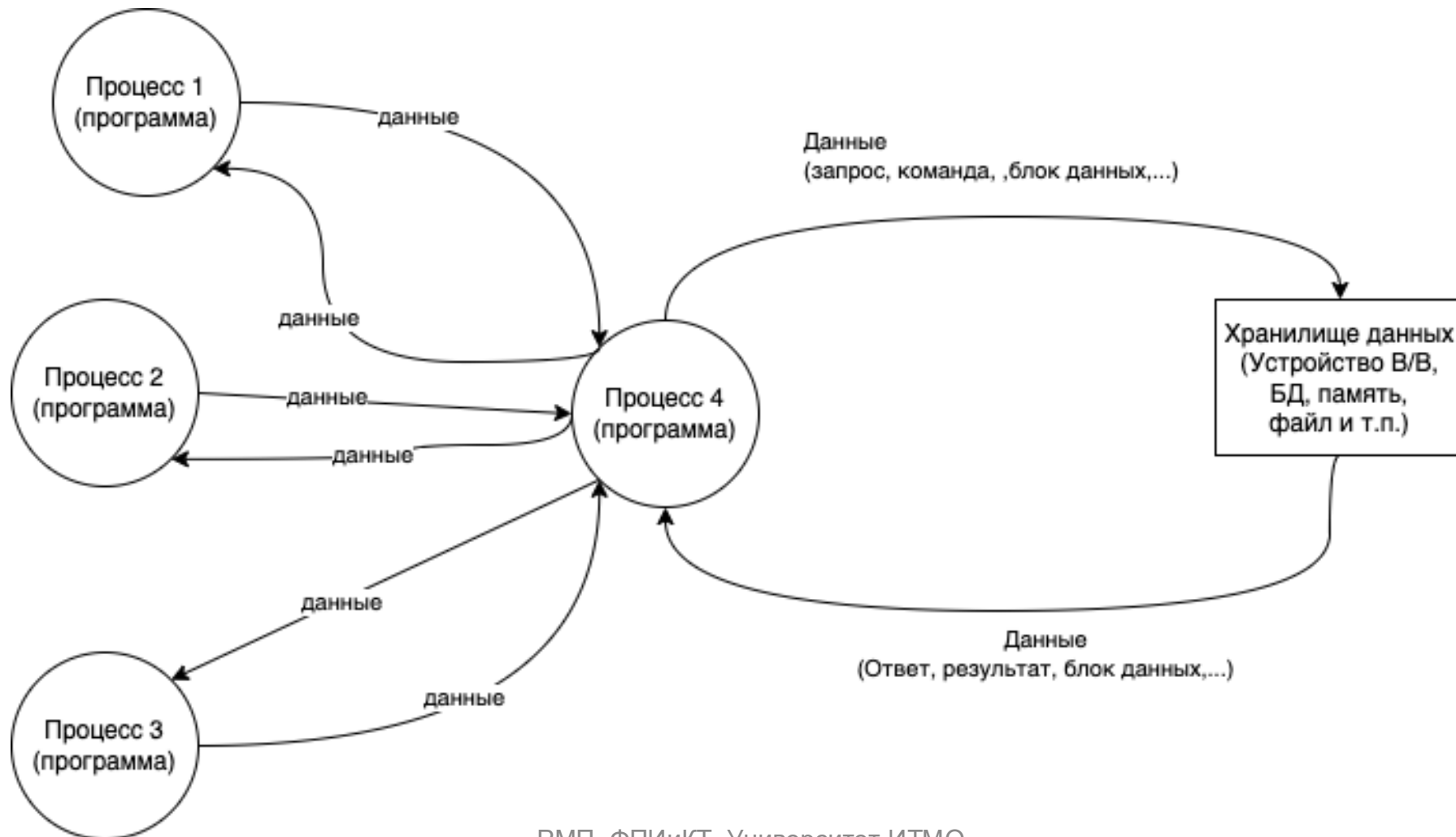
- Механизм может использоваться как метамодель.
- Придумываем механизмы и реализуем их на основе имеющихся паттернов.
- Хороший стиль – механизм делает что-то одно, чтобы в проекте не было «павлино-утко-ежей», совмещающих в себе самые неожиданные свойства (например, не нужно объединять тостер, бетономешалку и телевизор, хотя...)...

Вычислительный процесс *

Вычислительный процесс – процесс преобразования данных вычислительным устройством (вычислителем, вычислительной машиной) в соответствии с заданной функциональностью.

[] А.Е. Платунов. Теоретические и методологические основы высокоуровневого проектирования встраиваемых вычислительных систем*

Пример описания вычислительного процесса на базе Process Network



Что нужно для жизни процесса (PN)?

- Управление
 - Создание, запуск, пауза, останов, удаление
- Работа с очередями
 - Создание, удаление, очистка, проверка наличия данных, чтение и запись данных
 - Чтение из нескольких очередей
- Работа с хранилищами данных (внутри процесса), в рамках PN никаких хранилищ нет.
 - Создание, удаление, запись, чтение, позиционирование

Какие есть средства управления процессами в ОС?

- Создание/удаление
 - Системные вызовы ОС, создание и удаление тредов и корутин в библиотеках ЯВУ
 - Прерывания, сигналы Unix
- Приостановка/запуск
 - Семафоры, критические секции, ожидание данных (например, из очереди), задержка, исключения (приостановка)
- Работа с очередями
 - Каналы, очереди, именованные каналы и т.п.
- Работа с хранилищами
 - Драйверы В/В, файловая система, драйверы БД, общая память

Суть механизма прерываний (Interrupt)

1. Прерывание текущего потока команд.
 2. Передача управления в другое место программы.
 3. Выполнение нового потока команд.
 4. Возврат к старому потоку команд
- Что создает прерывание? Прерывание создает процесс, который может работать вместе с другими процессами на одном физическом процессоре Фон-Неймановского типа.
 - В чем суть этого явления? Это **метасистемный переход** (см. Турчин, «**Феномен науки**») от императивной модели вычислений (модели Фон-Неймана) к модели Kahn Process Network.
 - Мы получаем два уровня абстракции: внизу императивный, наверху - Потоковый.

Задачи операционной системы

- Управление ресурсами вычислительной системы
 - Процессорное время (реализация PN, IPC, планирование ...)
 - Ввод-вывод
 - Память
 - Хранилища данных
 - Системы питания
- Управление прикладными задачами
- Ограничение доступа (безопасность)
- Поддержка UI (не везде)
- Загрузка, замена, обновление прикладных программ

Что такое процесс, поток и корутина с точки зрения ОС?

- Процесс имеет собственное адресное пространство и требует больших накладных расходов на управление (переключение задач, планирование, управление, механизмы IPC). Для работы с процессом требуется использование системных вызовов ОС.
- Поток имеет доступ к данным других потоков, ему доступно все адресное пространство процесса. Накладные расходы на переключение и планирование сравнительно малы, многозадачность вытесняющая. Расходы на IPC малы, так как не требуется обращение к ОС.
- Корутина (сопрограмма) так же как и поток имеет доступ ко всему адресному пространству процесса. Нет затрат на переключение задач и планирование, но многозадачность согласующая.

Что такое процесс, поток и корутина с точки зрения Process Network?

- Нет разницы, с точки зрения модели вычислений PN эти понятия не имеют различий
- Именно поэтому архитектура на таком уровне абстракции инвариантна к реализации (или независима от реализации).
- Что дает независимость от реализации?
 - Нет привязки к деталям реализации, что расширяет спектр возможных решений
 - Модель системы содержит меньше деталей, что упрощает размышления (человеческий мозг очень ограничен и не любит слишком больших моделей!) и уменьшает количество ошибок
- Чем плоха инвариантность?
 - При решении ряда задач, требующих максимального использования ресурсов (реальное время, высокая загрузка) слишком общие архитектурные решения могут не учитывать специфических деталей.

Инструментальный и целевой компьютер в контексте Android

- **Инструментальная система** – система, на котором идет разработка целевого ПО. На этой системе мы разворачиваем Android SDK (с библиотеками, компилятором и отладчиком), Android NDK, IDE, симулятор Android (бывают сторонние разработки), репозиторий исходников AOSP (если мы хотим делать свою версию Android). Инструментарий можно развернуть на ПК с Windows, Linux, Mac OS X, но разработку с использованием AOSP лучше вести в Linux.
- **Целевая система** – система, на которой идет исполнение целевого ПО. Может быть развернута на целевой платформе (смартфоне, планшете, ГУ и т.п.), на прототипной плате (evaluation board, например, на [Raspberry PI](#)), в симуляторе, в специализированных симуляторах (например [GEM5](#)), в аппаратных симуляторах.

Что нужно для работы инструментальной части вычислительной системы?

- Система программирования (написание и компиляция исходных текстов, сборка загрузочных файлов с исполняемым кодом и различными конфигурационными файлами)
- Система доставки исполняемого кода и конфигурационных файлов в целевую систему
- Система приема телеметрии из вычислительной системы, логов и каких либо метрик:
 - Фиксация событий, контроль состояния FSM и т.п.
 - Измерение времени исполнения
 - Контроль последовательности исполнения
 - Контроль потребления ресурсов: (энергопотребление, ОЗУ, хранилище данных, CPU, GPU, ...)
- Система удаленной отладки и тестирования в целевой системе, Отладочный интерфейс для пошаговой отладки, остановки, запуска и просмотра переменных и памяти.
- Симуляция целевой системы для упрощения процесса отладки и тестирования

Что нужно для работы целевой системы

- **Загрузчик** – инициализирует систему после холодного или горячего старта (читает исполняемый код из хранилища данных, загружает его в ОЗУ, инициализирует и передает управление), обычно содержит в себе средства для замены операционной системы.
- **HAL** – уровень абстракции от аппаратуры, упрощает перенос операционной системы на разные аппаратные платформы.
- **Операционная система** – управляет ресурсами вычислительной системы (содержит в себе драйверы устройств, различные службы, систему для поддержки прикладных программ, систему загрузки прикладных программ, систему отладки прикладных программ)
- **Прикладные программы** - решают прикладные задачи как в обычном компьютере.

Попробуем понять из чего сделан Android...

Цели создания Android

- **Простота:** Пользовательский интерфейс должен быть простым и понятным, без лишних элементов и сложностей.
- **Настраиваемость:** Пользователь должен иметь возможность настраивать интерфейс под свои нужды и предпочтения.
- **Универсальность:** Интерфейс должен работать одинаково на всех устройствах и версиях Android, независимо от их размера, разрешения экрана и других характеристик.
- **Адаптивность:** Интерфейс должен адаптироваться к различным размерам экранов и ориентациям устройств, чтобы обеспечить удобство использования.
- **Эффективность:** Использование ресурсов устройства должно быть эффективным, чтобы обеспечить стабильную работу приложений и системы в целом.

В чем суть Android?

- Android это обычная операционная система, специфика которой заключается в ориентации на оперативную работу с сенсорным экраном сравнительно небольшого размера (например, на ходу, на диване, на складе, в автомобиле и т.п.).
- Android представляет собой совокупность весьма старых технологий, объединенных в рамках «современной» платформы.
 - Android Studio – Eclipse (2001), потом IntelliJIdea (2001)
 - Java (1995), затем Kotlin (2011)
 - Android (2003)
 - Linux (1991)
 - Dalvik/ART, построенный почти как JVM (1994),
 - Zygote, запуск прикладных задач в песочнице (VM) - аналогичный подход в Inferno (1995)
 - Средства IPC на базе Binder (BeOS 1990)
 - UTF-8 – Plan 9 (конец 80-х)
 - Content provider – менеджер ресурсов
 - Intent – plumber Plan 9 (конец 80-х)
- Другими словами, никаких особых технологических прорывов в Android нет.

В чем главная специфика Android?

- Массовость!
 - 3 миллиарда пользователей в мире
 - [70% рынка Android, 30% iOS](#)
 - В мире много прикладных программистов Android (в 2022 было около 5 миллионов). Сейчас в мире около 30 миллионов программистов (пока еще каждые 5 лет количество программистов удваивается).
 - Производители инструментального ПО стараются максимально упростить (и удешевить) процесс разработки, вовлекая в индустрию все больше плохо подготовленных специалистов. Программирование чистого Android с внедрением Kotlin и Jetpack Compose сильно упростилось, есть много фреймворков на базе Web-технологий и на базе C#, которые помогают «перетянуть» программистов со сторонних платформ.

Механизмы Android

- Система программирования
- Средства отладки и профилировки
- Среда исполнения и организация вычислительного процесса
 - Zygote, ядро Linux, ART
 - Процессы Linux, Linux IPC, права доступа Linux
 - Android:
 - Права доступа и безопасность
 - Ввод-вывод и сеть
 - Android IPC (Inter-Process Communication)
 - Средства реализации UI (Jetpack compose и т.д.)

Основные механизмы, доступные программисту

- CPU – средство для поддержки инструментария программирования, основанного на идее машины фон-Неймана
- Не фоннеймановские вычислители:
 - Сигнальные процессоры – обработка аналоговых сигналов (звук)
 - GPU – средство для ускорения 2D и 3D графики
 - Нейропроцессор, тензорный процессор, систолические вычислители – ускорение операций над матрицами для реализации искусственных нейронных сетей
- HAL – средство для упрощения портирования ядра Linux и драйверов устройств на разные аппаратные платформы
- Классическая ОС – стандартизация реализации драйверов нижнего уровня
- Песочницы – средства разделения приложений для обеспечения информационной безопасности
- Виртуальная машина ART – реализация кроссплатформенности для обеспечения унификации ПО для очень разного аппаратного обеспечения и привязка технологических цепочек Java/JVM из «кроватьного энтерпрайза» (информационных систем) к мобильной платформе
- Activity – как способ реализации GUI
- Хранилища данных – (Shared Preferences - пары ключ/значение, SQL базы, файловая система)
- Средства обмена данными
 - Intent – низкоуровневый механизм обмена сообщениями между Activity и различными службами.
 - Content provider – механизм обмена высокоуровневой информацией (настройки, списки контактов, записи календаря и т.п.)
 - Уведомления
- Средства сетевого взаимодействия
- Управление правами доступа